International Journal of Computer Games Technology

Artificial Intelligence for Computer Games

Guest Editors: Abdennour El Rhalibi, Kok Wai Wong, and Marc Price

Artificial Intelligence for Computer Games

International Journal of Computer Games Technology

Artificial Intelligence for Computer Games

Guest Editors: Abdennour El Rhalibi, Kok Wai Wong, and Marc Price

Contents

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 251652, 3 pages
doi:10.1155/2009/251652

Editorial
Artificial Intelligence for Computer Games

Abdennour El Rhalibi,[1] Kok Wai Wong,[2] and Marc Price[3]

[1] *School of Computing and Mathematical Sciences, Liverpool John Moores University, Byrom Street, Liverpool L3 3AF, UK*
[2] *School of Information Technology, Murdoch University, 90 South Street, Murdoch WA 6150, Australia*
[3] *BBC Research, Kingswood Warren, Tadworth, Surrey KT20 6NP, UK*

Correspondence should be addressed to Abdennour El Rhalibi, a.elrhalibi@ljmu.ac.uk

Received 15 December 2008; Accepted 15 December 2008

Artificial intelligence (AI) in computer games covers the behaviour and decision-making process of game-playing opponents (also known as nonplayer character or NPC). Current generations of computer and video games offer an amazingly interesting testbed for AI research and new ideas. Such games combine rich and complex environments with expertly developed, stable, physics-based simulation. They are real-time and very dynamic, encouraging fast and intelligent decisions. Computer games are also often multiagents, making teamwork, competition, and NPC modelling key elements to success. In commercial games, such as action games, role-playing games, and strategy games, the behaviour of the NPC is usually implemented as a variation of simple rule-based systems. With a few exceptions, machine-learning techniques are hardly ever applied to state-of-the-art computer games. Machine-learning techniques may enable the NPCs with the capability to improve their performance by learning from mistakes and successes, to automatically adapt to the strengths and weaknesses of a player, or to learn from their opponents by imitating their tactics.

In this special issue, we introduce a number of interesting papers contributing to a wide range of these topics and reflecting the current state of AI for Computer Game in academia. A total of 20 papers have been submitted to this special issue, of which 9 high-quality papers have been accepted after the peer review process.

This special issue starts with the first paper "Performance simulations of moving target search algorithms" by Peter Kok Keong Loh et al. In this paper, the authors focused on the design of moving target search (MTS) algorithms for computer generated bots. MTS algorithms pose important challenges as they have to satisfy rigorous requirements which involve combinatorial computation and performance. In this paper, the authors investigate the performance and behaviour of existing moving target search algorithms when applied to search-and-capture gaming scenarios. As part of the investigation, they also introduce a novel algorithm known as abstraction MTS. They conduct performance simulations with a game bot and moving target within randomly generated mazes of increasing sizes and reveal that abstraction MTS exhibits competitive performance even with large problem spaces.

The second paper is proposed by Julio Clempner and is entitled "A shortest-path lyapunov approach for forward decision processes." In this paper, the author presents a formal framework for shortest-path decision process problem representation. Dynamic systems governed by ordinary difference equations described by Petri nets are considered. The trajectory over the net is calculated forward using a discrete Lyapunov-like function. Natural generalizations of the standard outcomes are proved for the deterministic shortest-path problem. In this context, the authors are changing the traditional cost function by a trajectory-tracking function which is also an optimal cost-to-target function for tracking the net. This makes an important contribution in the conceptualization of the problem domain. The Lyapunov method introduces a new equilibrium and stability concept in decision process for shortest path.

The third paper is entitled "Fractal analysis of stealthy pathfinding aesthetics" authored by Coleman Ron. In this paper, the author uses a fractal model to analyze aesthetic values of a new class of obstacle prone or "stealthy" pathfinding which seeks to avoid detection, exposure, and

openness in videogames. This study is interesting since in general the artificial intelligence literature has given relatively little attention to aesthetic outcomes in pathfinding. The data reported, according to the fractal model, suggests that stealthy paths are statistically significantly unique in relative aesthetic value when compared to control paths. The author also shows that paths generated with different stealth regimes are also statistically significantly unique. These conclusions are supported by statistical analysis of model results on experimental trials involving pathfinding in randomly generated, multiroom virtual worlds.

The next paper is proposed by Frank Dignum et al. and discusses "Games and agents: designing intelligent Gameplay." Multiagent system research offers a promising technology to implement cognitive intelligent NPC's. However, the technologies used in game engines and multiagent platforms are not readily compatible due to some inherent differences of concerns. Where game engines focus on real-time aspects and thus propagate efficiency and central control, multiagent platforms assume autonomy of the agents. Increased autonomy and intelligence may offer benefits for a more compelling gameplay and may even be necessary for serious games. However, it raises problems when current game design techniques are used to incorporate state of the art multiagent system technology. In this paper, the authors focused on three specific problem areas that arise from this difference of view: synchronization, information representation, and communication. They argue that current attempts for integration still fall short on some of these aspects. They show that to fully integrate intelligent agents in games, one should not only use a technical solution, but also a design methodology such as OperA, that is amenable to agents.

The fifth paper presents "A multiagent potential fields based bot for real-time strategy games" authored by Johan Hagelback et al. The paper discusses bots for real-time strategy (RTS). A bot controls a number of units that will have to navigate in a partially unknown environment, while at the same time avoid each other, search for enemies, and coordinate attacks to fight them down. "Potential fields" is a technique originating from the area of robotics where it is used in controlling the navigation of robots in dynamic environments. The authors present a multiagent potential field based bot architecture which is evaluated in two different real-time strategy game settings and compare it, in terms of performance, and configurability, to other state-of-the-art solutions. The authors show that the solution is a highly configurable bot which can match the performance of traditional RTS bots. They also show that a multiagent potential field-based bot is highly competitive in a resource gathering scenario.

The next paper is "Combining artificial intelligence methods for learning bots in a real time strategy game" authored by Robin Baumgarten et al. The authors describe an approach to simulate human game-play in strategy games using a variety of AI techniques, including simulated annealing, decision tree learning, and case-based reasoning. They have implemented an AI-bot that uses these techniques to form a novel approach to plan fleet movements and attacks in DEFCON, a nuclear war simulation strategy game released in 2006 by Introversion Software Ltd, Surrey, UK. They describe how the AI-bot operates, and the experimentation they have performed in order to determine an optimal configuration for it. With this configuration, the proposed AI-bot beats Introversion's finite state machine automated player in 76.7% of 150 matches played.

The seventh paper is "Enhancing artificial intelligence on a real mobile game" by Fabio Aiolli et al. Mobile gaming represents a killer application that is attracting millions of subscribers worldwide; yet, several technical issues in this context remain unsolved. One of the aspects crucial to the commercial success of a game is ensuring an appropriately challenging artificial intelligence (AI) algorithm against which to play. However, creating this component is particularly complex as classic search AI algorithms cannot be employed by limited devices such as mobile phones or, even on more powerful computers, when considering imperfect information games (i.e., games in which participants have not a complete knowledge of the game state at any moment). In this paper, the authors propose to solve the imperfect information game issue by resorting to a machine learning algorithm which uses profiling functionalities in order to infer the missing information, and making the AI able to efficiently adapt its strategies to the human opponent. They studied a very simple and computationally light machine learning method that can be employed with success, enabling AI improvements for imperfect information games even on mobile phones. They present results on a simple game called Ghosts which show the ability of their algorithm to quickly improve its own predictive performance as the number of games against the same human opponent increases. A mobile phone-based version of the game has been also created which can be played either against another player or against the AI algorithm.

The eighth paper is "Breeding terrains with genetic terrain programming—the evolution of terrain generators" by Miguel Frade et al. Although a number of terrain generation techniques have been proposed during the last few years, all of them have some key constraints. Modelling techniques depend highly upon designer's skills, time, and effort to obtain acceptable results, and cannot be used to automatically generate terrains. The simpler methods allow only a narrow variety of terrain types and offer little control on the outcome terrain. The Genetic Terrain Programming technique, proposed, based on evolutionary design with genetic programming, allows designers to evolve terrains according to their aesthetic intentions or desired features. This technique evolves terrain programmes (TPs) that are capable of generating a family of terrains - different terrains that consistently present the same morphological characteristics. This paper presents a study about the persistence of morphological characteristics of terrains generated with different resolutions by a given TP. Results show that it is possible to use low resolutions during the evolutionary phase without compromising the outcome and that terrain macrofeatures are scale invariant.

Finally, the last paper is "Fine-tuning parameters for emergent environments in games using artificial intelligence"

authored by Vishnu Kotrajaras et al. This paper presents the design, development, and test results of a tool for adjusting properties of emergent environment maps automatically according to a given scenario. Adjusting properties for a scenario allows a specific scene to take place while still enables players to meddle with emergent maps. The tool uses genetic algorithm and steepest ascent hill-climbing to learn and adjust map properties. The authors shows that using the proposed tool, the need for time consuming and labour-intensive parameter adjustments when setting up scenarios in emergent environment maps, is greatly reduced. The tool works by converting the paths of events created by users for a map to the properties of the map that plays out the scenario set by the given paths of events. Test results show good properties preservation.

Abdennour El Rhalibi
Kok Wai Wong
Marc Price

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 745219, 6 pages
doi:10.1155/2009/745219

Research Article

Performance Simulations of Moving Target Search Algorithms

Peter K. K. Loh[1] and Edmond C. Prakash[2]

[1] *Division of Computer Science, School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798*
[2] *Department of Computing and Mathematics, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, UK*

Correspondence should be addressed to Peter K. K. Loh, askkloh@ntu.edu.sg

Received 9 April 2008; Revised 9 July 2008; Accepted 25 September 2008

Recommended by Kok Wai Wong

The design of appropriate moving target search (MTS) algorithms for computer-generated bots poses serious challenges as they have to satisfy stringent requirements that include computation and execution efficiency. In this paper, we investigate the performance and behaviour of existing moving target search algorithms when applied to search-and-capture gaming scenarios. As part of the investigation, we also introduce a novel algorithm known as abstraction MTS. We conduct performance simulations with a game bot and moving target within randomly generated mazes of increasing sizes and reveal that abstraction MTS exhibits competitive performance even with large problem spaces.

1. Introduction

In most RPG/adventure-based computer games, different types of bots act as adversaries to players. For example, in the recently launched Hellgate [1], players need to defend against and fight computer-generated demonic hordes. In such games, each generated bot is typically incorporated with suitable algorithms that enable it to locate and move towards a player. Each bot also has a "detection range or area" within which, it can detect a player. Unlike existing algorithms for static targets [2, 3], algorithmic designs for moving target search (MTS) algorithms are inherently more involved. The computational and memory requirements are significant. In some computer games, search algorithms can take up as much as 70% of CPU time [4–6]. This is due to the large number of objects (e.g., player, NPC, building, and walls) that need to be taken into consideration in the game environment [7]. The computational and memory requirements are also high when multiple bots communicate to find strategic paths as shown in our earlier work on Team AI [8]. Graphics also consume a significant proportion of computational resources leaving a limited amount for game AI [4]. Many contemporary graphics-intensive computer games are real-time, however, which means that bot responses to a player must be made as soon as possible. Such a scenario poses conflicting demands on the design of MTS algorithms.

This paper presents a study of the performance and behaviour of existing moving target search (MTS) algorithms in a maze search-and-capture scenario. As part of the study, we include a novel MTS algorithm called *abstraction MTS* and evaluate its performance and behaviour against the existing algorithms. Section 2 of this paper reviews three existing widely used MTS algorithms. Section 3 states the definitions and notations on which subsequent sections are based. The design of the abstraction MTS algorithm is detailed in Section 4. Section 5 describes the performance and behavioural analyses. The paper concludes with Section 6 followed by the Acknowledgments and References.

2. Survey

In a contemporary player-bot engagement-based computer game, the bot's response and behaviour are designed to be as realistic as possible. For example, a bot would be able to sense (detect) a player within its visibility region and not beyond. To make the game more engaging and playable, a typical bot should not be able to detect beyond some

finite region. Deep look-ahead search techniques that would be useful in certain games like chess would add an unfair advantage to a bot's capabilities and reduce the engagement and playability of the game [9]. In our work, therefore, we focus on algorithm designs that exploit "neighbourhood" information—information that can be determined within a finite detection region surrounding a bot. In this section, we review three well-known existing MTS algorithms for moving targets: basic moving-target search, weighted moving target search, and commitment and deliberation moving target search algorithms.

2.1. Basic Moving Target Search. The basic moving target search (BMTS) algorithm [10] is a generalisation of the learning real-time A* algorithm [11]. A matrix of heuristic values is maintained during the search process to improve their accuracy. The upper bounds on space and time complexities of B-MTS are N^2 and N^3, respectively, where N is the number of states in the problem space. Although MTS could converge to an optimum path (solution) eventually, it suffers from *heuristic depression* [10], which is a set of states with heuristic values not exceeding those of all neighbouring states. This may occur since heuristic value updates are localised leaving state inaccuracies over other areas in the problem space. In a heuristic depression, an agent repeatedly traverses the same subset of neighbouring states without visiting the rest. The agent may also continue to look for a shorter path even though a fairly good path to the target has been found. This would incur additional computational overheads and reduce bot performance during game play.

2.2. Weighted Moving Target Search. In certain scenarios, an optimal solution may not be needed and suboptimal paths may be found in a shorter time. The weighted moving target search (WMTS) algorithm [12] reduces the amount of exploration in MTS and accelerates convergence by producing a suboptimal solution. It allows a suboptimal solution with ε-error and δ-search (real-time search with upper bound) to achieve a balance in solution path quality and exploration cost. During the search, heuristic values are brought as close as possible to, but not reaching, the actual values. So, there is no guarantee that the search will eventually converge to an optimal solution. It is also important to determine a value of δ such that it can restrain exploration and find better solutions. The amount of memory space increases as δ increases.

2.3. Commitment and Deliberation Moving Target Search. With the commitment and deliberation moving target search (CDMTS) algorithm [10], the agent may ignore some of the target's moves. The agent only updates the target's moves when the agent is not in a heuristic depression. The *commitment* to the current target state increases if the agent moves in a direction where the heuristic value is reducing. If the agent is in a depression, it ignores the target's moves and commitment is set to 0. During *deliberation*, real-time search is performed when heuristic difference decreases, and offline search is performed when the agent is in heuristic depression. The offline search is used to determine the boundary of the heuristic depression. The CDMTS algorithm improves upon the efficiency of BMTS since the agent can exit from the heuristic depression faster.

3. Preliminaries

To simplify the problem, we prohibit movements in the third dimension (e.g., jumping, climbing) by either the bot or the player. The problem space is then reduced to that of a two-dimensional (2D) region, whereby movements of both bot and player are restricted to *left*, *right*, *forwards,* and *backwards*. We also require that the size of the problem space can be varied with obstacles generated and placed randomly. The unobstructed locations (no obstacles) in the maze are defined as a set of *states* and all traversals between a state and neighbouring states are defined by a set of edges with edge cost = 1.

In the following, we will use the terms "agent" or "bot" and "target" or "player" interchangeably. From each state, the agent or target can move to any of a maximum of four neighbouring states (representing locations to the *left*, *right*, *forward,* and *backward* directions) if unobstructed. The target moves randomly and slower than the agent so that the target will be acquired in a finite time. The goal for the agent is then to find a path from starting state s to the current target state g, if there is at least one path from s to g. The goal is accomplished if both agent and target occupy the same state. We define the following:

s = current state;

g = goal state;

s' = other state (not s or g);

$succ(s)$ = the set of successor states of s (neighbour states of state s);

$j(a, b)$ = total edge cost from state a to state b;

$h(a, b)$ = heuristic value from state a to state b;

$h^*(a, b)$ = minimal heuristic value from state a to state b considering all alternative paths;

$f(s, g) = j(s, s') + h(s', g)$, where f is the computed cost of a path from s to g.

To guarantee the completeness of the algorithm, we assume that the minimum heuristic value is never overestimated, that is, $h(a, b) \leq h^*(a, b)$ [10]. This is the case in the previous 3 algorithms surveyed. Information that includes the maze configuration, target position, and target movement pattern are not available initially. As shown in Figure 1, the agent can only detect the target's position if the target is within detection range r, regardless of whether there is an obstacle between them.

To simplify the calculation, the detection area is represented as a square and the value r is greater than one to emulate a bot equipped with above average human player's sensory-detection capabilities. For comparison purpose, the target (human) may be assumed to have a detection area with $r = 1$. This is typical in the more challenging

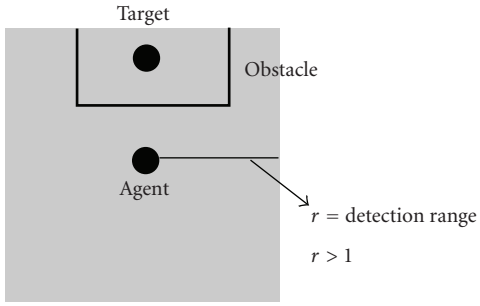
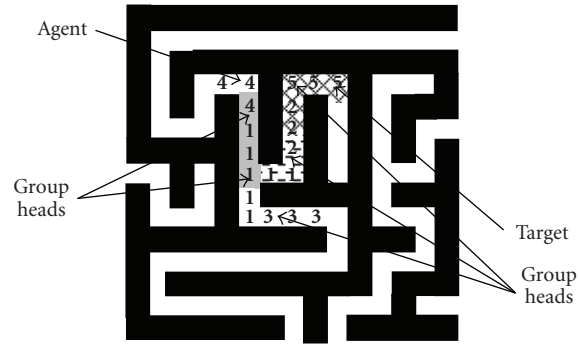


Figure 1: Target is detected within range.



Real move list 1
Real move list 2
Real move list 3

Figure 2: Abstraction maze example.

contemporary games. In the case of BMTS, WMTS, and CDMTS, the agent does not know the value of $h^*(a, b)$ until it finds an optimal solution. A path $(s_0, s_1, \ldots, s_n)$ is optimal if and only if $h(s_i) = h^*(s_i)$ for $0 \leq i \leq n$, where $h^*(s_i)$ equals the actual cost from s to the goal and $h(s_i)$ equals the heuristic value. Heuristic value is computed with the Manhattan distance method.

4. Abstraction Moving Target Search Algorithm Design

Each of the existing MTS algorithms employs a heuristic array table in its learning process. For each state s, we need to store the heuristic value with all states. That is, we need to store the state pairs: $h(s, k)$, where $k \in S$ (problem state space) except $h(s, s) = 0$. Group values are stored in a 2D-array, *abstract*. Also, since $h(x, y) = h(y, x)$, the total memory needed is upper-bounded by $(n^2 - n)/2$, where n is the number of states in S. To solve this problem, we apply the *abstraction maze* approach. Our approach is to have a 2-level search as illustrated by an example of an abstraction maze in Figure 2.

Figure 2 shows that when the agent detects the target, it checks if the target position belongs to a group. If so, the agent will determine the best *abstraction move list AL* and required *real move list(s) RL* to acquire the target. In this example, AL = $\{4, 1, 2, 5\}$. Using group numbers to simplify representation, three real move lists would be generated as follows: real move list 1 = $\{4, 1, 1, 1\}$, real move list 2 = $\{1, 1, 2\}$, and real move list 3 = $\{2, 2, 5, 5, 5\}$. Each *real move list* contains a movement path up to the next group head and the last leading up to the target.

Specifically, each node x in the abstraction maze may be labeled with a number which indicates an associated group, gr_x. These group values are stored in a 2D-array labeled *abstract*. A group p also has a *group head*, $hd(p)$. The group head is used as a base to measure the cost (distance) to all nodes in the same group. The distance from a node s to its group head, gr_s, must be less than some constant *abstractDistance*. That is, $j(s, hd(gr_s)) < abstractDistance$. In the example in Figure 2, *abstractDistance* is equal to 3. Each group also maintains a list of its neighbours. For example, the neighbours of group 1 are groups 2, 3, and 4.

In the initialisation step of the algorithm, the starting point of the agent is set at the location of the head of group 1 and *number AbstractNode* is set to 1. The variable *number AbstractNode* is defined as the number of groups that has been created. During exploration, after the agent has moved to a new state (position) s', it will check if this node has a group. If it has not, the agent will check if the nearest head distance is less than *abstractDistance*. If it is, then apply *setAbstract* method. In *setAbstract* method, the current node will be grouped with the nearest group head. If the nearest head distance is not less than *abstractDistance*, then *number AbstractNode* is increased by one and the current node will become a new group head. Formally, this is expressed as follows.

For each $s' \notin gr_i$, (where $gr_i \in Abstract$ and $1 \leq i \leq |Abstract|$):

$j(s', hd(gr_i)) < abstractDistance) \Rightarrow setAbstract\ (s', gr_i)$;

$j(s', hd(gr_i)) \geq abstractDistance) \Rightarrow (gr_k = number\ AbstractNode + 1 \wedge hd(gr_k) = s')$.

The above process then repeats with the new group gr_k.

The abstraction moving target search (AMTS) algorithm is shown in Figure 3. The *abstraction move list* guides the agent's movement sequence in the abstraction maze. The *real move list* guides the agent's movement sequence in the original (unabstracted) maze. Variable *detectTarget* denotes if the agent currently detects the target and *exploreLocation* denotes the nearest node location which does not belong to any group. If the agent does not detect the target, it will move according to the last generated *real move list* and complete the moves in this list. If the target has been acquired at the completion of moves in the real move list, *withinRange* is set to false and the run ends. Otherwise, the algorithm is repeated with the next detection of the target by the agent. If the real move list is empty, the algorithm attempts to generate

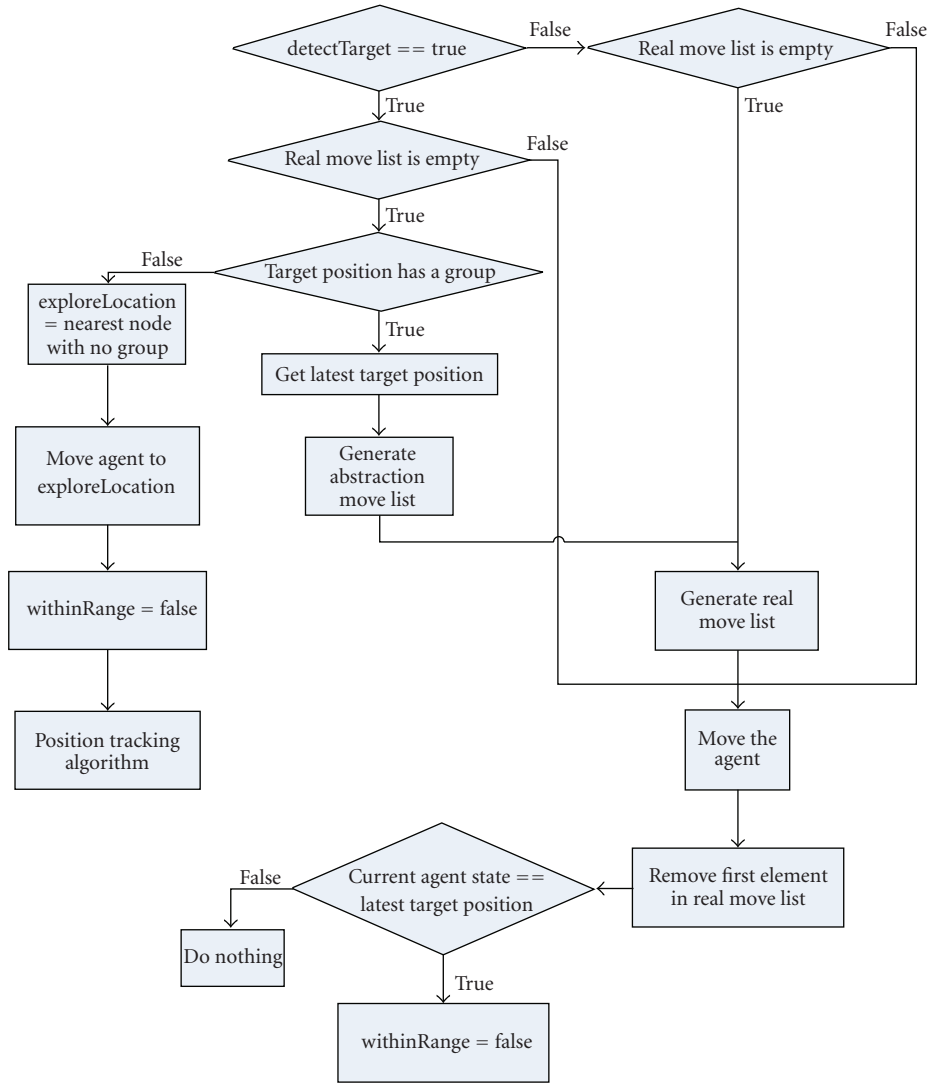


Figure 3: Abstraction MTS algorithm.

one, based on the last known abstraction maze position of the agent. The agent then follows the moves in this generated *real move list*.

However, if the agent detects the target, *withinRange* is set to true and the target position t is checked for association with a group. If it is, the agent will generate an *abstraction move list* (with the latest target position) and *real move list*. Each *real move list* contains a movement path up to the next group head and the last *real move list* with a path up to the target. Both abstraction and real move lists are then used to guide the agent's movement to acquire the target. When agent state is the same as the target position, withinRange is set to false so that the next search-acquisition cycle can begin. If, on the other hand, the target position is not associated with a group, the agent finds the nearest node with no group and continues with exploration from this location using the position tracking algorithm. Formally, this is expressed as follows.

While (*withinRange* = TRUE):

for all $gr_{s'}$, where $gr_{s'} \in succ(gr_s)$, find $f_{\min}(gr_{s'}, gr_t) = j(gr_{s'}, gr_t) + h(gr_{s'}, gr_t)$;

for $f_{\min}(gr_{s'}, gr_t) \Rightarrow \mathrm{AL} = \{gr_s, gr_{s'}, \ldots, gr_t\} \wedge \mathrm{RL} = \{s, s_1, s_2, \ldots hd(gr_{s'})\} \Rightarrow$ repeat generating RL for next group till $s_k = t$, where $s_k \in gr_t$.

Abstraction MTS does not employ heuristic values and, therefore, does not learn. To reduce first move decision latency, each *real move list* is generated after the previous one has been traversed. This process continues until the agent arrives at the target location. When the agent traverses a *real move list*, it will ignore the target's move. It only updates the target's position when it generates a new *real move list*. In this way, it will be faster to search for the target with lower memory requirements.

5. Performance Simulation and Analysis

The simulation was conducted for 6 different maze sizes: 50×50, 100×100, 150×150, 200×200, 250×250, and 300×300. All algorithms had the same starting points for their agent and target in the same maze. The starting points of agent and target are random and the heuristic distance between their starting points is at least half of the diagonal length of the maze. For each maze, every algorithm was executed 100 times and the average of the results was recorded.

5.1. Results. The results show that the degree of learning required in the algorithm depends on the target game application. For turn-based games like chess and weiqi, compute-intensive learning algorithms based on heuristics are effective propositions. However, for real-time action games where expected responses are in seconds or milliseconds, compute-intensive learning approaches significantly degrade playability. For real-time MTS gaming scenarios, in specific, an adaptive algorithm, with minimal or no learning but favouring faster acquisition, proves a more viable solution.

Figure 4 shows an example of a maze used in our experiments. The black portions represent the pathways in the maze. In this particular maze, there are two entry/exit points (top-left and bottom-right). White portions indicate walls in the maze. Each maze is essentially a square $n\times n$ grid.

Since all algorithms incorporate the same position tracking routine, the number of exploration moves is similar for all algorithms. In position tracking routine, the agent only needs to check the value of its neighbour. So, the other four algorithms should have a constant time, independent of maze size. Figure 5 shows the number of agent steps taken by the moving target search algorithm to acquire the target. The number of moves required for the learning process depends on the difference between the heuristic value and the actual value. As this value difference increases, the agent takes more steps to update the heuristic value. Weighted MTS incurs additional moves to find alternate path to the target. Because of this, weighted MTS has the worst performance in a perfect maze. Commitment MTS has the second best performance. It can be explained by the behaviour of the target that moves randomly. Because of that, the target will not move far away from initial position. So, it will be better for the agent to ignore some of target move to reduce learning process.

Each point on the graph represents an average of 100 runs of the AMTS algorithm. Although the agent and target start at the same locations for each run, the target moves randomly. In some of the runs, it is possible that the target approaches the agent more closely leading to an acquisition with less moves. The target's movement can also be influenced to an extent by the maze topology generated. The maze in Figure 4, for example, has a number of linear pathways without many junctions with the result that the target may have a net effect of moving along one dimension without deviation, leading it closer to the agent despite starting further away. On the whole, therefore, the average number of moves may drop even when the maze size increases. This is shown in Figure 5, from 100 to 150 nodes as well as from 250 to 300 nodes. As the problem space increases, however, the overall trend shown by the AMTS algorithm is still an increasing number of moves.

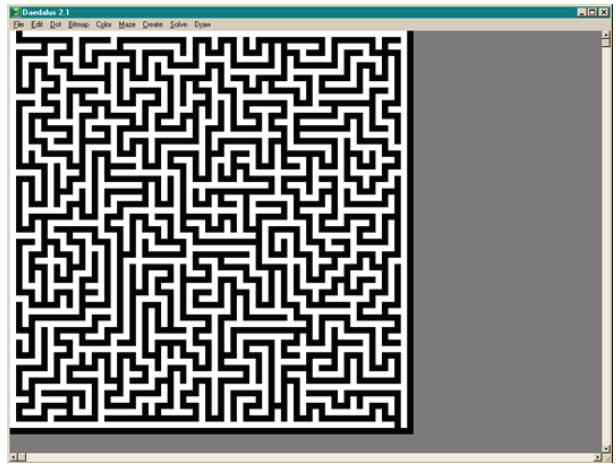


Figure 4: Two-dimensional maze generated by Daedalus program [13].

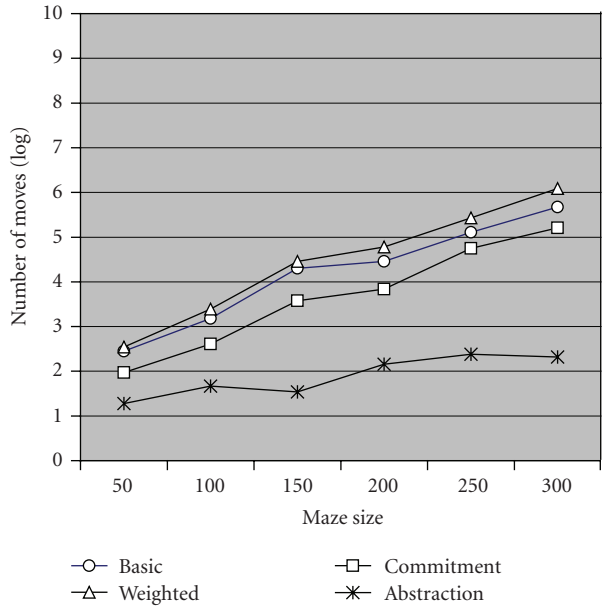


Figure 5: Target acquisition moves comparison.

Figure 6 shows the maximum time required for each movement in the moving target search algorithm. It is known that BMTS, WMTS, and CDMTS have $O(1)$ computation complexity and $O(n^2)$ memory requirements. The upper bound for computation complexity is $O(p^k)$; where p is the actual path length and k is number of branching in a node (constant value) [8]. Hence, the performance of these algorithms scales exponentially with increasing maze size. AMTS, however, has computation complexity that depends on the maze structure and the actual path length. The computation complexity is also greatly reduced by introducing abstraction maze and computing partial path

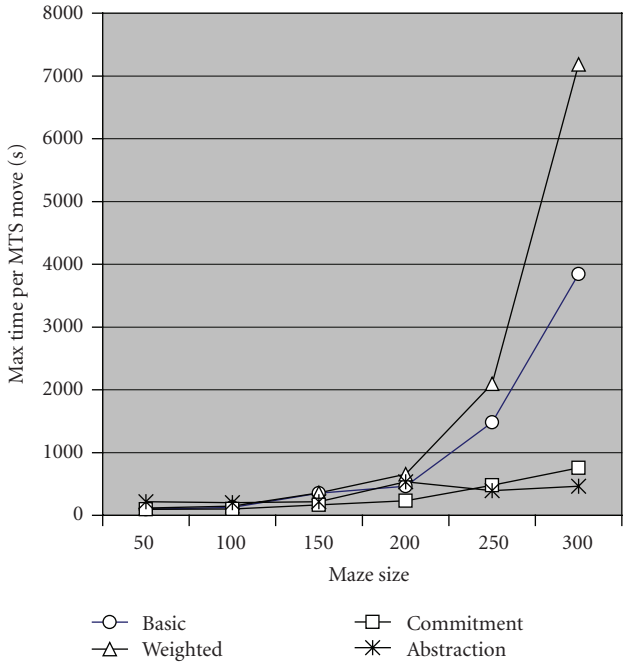


Figure 6: Maximum time per MTS move.

based on neighbouring abstract group information instead of problem space-wide heuristic arrays. This results in the performance of AMTS being linear rather than exponential. As shown in Figure 6, the increase rate for computation complexity in AMTS is on average linear, and the upper bound of memory storage in AMTS is $O(n)$; where n is number of states.

Since the target movement is random, there could be instants in some runs when the target leaves the detection range of the agent while it is executing either "abstracted" or "real" moves. As a result, the agent switches back to exploration with the position tracking algorithm before it can complete the movements that lead to target acquisition. This invariably leads to a higher overhead in decision time per move even though the target starts off closer to the agent in a smaller maze. However, since the target moves slower than the agent, these "irregularities" do not occur often. Figure 6 shows such an irregularity occurring from 200 to 250 nodes before the max time per move increases again.

6. Conclusion

This paper compared and analysed several variants of the MTS algorithm. The main focus of performance, such as effectiveness of learning and speed of response, has been compared with various algorithms. Overall, abstraction MTS has the best performance. It may have the highest exploration move, but it has the lowest MTS move. However, there are some weaknesses of abstraction MTS that we will be studying.

(1) It is difficult to determine *abstractDistance* correctly, especially when the agent does not know the size of maze. *AbstractDistance* is the distance from one node to its group head.

(2) As the maze size increases, it will take a longer time to generate the movement path.

(3) Abstraction MTS may not generate optimal path since it computes complete path in abstraction level, not in actual maze.

Acknowledgment

The authors gratefully thank and acknowledge the critique from the anonymous reviewers that have significantly improved the presentation, organisation, and content of this manuscript.

References

[1] Hellgate Alliance, "The Official Hellgate: London Community Site for South East Asia, offering community, news, information, contests, guides, and more," September 2008, http://hellgate.iahgames.com/.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press and McGraw-Hill, Boston, Mass, USA, 2nd edition, 2001.

[3] A. J. Patel, "Amit's Game Programming," September 2006, http://theory.stanford.edu/~amitp/GameProgramming/.

[4] D. C. Pottinger, "Terrain analysis in realtime strategy games," in *Proceedings of Computer Game Developers Conference (CGDC '00)*, 2000.

[5] S. Koenig, "A comparison of fast search methods for real-time situated agents," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '04)*, vol. 2, pp. 864–871, IEEE Computer Society, New York, NY, USA, August 2004.

[6] T. Ishida and M. Shimbo, "Improving the learning efficiencies of realtime search," in *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI '96)*, vol. 1, pp. 305–310, Portland, Ore, USA, August 1996.

[7] S. Rabin, *AI Game Programming Wisdom*, Charles River Media, Rockland, Mass, USA, 2002.

[8] T. C. H. John, E. C. Prakash, and N. S. Chaudhari, "Strategic team AI path plans: probabilistic pathfinding," *International Journal of Computer Games Technology*, vol. 2008, Article ID 834616, 6 pages, 2008.

[9] V. Bulitko and G. Lee, "Learning in real-time search: a unifying framework," *Journal of Artificial Intelligence Research*, vol. 25, pp. 119–157, 2006.

[10] T. Ishida and R. E. Korf, "Moving-target search: a real-time search for changing goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, pp. 609–619, 1995.

[11] R. E. Korf, "Artificial intelligence search algorithms," in *Handbook of Algorithms and Theory of Computation*, CRC Press, Boca Raton, Fla, USA, 1999.

[12] M. Shimbo and T. Ishida, "Controlling the learning process of real-time heuristic search," *Artificial Intelligence*, vol. 146, no. 1, pp. 1–41, 2003.

[13] Think Labyrinth: Computer Mazes, September 2008, http://www.astrolog.org/labyrnth/daedalus.htm.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 162450, 12 pages
doi:10.1155/2009/162450

Research Article

A Shortest-Path Lyapunov Approach for Forward Decision Processes

Julio B. Clempner[1,2]

[1] *Center for Computing Research, National Polytechnic Institute, Avenue Juan de Dios Batiz s/n, Edificio CIC, Col. Nueva Industrial Vallejo, 07738 Mexico City, Mexico*
[2] *Center for Applied Science and High Technology Research, National Polytechnic Institute Legaria 69 Col. Irrigación, 11500 Mexico City, Mexico*

Correspondence should be addressed to Julio B. Clempner, julio@k-itech.com

Received 1 May 2008; Accepted 7 September 2008

Recommended by Abdennour El Rhalibi

In previous work, attention was restricted to tracking the net using a backward method that knows the target point beforehand (Bellmans's equation), this work tracks the state-space in a forward direction, and a natural form of termination is ensured by an equilibrium point p^* ($M(p^*) = S < \infty$ and $p^* \bullet = \varnothing$). We consider dynamical systems governed by ordinary difference equations described by Petri nets. The trajectory over the net is calculated forward using a discrete Lyapunov-like function, considered as a distance function. Because a Lyapunov-like function is a solution to a difference equation, it is constructed to respect the constraints imposed by the system (a Euclidean metric does not consider these factors). As a result, we prove natural generalizations of the standard outcomes for the deterministic shortest-path problem and shortest-path game theory.

1. Introduction

The shortest-path problem (see [1–3]) plays a fundamental role in Petri nets theory, since it can be used to model processes. The analysis of these models can show useful information about the process. For example, deadlocks, equilibrium points, and so forth can be identified by computational analysis.

While it is possible to analyze such processes using the existing classical theory through the Bellman's equation with the cost criterion ([4–15]), much of this theory has few disadvantages. Bellman's equation is expressed as a sum over the state of a trajectory needs to be solved backwards in time from the equilibrium point (target point). It results in an optimal function when it is governed by Bellman's principle, producing the shortest path needed to reach a known equilibrium point. Notice that the necessity to know the equilibrium point beforehand when applying the equation is a significant constrain, given that, in many practical situations, the state space of a Petri net is too large for an easy identification of the equilibrium point.

Moreover, algorithms using Bellman's equation usually solve the problem in two phases [16]: preprocessing and search. In the preprocessing phase, the distance is usually calculated between each state and the equilibrium points (final states) of the problem, in a backward direction. Then, in the search phase, these results are employed to calculate the distance between each state and the equilibrium points, leading the search process to a forward search.

Tracking the state space in a forward direction allows the decision maker to avoid invalid states that occur in the space generated by a backward search. In most cases, the forward search gives the impression to be more useful than the backward search. The explanation is that in the backward direction, when the case of incomplete final states arises, invalid states appear causing problems.

Shortest-path problem [17, 18] can be classified by two key categories [19]: (a) the single-source shortest-path problem where the goal is to find the shortest path from a given node to a target node (e.g., the algorithms of Dijkstra and Bellman-Ford); and (b) the all-pairs shortest-path problem is a similar problem in which the objective is

to determine the shortest path between every pair of nodes in the net (e.g., the algorithms of Floyd-Warshall and Johnson).

We are concerned about the first case. However, we consider dynamical systems governed by difference equations described by Petri nets. The trajectory over the net is calculated using a discrete Lyapunov-like function. A Lyapunov-like function is considered as a distance function denoting the length from the source place to the equilibrium point. This work is concerned with the analysis of the decision process where a natural form of termination is ensured by an equilibrium point.

Lyapunov-like functions can be used as forward trajectory-tracking functions. Each applied optimal action produces a monotonic progress towards an equilibrium point. Because it is a solution to the difference equation, naturally it will lead the system from the source place to the equilibrium point.

It is important to note that there exist areas of research using Petri nets as modeling tool where the use of a Lyapunov-like function is inherent. For instance, the "Entropy" function is a specific Lyapunov-like function used in Information Theory as a measure of the information disorder. The "free Gibbs energy function" is a Lyapunov-like function used in molecular biology for calculating the energy change in a metabolic network.

This paper introduces a modeling paradigm for shortest-path decision process representation in Petri nets theory. The main point of this paper is its ability to represent the characteristics related only with the global system behavior, and those characteristics related with the trajectory-tracking behavior.

Within the global system behavior properties, we show notions of stability. In this sense, we call equilibrium point to the place in a Petri net that its marking is bounded and it is the last place in the net (sink).

In the trajectory-tracking behavior properties framework, we define the trajectory function as a Lyapunov-like function. By an appropriate selection of the Lyapunov-like function, it is possible to optimize the trajectory. By optimizing the trajectory, we understand that it is the minimum trajectory-tracking value (in a certain sense). In addition, we use the notions of stability in the sense of Lyapunov to characterize the stability properties of the Petri net. The core idea of our approach uses a nonnegative trajectory function that converges in decreasing form to a (set of) final decision states. It is important to point out that the value of the trajectory function associated with the Petri net implicitly determines a set of policies, not just a single policy (in case of having several decisions states that could be reached). We call "optimum point" the best choice selected from a number of possible final decision places that may be reached (to select the optimum point, the decision process chooses the strategy that optimizes the trajectory-tracking value).

As a result, we show that the global system behavior properties and the trajectory-tracking behavior properties of equilibrium, stability, and optimum-point conditions meet under certain restrictions: if the Petri net is finite, then we have that a final decision place is an equilibrium point.

The paper is structured in the following manner. The next section discusses the motivation of the work. Section 3 presents the formulation of the decision model, and all the structural assumptions are introduced there, giving a detailed analysis of the equilibrium, stability, and optimum-point conditions for the global system behavior properties and the trajectory tracking behavior parts of the Petri net. Section 4 presents the properties of the model. Finally, in Section 5 some concluding remarks are outlined.

2. Motivation

In this paper, we consider dynamical systems in which the time variable changes discretely, and the system is governed by ordinary difference equations. Let us consider systems of first-order difference equations given by

$$s_{n+1} = f(s_n, a_n), \quad s_{n_0} = s_0, \quad n \in \mathbb{N}_+^{n_0}, \tag{1}$$

where s_i with $i \in \mathbb{N}$ are the state variable of the system, s_0 is the initial state, a_i and $i \in \mathbb{N}$ are the action of the system, $\mathbb{N}_+^{n_0} = \{n_0, n_0 + 1, \ldots, n_0 + k, \ldots\}$, $n_0 \geq 0$. The system is specified by the state transition function f, which is always assumed as a one-to-one function for any fixed a and $n \in \mathbb{N}$, continuous in all its arguments.

Lyapunov defined a scalar function L, called a Lyapunov-like function, inspired by a classical energy function, which has four important properties that are sufficient for establishing the domain of attraction of a stable equilibrium point: (a) $\exists s^*$ such that $L(s^*) = 0$; (b) $L(s) > 0$ for all $s \neq s^*$; (c) $L(s) \to \infty$ when $s \to \infty$; and (d) $\Delta L = L(s_{i+1}) - L(s_i) < 0$ for all i, $s_i \neq s^*$. The condition (a) requires the equilibrium point to have zero potential by means of a translation to the origin, (b) means that the Lyapunov-like function to be semipositive defined, (c) means that there is no s^* reachable from some s, and (d) means that the Lyapunov-like function has a minimum at the equilibrium point.

The main idea of Lyapunov is attained in the following interpretation: given an isolated physical system, if the change of the energy E for every possible state s is negative, with the exception of the equilibrium point s^*, then the energy will decrease until it finally reaches the minimum at s^*. Intuitively, this concept of stability means that a system perturbed from its equilibrium point will always return to it.

A system is stable [20, 21] if for a given set of initial states the state of the system ensures (i) to reach a given set of states and stay there perpetually or, (ii) to go to a given set of states infinitely often. The conventional notions of stability in the sense of Lyapunov and asymptotic stability can be used to characterize the stability properties of discrete event systems. An important advantage of the Lyapunov approach is that it does not require high-computational complexity but the difficulty lies in specifying the Lyapunov-like function for a given problem.

At this point, it is important to note that the Lyapunov-like function L is not unique, however the energy function of a system is only one of its kind. A system whose energy E decreases on the average, but not necessarily at each instance, is stable but E is not a Lyapunov-like function.

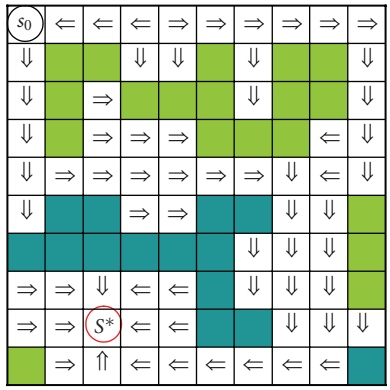


Figure 1: An illustrative example of finding the shortest path in a grid world.

Lyapunov-like functions [22] can be used as trajectory-tracking functions and optimal cost-to-target functions. As a result of calculating a Lyapunov-like function, a discrete vector field can be built for tracking the actions over the net. Each applied optimal action produces a monotonic progress (of the optimal cost-to-target value) toward an equilibrium point. In this sense, if the function decreases with each action taken, then it approaches an infimum/minimum (that converges asymptotically or reaches a constant).

From what we have stated before, we can deduce the following geometric interpretation of distance [22]: (a) $L(s)$ is a measure of the distance from the starting state s_0 to any state s in the state space (this is straightforward from the fact that $\exists s^*$ such that $L(s^*) = 0$ and $L(s) > 0$ for all $s \neq s^*$); and (b) the distance from the stating state s_0 to any state s_n in the state space decreases, when $n \to \infty$. It is because $L(s_{i+1}) - L(s_i) < 0$ for all i, $s_i \neq s^*$.

A Lyapunov-like function can be considered as a distance function denoting the length from the initial state to the equilibrium point. It is important to note that the Lyapunov-like function is constructed to respect the constraints imposed by the difference equation of the system. In contrast, a Euclidean metric does not take into account these factors. For that reason, the Lyapunov-like function offers a better understanding of the concept of the distance required to converge to an equilibrium point in a discrete dynamical system.

By applying the computed actions, a kind of discrete vector field can be imagined over the search graph. Each applied optimal action yields a reduction in the optimal cost-to-target value, until the equilibrium point is reached. Then, the cost-to-target values can be considered as a discrete Lyapunov function.

In our case, an optimal discrete problem, the cost-to-target values are calculated using a discrete Lyapunov-like function. Every time a discrete vector field of possible actions is calculated over the decision process. Each applied optimal action (selected via some "criteria") decreases the optimal value, ensuring that the optimal course of action is followed and establishing a preference relation. In this sense, the criteria change the asymptotic behavior of the Lyapunov-like function by an optimal trajectory-tracking value.

Usually, the criterion in optimization problems is related with the choice of whether to minimize or maximize the optimal action. If the problem is related with energy transformations, as is classically the case in control theory, then the criterion of minimization is applied. However, if the dilemma involves a reward, typical in game theory, then maximization is considered. In this work, we will arbitrary consider the criterion of minimization.

The Lyapunov-like function can be employed as a trajectory-tracking function through the use of an operator, which represents the criterion that selects the optimal action that forces the function to decrease and approaches an infimum/minimum. It forces the function to make a monotonic progress toward the equilibrium point. The Lyapunov-like function can be defined, for example, as

$$L^*(s_{n+1}) = \min_{a^* \in A} L(f(s_n, a_n^*)) \tag{2}$$

which means that the optimal action is chosen to reach the infimum/minimum. The function L^* works as a guide leading the system optimally from its initial state to the equilibrium point.

Example 1. To illustrate the shortest-path problem, let us consider a grid world (see Figure 1). At each time step, an agent is able to select an action among a finite set A of actions, for example, $A = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}\}$. A transition model specifies how the world changes when an action is executed. An "equilibrium point" s^* is a natural final state of the system. Therefore, the shortest-path problem is a search through the state space for an optimal path to the equilibrium point s^*, using a deterministic transition model. The value of a state s is a number $V(s)$ that intuitively speaking expresses the desirability of state s. For instance, let us consider the state-value function V being equal to the min function [23] as a specific Lyapunov-like function able to lead an agent to an equilibrium point in a grid world.

Example 2. The relative entropy or Kullback-Leibler [24, 25] distance between two probability distributions $q^1_{ij|k}$ and $q^2_{ij|k}$ is defined as

$$\mathcal{V}(q^1, q^2) = \sum_{i=1}^{N} \sum_{j=1}^{N} q^1_{ij|k} \log \frac{q^1_{ij|k}}{q^2_{ij|k}}. \tag{3}$$

In the above definition, we use the convention (based on continuity arguments) that $0 \log(0/q^2_{ij|k}) = 0$ and $q^1_{ij|k} \log(q^1_{ij|k}/0) = \infty$. The relative entropy is always nonnegative and is zero if and only if $q^1_{ij|k} = q^2_{ij|k}$. $\mathcal{V}(q^1, q^2)$ is a distance-like function between distributions since it is not symmetric and does not satisfy the triangle inequality.

Example 3. Glycolysis pathway (see Figure 2) is well known and described [11, 26, 27]. It is a ten-step catabolic pathway that makes use of eleven different enzymes. The outcome are the conversion of glucose in two molecules of pyruvate with concurrent net production of 2 ATPs. Glycolysis process can be divided in two stages: (1) the conversion of glucose to glyceraldehyde 3-phosphate with a required input of 2 ATPs,

(2) the conversion of glyceraldehyde 3-phosphate to pyruvate with a net output of 4 ATPs.

Glycolysis can be informally explained from an energetic perspective as follows. The initial amount of glucose may be represented as a ball at the top of an irregular hill. Every time the ball bounces, the hill represents a reaction state in the breakdown of the sugar process. Each bounce of the ball corresponds to a change in free energy level. This energy change is modeled by the Gibbs energy function which is a Lyapunov-like function. It is important to note that bounces are irregular (reaching lower and higher energy levels) and determined by the environment conditions. The final state (pyruvate) is represented by the bottom of the hill where the ball reaches a steady state (not bounces).

Let us explain the Petri net dynamics of the system model as follows. Continuing with the ball and hill explanation, let us suppose that the ball, representing the product pyruvate, is at the bottom of the hill. And let us suppose that there is no net force able to move the ball either up or down the hill. That means that the reactions (forward and backward) are evenly balanced. Therefore, the substances and products are in equilibrium, and no net dynamics will take place. That is, "the metabolic network system is in equilibrium."

3. Formulation

We introduce the concept of decision process Petri nets (DPPNs) by locally randomizing the possible choices, for each individual place of the Petri net [23, 28].

Definition 1. A decision process Petri net is a 7-tuple DDPN $= \{P, Q, F, W, M_0, \pi, U\}$, where

(i) $P = \{p_0, p_1, p_2, \ldots, p_m\}$ is a finite set of places,

(ii) $Q = \{q_1, q_2, \ldots, q_n\}$ is a finite set of transitions,

(iii) $F \subseteq I \cup O$ is a flow relation, where $I \subseteq (P \times Q)$ and $O \subseteq (Q \times P)$ such that $P \cap Q = \varnothing$ and $P \cup Q \neq \varnothing$,

(iv) $W\colon F \to \mathbb{N}_+^1$ is a weight function,

(v) $M_0\colon P \to \mathbb{N}$ is the initial marking,

(vi) $\pi\colon I \to \mathbb{R}_+$ is a routing policy representing the probability of choosing a particular transition, such that for each $p \in P$, $\sum_{q_j:(p,q_j)\in I} \pi((p, q_j)) = 1$,

(vii) $U\colon P \to \mathbb{R}_+$ is a trajectory-tracking function.

We adopt the standard rules about representing nets as directed graphs, namely, places are represented as circles, transitions as rectangles, the flow relation by arcs, and markings are shown by placing tokens within circles [29]. As usual, we will denote $z\bullet = \{y|(z, y) \in F\}$ and $\bullet z = \{y|(y, z) \in F\}$, for all $z \in I \cup O$. A source place is a place $p_0 \in P$ such that $\bullet p_0 = \varnothing$ (there are no incoming arcs into place p_0). A sink place is a place $p_f \in P$ such $p_f\bullet = \varnothing$ (there are no outgoing arcs from p_f). A net system is a pair $\Sigma = (N, M_0)$ comprising a finite net $N = (P, Q, F)$ and an initial marking M_0. A transition $q \in Q$ is enabled at a marking M, denoted by $M[q\rangle$, if for every $p \in \bullet q$, we have that $M(p) \geq 1$. Such a transition can be executed, leading to a marking M' defined by $M' = M - \bullet q + q\bullet$. We denote this by $M[q\rangle M'$ or $M[\rangle M'$. The set of reachable markings of Σ is the smallest (with respect to set inclusion) set $[M_0\rangle$ containing M_0 and such that if $M \in [M_0\rangle$ and $M[\rangle M'$, then $M' \in [M_0\rangle$.

The previous behavior of the *DPPN* is described as follows. When a token reach a place, it is reserved for the firing of a given transition according to the routing policy determined by U. A transition q must fire as soon as all the places $p_1 \in P$ contain enough tokens reserved for transition q. Once the transition fires, it consumes the corresponding tokens and immediately produces an amount of tokens in each subsequent place $p_2 \in P$. When $\pi(\iota) = 0$ for $\iota \in I$ means that there are no outgoing arcs in the place-transitions Petri net (i.e., $p \in \iota$ is a sink).

In Figure 2, we have represented partial routing policies π that generate a transition from state p_1 to state p_2, where $p_1, p_2 \in P$as follows .

Case 1. The probability that q_1 generates a transition from state p_1 to p_2 is 1/3. But, because q_1 transition to state p_2 has two arcs, the probability to generate a transition from state p_1 to p_2 is increased to 2/3.

Case 2. We set by convention for the probability that q_1 generates a transition from state p_1 to p_2 is 1/3 (1/6 plus 1/6). However, because q_1 transition to state p_2 has only one arc, the probability to generate a transition from state p_1 to p_2 is decreased to 1/6.

Case 3. Finally, we have the trivial case when there exists only one arc from p_1 to q_1 and from q_1 to p_2.

It is important to note that, by definition, the trajectory-tracking function U is employed only for establishing a trajectory tracking, working in a different execution level of that of the place-transitions Petri net. The trajectory-tracking function U in no way change the place-transitions Petri net evolution or performance.

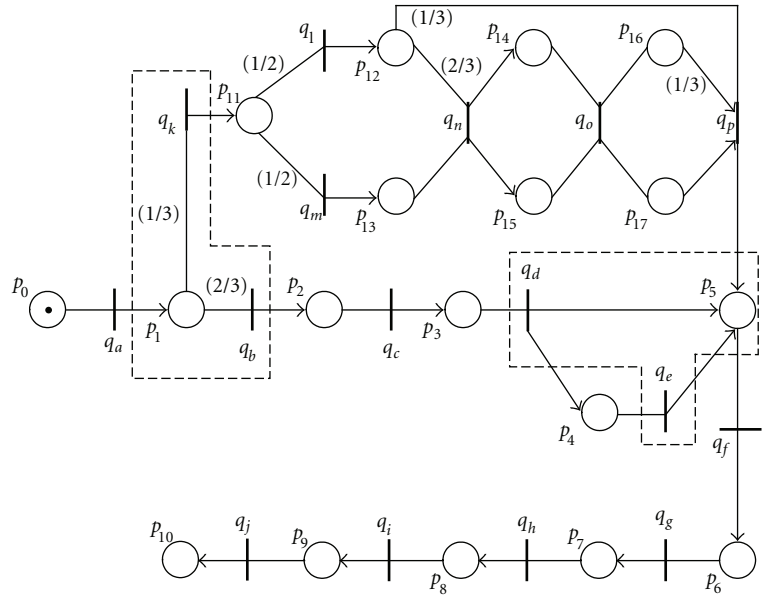
$U_k(\cdot)$ denotes the trajectory-tracking value at place $p_i \in P$ at time k and let $U_k = [U_k(\cdot), \ldots, U_k(\cdot)]^T$ denote the trajectory-tracking state of DDPN at time k. $FN : F \to \mathbb{R}_+$ is the number of arcs from place p to transition q (the number of arcs from transition q to place p).

Consider an arbitrary $p_i \in P$ and for each fixed transition $q_j \in Q$ that forms an output arc $(q_j, p_i) \in O$, we look at all theprevious places p_h of the place p_i denoted by the list (set) $\bullet p_{\eta_{ij}} = \{p_h : h \in \eta_{ij}\}$, where $\eta_{ij} = \{h : (p_h, q_j) \in I, (q_j, p_i) \in O\}$, that materializes all the input arcs $(p_h, q_j) \in I$ and forms the sum

$$\sum_{h\in\eta_{ij}} \Psi(p_h, q_j, p_i)\, U_k(p_h), \tag{4}$$

where $\Psi(p_h, q_j, p_i) = \pi(p_h, q_j) * (FN(q_j, p_i)/FN(p_h, q_j))$ and the index sequence j is the set $\kappa = \{j : q_j \in (p_h, q_j) \cap (q_j, p_i)$ & p_h running over the set $\bullet p_{\eta_{ij}}\}$.

Remark 1. $\bullet p_{\eta_{ij}}$ denote the previous places to p_i for a fixed transition $q_j \in Q$.



- p_0 Gluc
- p_1 G6P
- p_2 F6P
- p_3 FBP
- p_4 DHP
- p_5 GAP
- p_6 BPG
- p_7 3-PG
- p_8 2-PG
- p_9 PEP
- p_{10} Pyr
- p_{11} Ru5P
- p_{12} Xu5P
- p_{13} R5P
- p_{14} 57P
- p_{15} GAP
- p_{16} E4P
- p_{17} F6P

Figure 2: Glycolysis and pentose-phosphate pathways model.

Continuing with all the q_j's, we form the vector indexed by the sequence j identified by $(j_0, j_1, \ldots, j_f)$ as follows:

$$\begin{bmatrix} \sum_{h\in\eta_{ij_0}} \Psi(p_h, q_{j_0}, p_i) U_k(p_h), \\ \sum_{h\in\eta_{ij_1}} \Psi(p_h, q_{j_1}, p_i) U_k(p_h), \ldots, \\ \sum_{h\in\eta_{ij_f}} \Psi(p_h, q_{j_f}, p_i) U_k(p_h) \end{bmatrix}. \tag{5}$$

Intuitively, vector (5) represents all the possible trajectories through the transitions q_js to a place p_i for a fixed i, where j is represented by the sequence $(j_1, j_2, \ldots, j_f)$ and $f = \#(\kappa)$.

Then, formally we define the trajectory-tracking function U as follows.

Definition 2. The trajectory-tracking function U with respect a decision process Petri net DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ is represented by the following equation

$$U_k^{q_j}(p_i) = \begin{cases} U_k(p_0) & \text{if } i = 0,\ k = 0, \\ L(\alpha) & \text{if } i > 0,\ k = 0,\ i \geq 0,\ k > 0, \end{cases} \tag{6}$$

where

$$\alpha = \begin{bmatrix} \sum_{h\in\eta_{ij_0}} \Psi(p_h, q_{j_0}, p_i) U_k^{q_{j_0}}(p_h), \\ \sum_{h\in\eta_{ij_1}} \Psi(p_h, q_{j_1}, p_i) U_k^{q_{j_1}}(p_h), \ldots, \\ \sum_{h\in\eta_{ij_f}} \Psi(p_h, q_{j_f}, p_i) U_k^{q_{j_f}}(p_h) \end{bmatrix}, \tag{7}$$

the function $L : D \subseteq \mathbb{R}_+^n \to \mathbb{R}_+$ is a Lyapunov-like function which optimizes the trajectory-tracking value through all possible transitions (i.e., through all the possible trajectories defined by the different q_js), D is the decision set formed by the j's; $0 \leq j \leq f$, of all those possible transitions $(q_j p_i) \in O$, $\Psi(p_h, q_j, p_i) = \pi(p_h, q_j) * (FN(q_j, p_i)/FN(p_h, q_j))$, η_{ij} is the index sequence of the list of previous places to p_i through transition q_j, $p_h (h \in \eta_{ij})$ is a specific previous place of p_i through transition q_j.

Example 4. OR-Path (see Figure 3). Define the Lyapunov-like function L in terms of the Entropy $H(p_i) = -p_i \ln p_i$ as $L = \min_{i=1,\ldots,|\alpha|}(-\alpha_i \ln \alpha_i)$:

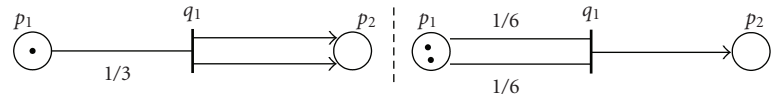


Figure 3: (Left): routing policy case 1. (Right): routing policy case 2.

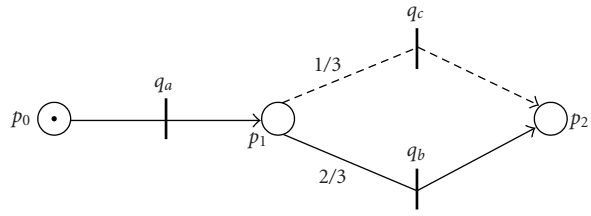


Figure 4: OR-Path Example.

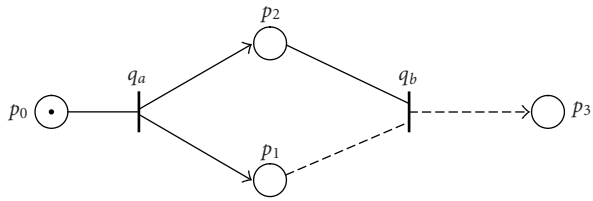


Figure 5: AND-path example.

(i) $U_{k=0}(p_0) = 0.2$,

(ii) $U_{k=0}^{q_a}(p_1) = L[\Psi(p_0, q_a, p_1)U_{k=0}^{q_a}(p_0)] = \min H[1 * 0.2] = H[0.2] = 0.321$,

(iii) $U_{k=0}^{q_b}(p_2) = L[\Psi(p_1, q_b, p_2)U_{k=0}^{q_b}(p_1), \Psi(p_1, q_c, p_2) U_{k=0}^{q_b}(p_1)] = \min H[1/3 * 0.321, 2/3 * 0.321] = \min[0.239, 0.329] = 0.239$.

Example 5. AND-Path (see Figure 4). Define the Lyapunov-like function L in terms of the Entropy $H(p_i) = -p_i \ln p_i$ as $L = \min_{i=1,\ldots,|\alpha|}(-\alpha_i \ln \alpha_i)$:

(i) $U_{k=0}(p_0) = 0.2$,

(ii) $U_{k=0}^{q_a}(p_1) = L[\Psi(p_0, q_a, p_1)U_{k=0}^{q_a}(p_0)] = \min H[1 * 0.2] = H[0.2] = 0.321$,

(iii) $U_{k=0}^{q_b}(p_2) = L[\Psi(p_0, q_b, p_2)U_{k=0}^{q_b}(p_0)] = \min H[1 * 0.2] = H[0.2] = 0.321$,

(iv) $U_{k=0}^{q_c}(p_3) = L[\Psi(p_1, q_c, p_3)U_{k=0}^{q_c}(p_1) + \Psi(p_2, q_c, p_3)U_{k=0}^{q_c}(p_2)] = \min H[1 * 0.321 + 1 * 0.321] = H[0.642] = 0.284$.

From the previous definition, we have the following remark.

Remark 2. (i) Note that the Lyapunov-like function L guarantees that the optimal course of action is followed (taking into account all the possible paths defined). In addition, the function L establishes a preference relation because, by definition, L is asymptotic; this condition gives to the decision maker the opportunity to select a path that optimizes the trajectory-tracking value.

(ii) The iteration over k for U is as follows:

(1) for $i = 0$ and $k = 0$ the trajectory-tracking value is $U_0(p_0)$ at place p_0 and for the rest of the places p_i the trajectory-tracking value is 0;

(2) for $i \geq 0$ and $k > 0$ the trajectory-tracking value is $U_k^{q_j}(p_i)$ at each place p_i, and is computed by taking into account the trajectory-tracking value of the previous places p_h for k and $k-1$ (when needed).

Property 1. The continues function $U(\cdot)$ satisfies the following properties:

(1) $\exists p^\Delta \in P$ such that

(a) if there exists an infinite sequence $\{p_i\}_{i=1}^{\infty} \in P$ with $p_n \underset{n\to\infty}{\longrightarrow} p^\Delta$ such that $0 \leq \cdots < U(p_n) < U(p_{n-1}) \cdots < U(p_1)$, then $U(p^\Delta)$ is the infimum, that is, $U(p^\Delta) = 0$;

(b) if there exists a finite sequence $p_1, \ldots, p_n \in P$ with $p_1, \ldots, p_n \to p^\Delta$ such that $C = U(p_n) < U(p_{n-1}) \cdots < U(p_1)$, then $U(p^\Delta)$ is the min imum, that is, $U(p^\Delta) = C$, where $C \in \mathbb{R}$, $(p^\Delta = p_n)$;

(2) $U(p) > 0$ or $U(p) > C$, where $C \in \mathbb{R}$, for all $p \in P$ such that $p \neq p^\Delta$;

(3) for all $p_i, p_{i-1} \in P$ such that $p_{i-1} \leq_U p_i$ then $\Delta U = U(p_i) - U(p_{i-1}) < 0$.

From the previous property, we have the following remark.

Remark 3. In property 1 point 3, we state that $\Delta U = U(p_i) - U(p_{i-1}) < 0$ for determining the asymptotic condition of the Lyapunov-like function. However, it is easy to show that such property is convenient for deterministic systems. In Markov decision process, systems are necessary to include probabilistic decreasing asymptotic conditions to guarantee the asymptotic condition of the Lyapunov-like function.

Property 2. The trajectory-tracking function $U : P \to \mathbb{R}_+$ is a Lyapunov-like function.

Proof. Proof comes straightforward from the previous definitions. □

Remark 4. From Properties 1 and 2, we have the following :

(i) $U(p^\Delta) = 0$ or $U(p^\Delta) = C$ means that a final state is reached. Without lost generality, we can say that $U(p^\Delta) = 0$ by means of a translation to the origin.

(ii) In Property 1, we determine that the Lyapunov-like function $U(p)$ approaches to a infimum/minimum when p is large thanks to property (d) of the definition the Lyapunov-like function (see motivation).

(iii) Property 1, point 3 is equivalent to the following statement: $\exists\{\varepsilon_i\}$, $\varepsilon_i > 0$ such that $|U(p_i) - U(p_{i-1})| > \varepsilon_i$, for all p_i, $p_{i-1} \in P$ such that $p_{i-1} \leq_U p_i$.

Explanation. Intuitively, a Lyapunov-like function can be considered as trajectory-tracking function and optimal cost function. In our case, an optimal discrete problem, the cost-to-target values are calculated using a discrete Lyapunov-like function. Every time a discrete vector field of possible transitions is calculated over the decision process. Each applied optimal transition (selected via some "criterion," e.g., $\min(\cdot)$) decreases the optimal value, ensuring that the optimal course of action is followed and establishing a preference relation. In this sense, the criterion changes the asymptotic behavior of the Lyapunov-like function by an optimal trajectory-tracking value. It is important to note that the process finished when the equilibrium point is reached. This point determines a significant difference with Bellman's equation.

Example 6 (Conc-Path (see Figure 2)). Biochemical pathway of the free energy profile of the glycolysis and pentose-phosphate. The following was adapted from Biochemistry Lehninger et al. [26] and Campbell and Farrel [30]. The free energy changes were calculated using the steady-state metabolite concentrations in RBC's and the equation $U = RT\ln([\text{Products}]/[\text{Reactants}])$. $U = 0$ was set arbitrarily at the end of the pathway after the pyruvate kinase step. The overall reaction for the pathway is shown in Figure 1. Because $L : D \subseteq \mathbb{R}^n \to \mathbb{R}_+$, we will use the function $\min_{i=1,\ldots,|\alpha|}(\alpha_i \in D)$ to select the proper element of the vector $\alpha \in D$:

(i) $U_{k=0}(\text{Glucose}) = 17.17$ kcal/mol;

(ii) $U^{q_a}_{k=0}(G6P) = L[\Psi(\text{Glucose}, q_a, G6P) * U^{q_a}_{k=0}(\text{Glucose})] = G[\Psi(p_0, q_a, p_1)U^{q_a}_{k=0}(p_0)] = 9.17$ kcal/mol.

A decision is taken and q_b is selected instead of q_k based in the environment condition modeled via the routing policy (1/3, 2/3).

(i) $U^{q_b}_{k=0}(F6P) = L[\Psi(G6P, q_b, F6P) * U^{q_b}_{k=0}(G6P)] = G[2/3 * U^{q_b}_{k=0}(p_1)] = 8.98$ kcal/mol.

(ii) $U^{q_c}_{k=0}(FBP) = L[\Psi(F6P, q_c, FBP) * U^{q_c}_{k=0}(F6P)] = G[\Psi(p_2, q_c, p_3) * U^{q_c}_{k=0}(p_2)] = 3.90$ kcal/mol.

(iii) $U^{q_d}_{k=0}(DHP) = L[\Psi(FBP, q_d, DHP) * U^{q_d}_{k=0}(FBP)] = G[\Psi(p_3, q_d, p_4) * U^{q_d}_{k=0}(p_3)] = 3.71$ kcal/mol.

The Conc-Path is calculated at p_5.

(i) $$U^{q_d}_{k=0}(GAP) = \begin{cases} L[\Psi(FBP, q_d, GAP) * U^{q_d}_{k=0}(FBP)] \\ L[\Psi(DHP, q_e, GAP) * U^{q_e}_{k=0}(DHP)] \end{cases} = \begin{cases} G[\Psi(p_3, q_d, p_5) * U^{q_d}_{k=0}(p_3)] \\ G[\Psi(p_4, q_e, p_5) * U^{q_e}_{k=0}(p_4)] \end{cases} = 4.00 \text{ kcal/mole.}$$

(ii) $U^{q_f}_{k=0}(BPG) = L[\Psi(GAP, q_f, BPG) * U^{q_f}_{k=0}(GAP)] = G[\Psi(p_5, q_f, p_6) * U^{q_f}_{k=0}(p_5)] = 3.71$ kcal/mol.

(iii) $U^{q_g}_{k=0}(3PG) = L[\Psi(BPG, q_g, 3PG) * U^{q_g}_{k=0}(BPG)] = G[\Psi(p_6, q_g, p_7) * U^{q_g}_{k=0}(p_6)] = 4.10$ kcal/mol.

(iv) $U^{q_h}_{k=0}(2PG) = L[\Psi(3PG, q_h, 2PG) * U^{q_h}_{k=0}(3PG)] = G[\Psi(p_7, q_h, p_8) * U^{q_h}_{k=0}(p_7)] = 4.20$ kcal/mol.

(v) $U^{q_i}_{k=0}(PEP) = L[\Psi(2PG, q_i, PEP) * U^{q_i}_{k=0}(2PG)] = G[\Psi(p_8, q_i, p_9) * U^{q_i}_{k=0}(p_8)] = 4.00$ kcal/mol.

(vi) $U^{q_j}_{k=0}(\text{Pyruvate}) = L[\Psi(PEP, q_j, \text{Pyruvate}) * U^{q_j}_{k=0}(PEP)] = G[\Psi(p_9, q_j, p_{10}) * U^{q_j}_{k=0}(p_9)] = 0$ kcal/mol.

(vii) $U^{q_j}_{k=0}(\text{Pyruvate}) = 0$ *was set arbitrarily at the end of the pathway, that is, after the pyruvate kinase step.*

Remark 5. We are using [] to denote the OR-Path, $\sum$ to denote the AND-Path, and $\{|$ to denote the Conc-Path.

4. Properties of the Model

We will identify the global system properties of the DPPN as those properties related with the PN.

Theorem 1. *The decision process Petri net* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *is bounded by a place* p^* *of the system.*

Proof. Let us suppose that the DPPN is not finite. Then p^* is never reached. Therefore, it is possible to evolve in time n and to reduce the trajectory function value over p^*. However, the Lyapunov-like trajectory function converges to zero when $n \to \infty$ (or reached a minimum), that is, $U_n = 0$ or $U_n = C$. □

Theorem 2. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a decision process Petri net bounded by a place* p^*. *Then, a Lyapunov-like trajectory function can be constructed if and only if* p^* *is reachable from* s_0.

Proof. ($\Rightarrow$) If U is a Lyapunov-like function then by the previous theorem p^* is reachable.

($\Leftarrow$) By induction, let us construct the optimal inverse path from p^* to p_0. At each discrete time $n \in \mathbb{N}$ in descending order (n is the maximum place index) the place of a system p_n is observed and a transition $q_k \in Q$ leading to p_{n-1} is chosen. We choose the trajectory function U as the best choice set of states. We continue this process until p_0 is reached. Then, the trajectory function U is a Lyapunov-like function. □

Notation. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, $\mathbb{N}^{n_0}_+ = \{n_0, n_0 + 1, \ldots, n_0 + k, \ldots\}$, $n_0 \geq 0$, $\mathbb{R} = (-\infty, \infty)$ and $\mathbb{R}_+ = [0, \infty)$.

Let us consider systems of first ordinary difference equations given by

$$\begin{aligned} x(n+1) &= \psi[n, x(n)] \\ x(n_0) &= x_0 \end{aligned} \quad \text{for } n \in \mathbb{N}^{n_0}_+, \tag{8}$$

where $x(n) \in \mathbb{R}^d$ and $\psi : \mathbb{N}^{n_0}_+ \times \mathbb{R}^d \to \mathbb{R}^d$ is continuous in $x(n)$.

Definition 3. The n-vector valued function $\phi(n, n_0, x_0)$ is a solution of (8) if $\phi(n_0, n_0, x_0) = x_0$ and $\phi(n+1, n_0, x_0) = \psi(n, \phi(n, n_0, x_0))$ for all $n \in \mathbb{N}_+^{n_0}$.

Definition 4. The system (8) is said to be (see [20, 21]) practically stable if, given (λ, A) with $0 < \lambda < A$, it holds that

$$|x_0| < \lambda \Longrightarrow |x(n, n_0, x_0)| < A, \quad \forall n \in \mathbb{N}_+^{n_0}, n_0 \geq 0. \tag{9}$$

Definition 5. The system (8) is said to be (see [20, 21]) uniformly practically stable, if it is practically stable for every $n_0 \geq 0$.

Definition 6. A continuous function $\alpha : [0, \infty) \to [0, \infty)$ belongs to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$.

Let us consider [21] the vector function $v(n, x(n))$, $v : \mathbb{N}_+^{n_0} \times \mathbb{R}^d \to \mathbb{R}_+^p$ and let us define the variation of v relative to (8) by

$$\Delta v = v(n+1, x(n+1)) - v(n, x(n)). \tag{10}$$

Then, we have the following results [20, 21, 31, 32].

Theorem 3. *Let $v : \mathbb{N}_+^{n_0} \times \mathbb{R}^n \to \mathbb{R}_+$ be a continuous function in x, such that for $\beta, \alpha \in \mathcal{K}$, it holds that $\beta(|x|) \leq v(n, x(n)) \leq \alpha(|x|)$ and $\Delta v(n, x(n)) \leq w(n, v(n, x(n)))$ holds for $n \in \mathbb{N}_+^{n_0}$, $x(n) \in \mathbb{R}^n$, where $w : \mathbb{N}_+^{n_0} \times \mathbb{R}_+ \to \mathbb{R}$ is a continuous function in the second argument. Suppose that $\gamma(n, u) \equiv u + w(n, u)$ is nondecreasing in u, $0 < \lambda < A$ are given and finally that $\alpha(\lambda) < \beta(A)$ is satisfied. Then, the stability properties of*

$$u(n+1) = \gamma(n, u(n)), \qquad u(n_0) = u_0 \geq 0 \tag{11}$$

imply the corresponding stability properties of the system (8).

Proof. The stability properties are preserved for the following.

(1) Practically stable. Let us suppose that $u(n+1)$ is practically stable for $(\alpha(\lambda), \beta(A))$ then, we have that $u_0 < \alpha(\lambda) \Rightarrow |u(n, n_0, u_0)| < \beta(A)$ for $n \geq n_0$, where $u(n, n_0, u_0)$ is the solution of (11). Let $|x_0| < \lambda$, we claim that $|x(n, n_0, x_0)| < A$ for $n \geq n_0$. If not, there would exist $n_1 \geq n_0$ and a solution $x(n, n_0, x_0)$ such that $|x(n_1)| \geq A$ and $|x(n)| < A$ for $n_0 \leq n < n_1$. Choose $u_0 = v(n_0, x_0)$, then $v(n, x(n)) \leq u(n)$ for all $n \geq n_0$. (If not $v(n, x(n)) \leq u(n)$ and $v(n+1, x(n+1)) > u(n+1) \Rightarrow \gamma(n, u(n)) = u(n+1) < v(n+1, x(n+1)) = \Delta v(n, x_0) + v(n, x(n)) \leq w(n, v(n)) + v(n, x(n)) = \gamma(n, v(n)) - v(n, x(n)) + v(n, x(n)) \leq \gamma(n, u(n))$ which is a contradiction) . Hence we get that $\beta(A) \leq \beta(|x(n_1)|) \leq v(n_1, x(n_1)) \leq u(n_1, n_0, u_0) < \beta(A)$ (where the last inequality is because the condition $|x_0| < \lambda \Rightarrow v(n_0, x_0) < \alpha(\lambda)$), which cannot hold therefore, system (8) is practically stable.

(2) Stable. Suppose that system (11) is stable, that is, for all $\varepsilon > 0 \exists \xi = \xi(\varepsilon, n_0) > 0$ such that if $u_0 < \xi \Rightarrow |u(n, n_0, u_0)| < \beta(\varepsilon)$ for $n \geq n_0$. Now, since v is a continuous function in x, there exists a $\delta = \delta(\varepsilon, n_0) > 0$ such that if $|x_0| < \delta \Rightarrow |v(n_0, x_0)| < \xi$ then setting $v(n_0, x_0)$ equal to u_0 by the comparison principle (which was implicitly proved in point 1) implies that $v(n, x(n)) \leq u(n)$ for all $n \geq n_0$. Taking δ equal to the one given from the continuity of v, $|x(n, n_0, x_0)| < \varepsilon$ for $n \geq n_0$. If not, there would exist $n_1 \geq n_0$ such that $|x(n_1)| \geq \varepsilon$ and $|x(n)| < \varepsilon$ for $n_0 \leq n < n_1$ but then

$$\begin{aligned} \beta(\varepsilon) &\leq \beta(x(n_1)) \leq v(n_1, x(n_1)) \\ &\leq u(n_1, n_0, u_0) < \beta(\varepsilon) \end{aligned} \tag{12}$$

which cannot hold therefore, we must have that $|x(n, n_0, x_0)| < \varepsilon$ for $n \geq n_0$ as desired.

(3) Asymptotically stable. We know that system (8) is stable, the fact that it is asymptotically stable follows thanks to

$$\begin{aligned} 0 \leq \lim_{n \to \infty} \beta(|x_n|) &\leq \lim_{n \to \infty} v(n, x(n)) \\ \leq \lim_{n \to \infty} u(n) = 0 &\Longrightarrow \lim_{n \to \infty} |x_n| = 0. \end{aligned} \tag{13}$$

(4) Uniformly stable. Assume that the comparison system is uniformly stable, meaning that $\exists \xi = \xi(\varepsilon) > 0$ (independent of n) such that $u_0 < \xi \Rightarrow |u(n, n_0, u_0)| < \beta(\varepsilon)$ for $n \geq n_0$ and let $\delta > 0$ independent of n such that $|x_0| < \delta \Rightarrow |x(n, n_0, x_0)| < \varepsilon$, for $n \geq n_0$. Since v is a decreasing function there exists $a\, \alpha \in \mathcal{K}$ such that $v(n, x(n)) \leq \alpha(|x_n|)$. Then, choosing $\delta = \alpha^{-1}(\xi)$ works (if $|x_0| < \delta = \alpha^{-1}(\xi) \Rightarrow v(n_0, x_0) = |v(n_0, x_0)| \leq \alpha(|x_0|) < \xi$) and choosing $u_0 = v(n_0, x_0)$ we arrive to the inequality

$$\beta(|x(n)|) \leq v(n, x(n)) \leq u(n, n_0, u_0) < \beta(\varepsilon). \tag{14}$$

But $\delta$is independent of n. Therefore, the system (8) is uniformly stable. □

We will extend the last theorem to the case of several Lyapunov functions. Let us consider a vector Lyapunov function $v(n, x(n))$, $v : \mathbb{N}_+^{n_0} \times \mathbb{R}^d \to \mathbb{R}_+^p$ and let us define the variation of v relative to (8). Then, we have the following theorem.

Theorem 4. *Let $v : \mathbb{N}_+^{n_0} \times \mathbb{R}^d \to \mathbb{R}_+^p$ be a continuous function in x, define the function $v_0(n, x(n)) = \sum_{i=1}^p v_i(n, x(n))$ such that it satisfies the estimates:*

$$\begin{aligned} \beta(|x|) \leq v_0(n, x(n)) \leq \alpha(|x|) \quad &\text{for } \alpha, \beta \in \mathcal{K}, \\ \Delta v(n, x(n)) \leq w(n, v(n, x(n))) & \end{aligned} \tag{15}$$

for $n \in \mathbb{N}_+^{n_0}$, $x(n) \in \mathbb{R}^d$, where $w : \mathbb{N}_+^{n_0} \times \mathbb{R}_+^p \to \mathbb{R}^p$ is a continuous function in the second argument. Assume that $\gamma(n, u) \triangleq u + w(n, u)$ is nondecreasing in u, $0 < \lambda < J$ are given and $\alpha(\lambda) < \beta(A)$ is satisfied. Then, the practical stability properties of

$$u(n+1) = \gamma(n, u(n)), \qquad u(n_0) = u_0 \geq 0 \tag{16}$$

implies the corresponding practical stability properties of system (8).

Proof. (1) Let us suppose that $u(n+1)$ is practically stable for $(\alpha(\lambda), \beta(A))$. Then we have that $\sum_{i=1}^p u_{0_i} < \alpha(\lambda) \Rightarrow$

$\sum_{i=1}^{p} u_i(n, n_0, u_0) < \beta(A)$, for $n \geq n_0$, where $u_i(n, n_0, u_0)$ is the vector solution of (16). Let $|x_0| < \lambda$, we claim that $|x(n, n_0, x_0)| < A$ for $n \geq n_0$. If not, there would exist $n_1 \geq n_0$ and a solution $x(n, n_0, x_0)$ such that $|x(n_1)| \geq A$ and $|x(n)| < A$ for $n_0 \leq n < n_1$. Choose $u_0 = v(n_0, x_0)$, then $v(n, x(n)) \leq u(n, n_0, u_0)$ for all $n \geq n_0$. Therefore we have that $\beta(A) \leq \beta(|x(n_1)|) \leq v_0(n_1, x(n_1)) \leq \sum_{i=1}^{p} u_i(n_1, n_0, u_0) < \beta(A)$ which cannot hold. As a result, system (8) is practically stable.

(2) From the continuity of v with respect to the second argument, it is always possible to make $v_0(n_0, x_0) < \sum_{i=1}^{p} u_{0_i} < \alpha(\lambda) \Rightarrow v_0(n, x(n)) \leq \sum_{i=1}^{p} u_i(n, n_0, u_0) < \beta(A)$. We want to prove that $|x(n, n_0, x_0)| < A$ for $n \geq n_0$. If it is not true, there exists an $n_1 \geq n_0$ and a solution $x(n, n_0, x_0)$ such that $|x(n_1)| \geq A$ and $|x(n)| < A$ for $n_0 \leq n < n_1$. Then, we have that $\beta(A) \leq \beta(|x(n_1)|) \leq v_0(n_1, x(n_1)) \leq \sum_{i=1}^{p} u_i(n_1, n_0, u_0) < \beta(A)$!, which proves our claim. □

Remark 6. If in the point 1 of the proof it is not true that $v(n, x(n)) \leq e(n, n_0, e_0)$ and $v(n+1, x(n+1)) > e(n+1, n_0, e_0)$, then we have that $\gamma(n, e(n)) = e(n+1, n_0, e_0) < v(n+1, x(n+1)) = \Delta v(n, x_0) + v(n, x(n)) \leq w(n, v(n)) + v(n, x(n)) = \gamma(n, v(n)) - v(n, x(n)) + v(n, x(n)) = \gamma(n, v(n)) \leq \gamma(n, e(n))$ which is a contradiction.

Then, we have the following result [21].

Corollary 1. *From Theorem 5, the following hold.*

(1) *If $w(n, e) \equiv 0$, the uniform practical stability of* (8) *which implies structural stability [21, 33] is obtained .*

(2) *If $w(n, e) = -c(e)$, for $c \in \mathcal{K}$, the uniform practical asymptotic stability of* (8) *[21] is obtained .*

Example 7. The diamond is the stable form of carbon at extremely high pressures while the graphite is the stable form at normal atmospheric pressures. Regardless of that, diamonds appear stable at normal temperatures and pressures, but, in fact, are very slowly converting to graphite. Heat increases the rate of this transformation, but at normal temperatures the diamond is uniformly practically stable.

For Petri nets, we have the following results of stability [31].

Proposition 1. *Let PN be a Petri net. Therefore, PN is uniform practical stable if there exists a Φ strictly positive m vector such that*

$$\Delta v = u^T A \Phi \leq 0. \tag{17}$$

Moreover, N is uniform practical asymptotic stability if the following equation holds:

$$\Delta v = u^T A \Phi \leq -c(e), \quad \text{for } c \in \mathcal{K}. \tag{18}$$

Proof. Let us choose as our candidate Lyapunov function $v(M) = M^T \Phi$ with Φ and m vector to be chosen. It is simple to verify that v satisfies all the conditions of Theorem 3. Therefore, the uniform practical asymptotic stability is obtained if there exists a strictly positive vector Φ such that equation (17) holds. □

Proposition 2. *Let PN be a Petri net. Therefore, PN is uniformly practically stable if there exists a Φ strictly positive m vector such that*

$$\Delta v = u^T A \Phi \leq 0 \Longleftrightarrow A\Phi \leq 0. \tag{19}$$

Proof. $\Rightarrow$) Since $u^T A \Phi \leq 0$ holds, therefore for every u we have that $A\Phi \leq 0$.

$\Leftarrow$) This came from the fact that u is positive. □

Remark 7. The if-and-only-if relationship of (19) exists from the fact that u is positive.

Example 8. The biochemical pathway of the glycolysis (Figure 1). The incidence matrix is as follows:

$$\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}. \tag{20}$$

Choosing $\Phi = [1, 1, 1, 1, 1/2, 1/2, 1/2, 1/2, 1/2, 1/2, 1/2]$, $\Phi > 0$, we obtain that $A\Phi = [0, 0, 0, -1/2, 0, 0, 0, 0, 0, 0]$ concluding stability.

Definition 7. An equilibrium point with respect to a decision process Petri net DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ is a place $p^* \in P$ such that $M_l(p^*) = S < \infty$, for all $l \geq k$, and p^* is a sink.

Theorem 5. *The decision process Petri net* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *is uniformly practically stable iff there exists a Φ strictly positive m vector such that $\Delta v = u^T A \Phi \leq 0$.*

Proof. $\Rightarrow$) It follows directly from Propositions 1 and 2.

$\Leftarrow$) Let us suppose by contradiction that $u^T A \Phi > 0$ with Φ fixed. From $M' = M + u^T A$ we have that $M'\Phi = M\Phi + u^T A \Phi > M\Phi$. Then, it is possible to construct an increasing sequence $M\Phi < M'\Phi < \cdots < M^n\Phi < \cdots$ which grows up without bound. Therefore, the DDPN is not uniformly practically stable. □

Remark 8. It is important to underline that the only places where the DPPN will be allowed to get blocked are those which correspond to equilibrium points.

We will identify the trajectory-tracking properties of the DPPN as those properties related with the trajectory-tracking value at each place of the PN. In this sense, we will relate an optimum point the best possible performance

choice. Formally we will introduce the following definition [23].

Definition 8. A final decision point $p_f \in P$ with respect to a decision process Petri net DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ is a place $p \in P$ where the infimum is asymptotically approached (or the minimum is attained), that is, $U(p) = 0$ or $U(p) = C$.

Definition 9. An optimum point $p^\Delta \in P$ with respect to a decision process Petri net DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ is a final decision point $p_f \in P$ where the best choice is selected "according to some criteria."

Property 3. Every decision process Petri net DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ has a final decision point.

Remark 9. In case that $\exists p_1, \ldots, p_n \in P$, such that $U(p_1) = \cdots = U(p_n) = 0$, then $p_1, \ldots, p_n$ are optimum points.

Remark 10. The monotonicity of U guarantees that it is possible to make the search starting from the decision points.

Then, we can conclude the following theorem.

Theorem 6. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a finite decision process Petri net and let* $(p_0, p_1, \ldots, p_n)$ *be a realized trajectory which converges to* p^Δ *such that* $\exists \epsilon_i : |U_{i+1} - U_i| > \epsilon_i$ *(with* $\epsilon_i > 0$*). Let* $\epsilon = \min\{\epsilon_i\}$*, then the optimum decision point* p^Δ *is reached in a time step bounded by* $O(U_0/\epsilon)$.

Proof. Let us suppose that p^Δ is never reached, then, p^Δ is not a sink (the last place) in the decision process Petri net. So, it is possible to find some output transition to p^Δ. Therefore, it is possible to reduce the trajectory function value over p^Δ by at least ϵ. As a result, it is possible to obtain a lower value than C (that is a contradiction). □

Theorem 7. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a decision process Petri net. Then, U converges to an optimum (final) decision point* $p^\Delta(p_f)$.

Proof. We have to show that U converges to an optimum (final) decision point $p^\Delta(p_f)$. By the previous theorem, the optimum decision point p^Δ is reached in a time step bounded by $O(U_0/\epsilon)$, therefore U converges to p^Δ. □

Proposition 3. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a decision process Petri net and let* $p^\Delta \in P$ *be an optimum point. Then* $U(p^\Delta) \leq U(p)$*, for all* $p \in P$ *such that* $p \leq_U p^\Delta$.

Proof. We have that $U(p^\Delta)$ is equal to the minimum or the infimum. Therefore, $U(p^\Delta) \leq U(p)$ for all $p \in P$ such that $p \leq_U p^\Delta$. □

Theorem 8. *The decision process Petri net* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *is uniformly practically stable iff* $U(p_{i+1}) - U(p_i) \leq 0$.

Proof. ($\Rightarrow$) Let us choose $v = Id(U(p_i))$, then $\Delta v = U(p_{i+1}) - U(p_i) \leq 0$. Then by the autonomous version of Theorem 4 and Corollary 1 the DPPN is stable.

($\Leftarrow$) We want to show that the DPPN is practically stable, that is, given $0 < \lambda < A$, we must show that $|U(p_i)| < A$. We know that $U(p_0) < \lambda$ and since U is nondecreasing, we have that $|U(p_i)| < |U(p_0)| < \lambda < A$. □

Theorem 9. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a decision process Petri net. If* $p^* \in P$ *is an equilibrium point, then it is a final decision point.*

Proof. Let us suppose that p^* is an equilibrium point, we want to show that its trajectory-tracking value has asymptotically approached an infimum (or reached a minimum). Since p^* is an equilibrium point, by definition, it is bounded and it is a sink, for example, its marking can not be modified. But, this implies that the routing policy attached to the transition(s) that follows p^* is 0 (in case there is such a transition(s), i.e., worst case). Therefore, its trajectory-tracking value can not be modified and since the value is a decreasing function of p_i, an infimum or a minimum is attained. Then, p^* is a final decision point. □

Theorem 10. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a finite and nonblocking decision process Petri net (unless* $p \in P$ *is an equilibrium point). If* $p_f \in P$ *is a final decision point, then it is an equilibrium point.*

Proof. If p_f is a final decision point, since the DDPN is finite, there exists a k such that $U_k(p_f) = C$. Let us suppose that p_f is not an equilibrium point.

Case 1. Then, it is not bounded. So, it is possible to increment the marks of p_f in the net. Therefore, it is possible to modify its trajectory-tracking value. As a result, it is possible to obtain a lower value than C.

Case 2. Then, it is not bounded and it is not a sink. So, it is possible to fire some output transition to p_f in such a way that its marking is modified. Therefore, it is possible to modify the trajectory-tracking value over p_f. As a result, it is possible to obtain a lower trajectory-tracking value than C. □

Corollary 2. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a finite and nonblocking decision process Petri net (unless* $p \in P$ *is an equilibrium point). Then, an optimum point* $p^\Delta \in P$ *is an equilibrium point.*

Proof. From the previous theorem, we know that a final decision point is an equilibrium point and since in particular p^Δ is final decision point, then it is an equilibrium point. □

5. Completeness

Theorem 11. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a decision process Petri net and let* $(p_0, p_1, \ldots, p_n)$ *be a realized trajectory which converges to* p^* *such that* $\exists \epsilon_i : |U_{i+1} - U_i| >$

ϵ_i (*with* $\epsilon_i > 0$). *Let* $\epsilon = \min\{\epsilon_i\}$, *then an optimum point* p^* *is reached in a time step bounded by* $O(U_0/\epsilon)$.

Proof. Let us suppose that p^* is never reached, then p^* is not the last place in the decision process Petri net. So, it is possible to find some output transition to p^*. Therefore, it is possible to reduce the trajectory function value over p^* by at least ϵ. As a result, it is possible to obtain a lower value than C (that is a contradiction). □

Remark 11. The complexity time $O(U_0/\epsilon)$ differs with that of the Dijkstra's algorithm.

Remark 12. Each path in DDPN corresponds to a trajectory of/in a given system. The trajectory-tracking function value of U at the source place (U_0^{mon}) divided by $\epsilon = \min\{\epsilon_i\}$ equals the length of the shortest-path. Then, the infimum is equivalent to the infimum length over all paths in DDPN.

Theorem 12. *Let* DDPN $= \{P, Q, F, W, M_0, \pi, U\}$ *be a decision process Petri net. Then,* U *converges to a point* p^*.

Proof. We have to show that U converges to a point p^*. By the previous theorem, the optimum point p^* is reached in a time step bounded by $O(U_0/\epsilon)$, therefore U converges to p^*. □

Proposition 4. *The finite and nonblocking (unless* $p \in P$ *is an equilibrium point) condition over the* DDPN *can not be relaxed.*

Proof. (1) Let us suppose that the DDPN is not finite, that is, p is in a cycle, then the Lyapunov-like function converges when $k \to \infty$, to zero, that is, $L(p) = 0$ but the DPPN has no final place therefore, it is not an equilibrium point.

(2) Let us suppose that the DDPN blocks at some place (not an equilibrium point) $p \in P$. Then, the Lyapunov-like function has a minimum at place p, lets say $L(p) = C$ but p is not an equilibrium point, because it is not necessary to have a sink in the net. □

6. Conclusions

In this work, a formal framework for shortest-path decision process problem representation has been presented. Whereas in previous work, attention was restricted to tracking the net using a utility function Bellman's equation, this work uses a Lyapunov-like function. In this sense, we are changing the traditional cost function by a trajectory-tracking function which is also an optimal cost-to-target function for tracking the net. This makes a significant difference in the conceptualization of the problem domain. The Lyapunov method introduces a new equilibrium and stability concept in decision process.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Mathematics of Operations Research*, vol. 16, no. 3, pp. 580–595, 1991.

[2] D. Blackwell, "Positive dynamic programming," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 415–418, University of California Press, Berkeley, Calif, USA, June-July 1965.

[3] C. Derman, *Finite State Markovian Decision Processes*, Academic Press, New York, NY, USA, 1970.

[4] F. Baccelli, G. Cohen, and B. Gaujal, "Recursive equations and basic properties of timed Petri nets," *Journal of Discrete Event Dynamic Systems*, vol. 1, no. 4, pp. 415–439, 1992.

[5] F. Baccelli, S. Foss, and B. Gaujal, "Structural, temporal and stochastic properties of unbounded free-choice Petri nets," Rapport de Recherche, INRIA, Sophia Antipolis, France, 1994.

[6] R. I. Bahar, E. A. Frohm, C. M. Gaona, et al., "Algebraic decision diagrams and their applications," *Formal Methods in System Design*, vol. 10, no. 2-3, pp. 171–206, 1997.

[7] G. Ciardo and R. Siminiceanu, "Using edge-valued decision diagrams for symbolic generation of shortest paths," in *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD '02)*, M. D. Aagaard and J. W. O'Leary, Eds., vol. 2517 of *Lecture Notes in Computer Science*, pp. 256–273, Portland, Ore, USA, November 2002.

[8] G. Cohen, S. Gaubert, and J. Quadrat, "Algebraic system analysis of timed Petri nets," in *Idempotency*, J. Gunawardena, Ed., Collection of the Isaac Newton Institute, Cambridge University Press, Cambridge, UK, 1998.

[9] J. H. Eaton and L. A. Zadeh, "Optimal pursuit strategies in discrete-state probabilistic systems," *Journal of Basic Engineering*, vol. 84, pp. 23–29, 1962.

[10] B. Gaujal and A. Giua, "Optimal routing of continuous timed Petri nets," *Automatica*, vol. 40, no. 9, pp. 1505–1516, 2004.

[11] K. Hinderer and K.-H. Waldmann, "The critical discount factor for finite Markovian decision processes with an absorbing set," *Mathematical Methods of Operations Research*, vol. 57, no. 1, pp. 1–19, 2003.

[12] K. Hinderer and K.-H. Waldmann, "Algorithms for countable state Markov decision models with an absorbing set," *SIAM Journal on Control and Optimization*, vol. 43, no. 6, pp. 2109–2131, 2005.

[13] V. Khomenko and M. Koutny, "Verification of bounded Petri nets using integer programming," *Formal Methods in System Design*, vol. 30, no. 2, pp. 143–176, 2007.

[14] A. F. Veinott, "Discrete dynamic programming with sensitive discount optimality criteria," *Annals of Mathematical Statistics*, vol. 40, no. 5, pp. 1635–1660, 1969.

[15] H.-C. Yen, "A valuation-based analysis of conflict-free Petri nets," *Systems and Control Letters*, vol. 45, no. 5, pp. 387–395, 2002.

[16] A. S. Poznyak, K. Najim, and E. Gomez-Ramirez, *Self-Learning Control of Finite Markov Chains*, Marcel Dekker, New York, NY, USA, 2000.

[17] A. Auslender and M. Teboulle, "Interior gradient and proximal methods for convex and conic optimization," *SIAM Journal on Optimization*, vol. 16, no. 3, pp. 697–725, 2006.

[18] S. Pan and J.-S. Chen, "Entropy-like proximal algorithms based on a second-order homogeneous distance function for quasi-convex programming," *Journal of Global Optimization*, vol. 39, no. 4, pp. 555–575, 2007.

[19] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press and McGraw-Hill, Cambridge, Mass, USA, 2nd edition, 2001.

[20] V. Lakshmikantham, S. Leela, and A. A. Martynyuk, *Practical Stability of Nonlinear Systems*, World Scientific, Singapore, 1990.

[21] V. Lakshmikantham, V. M. Matrosov, and S. Sivasundaram, *Vector Lyapunov Functions and Stability Analysis of Nonlinear Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[22] R. E. Kalman and J. E. Bertram, "Control system analysis and design via the "second method" of Lyapunov," *Journal of Basic Engineering*, vol. 82(D), pp. 371–393, 1960.

[23] J. B. Clempner, "Colored decision process Petri nets: modeling, analysis and stability," *International Journal of Applied Mathematics and Computer Science*, vol. 15, no. 3, pp. 405–420, 2005.

[24] I. Csiszár, "Information-type measures of difference of probability distribution and indirect observations," *Studia Scientiarum Mathematicarum Hungarica*, vol. 2, pp. 299–318, 1967.

[25] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[26] A. L. Lehninger, D. L. Nelson, and M. M. Cox, *Principles of Biochemistry*, Worth, New York, NY, USA, 4th edition, 2004.

[27] H. Matsuno, S. Fujita, A. Doi, M. Nagasaki, and S. Miyano, "Towards biopathway modeling and simulation," in *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN '03)*, vol. 2679 of *Lecture Notes in Computer Science*, pp. 3–22, Eindhoven, The Netherlands, June 2003.

[28] J. B. Clempner, "Towards modeling the shortest-path problem and games with Petri nets," in *Proceedings of the Doctoral Consortium at the ICATPN*, pp. 1–12, Turku, Finland, June 2006.

[29] J. Desel and J. Esparza, "Free choice Petri nets," in *Cambridge Tracts in Theoretical Computer Science*, vol. 40, Cambridge University Press, Cambridge, UK, 1995.

[30] M. K. Campbell and S. O. Farrell, *Biochemistry*, Brooks Cole, Florence, Ky, USA, 4th edition, 2002.

[31] K. M. Passino, K. L. Burgess, and A. N. Michel, "Lagrange stability and boundedness of discrete event systems," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 5, no. 4, pp. 383–403, 1995.

[32] Z. Retchkiman, "Place-Transitions Petri Nets: Class Notes," CIC-I.P.N., 1998.

[33] T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 670459, 7 pages
doi:10.1155/2009/670459

Research Article

Fractal Analysis of Stealthy Pathfinding Aesthetics

Ron Coleman

Computer Science Department, Marist College, Poughkeepsie, NY 12601, USA

Correspondence should be addressed to Ron Coleman, ron.coleman@marist.edu

Received 31 May 2008; Accepted 25 September 2008

Recommended by Kok Wai Wong

This paper uses a fractal model to analyze aesthetic values of a new class of obstacle-prone or "stealthy" pathfinding which seeks to avoid detection, exposure, openness, and so forth in videogames. This study is important since in general the artificial intelligence literature has given relatively little attention to aesthetic outcomes in pathfinding. The data we report, according to the fractal model, suggests that stealthy paths are statistically significantly unique in relative aesthetic value when compared to control paths. We show furthermore that paths generated with different stealth regimes are also statistically significantly unique. These conclusions are supported by statistical analysis of model results on experimental trials involving pathfinding in randomly generated, multiroom virtual worlds.

1. Introduction

Nonplayer character (NPC) agents in videogames depend on pathfinding to navigate virtual worlds autonomously. The literature on artificially intelligent pathfinding has generally focused on machine efficiency and shortest paths. While these concerns cannot be neglected, they may be of secondary or even doubtful benefit if, in videogames, they lead to movement lacking in sensori-emotional or aesthetic qualities that would otherwise appeal to player expectations of plausibility, intelligence, beauty, and so forth. Indeed, pathfinding without aesthetic considerations tends to look unrealistic and mechanical, detracting from a game's immersive potential and frustrating players [7].

Aesthetics, however, pose challenges. According to a modernist, Kantian view [10], aesthetics in general and notions of beauty and matters of taste in particular are thought to be subjective, relative, and presumably beyond the pale of automation. Yet, game researchers and developers have side-stepped these dilemmas, asking not what *is* beauty in pathfinding but rather what is *knowable* about such beauty which can be captured by heuristics called "aesthetic optimizations" [17] and "aesthetic corrections" [7].

These efforts have yielded encouraging results and drawn attention to basic issues of incorporating aesthetics in pathfinding. Unfortunately, they have depended almost entirely on anecdotal arguments rather than metrics that facilitate hypothesizing about and testing aesthetic outcomes under more quantifiable, independently verifiable regimes. These investigators have furthermore addressed only beautifying heuristics that navigate by straight lines, smooth turns, and avoiding obstacles without tracking them. Such movement, although appealing in some contexts, is not appropriate for all forms of play and types of games.

In this paper, we use fractal analysis to examine a new pathfinding aesthetic which we call "stealthy." These paths, obstacle-prone by nature, are reminiscent of and suitable for covert movement in first-person shooter, role playing, and other types of games wherein the goal is to avoid detection, exposure, all-out encounters—concepts we define mathematically later. We use fractal analysis since, among other reasons we discuss later, this approach has been shown to reliably predict and comport with player expectations of aesthetic appeal in pathfinding [4]. What is interesting is that stealthy pathfinding has a statistically significantly unique fractal signature compared to controls which have not been treated with stealth regimes.

We develop a simple cost heuristic to generate *stealth effects*, that is, stealthy movement patterns. In a series of $N = 100$ experimental trials involving randomly generated, multiroom virtual worlds, we show that the fractal model reliably discriminates between stealthy paths versus two types

of control paths with $p \approx 10^{-9}$ and $p \approx 10^{-13}$, depending, respectively, on the stealth effect. We show furthermore that paths with different stealth effects are unique compared to one another with $p \approx 10^{-3}$. These results confirm previous studies of fractals as a reliable metric for measuring pathfinding aesthetic outcomes.

2. Background and Related Work

The fractal dimension, originally developed by Mandelbrot in his seminal paper [11] as we describe below, has been used by others to assess aesthetic values in artistic masterpieces like Jackson Pollack's "action paintings" [8, 18] and Bach's *Brandenburg Concertos* [20]. Investigators working in these areas were not specifically interested in pathfinding or even for that matter, artificial intelligence.

The artificial intelligence literature, however, is generally silent on pathfinding aesthetics. For example, see texts like those of Bourg and Seemann [2], Millington [13], and Russell and Norvig [19] that cover various forms of automated movement but do not discuss aesthetics.

Rabin [17], Higgins [7], and Stout [22] have noted the need for aesthetic considerations in pathfinding and proposed arguments and heuristics to improve aesthetic outcomes in ways likely to appeal to player expectations of "realism," "beauty," and so forth. For Botea et al. [1], the main interest is machine performance. However, they acknowledge, if only in passing, that navigation in games is incomplete without aesthetic concerns. These efforts, in any case, have all focused on *how* to achieve aesthetic outcomes but not grading, scoring, or in any way, measuring them.

For precisely this reason, Coleman [3] put forth the beauty intensity, $\mathfrak{R}$, as a relative, nonlinear measure of aesthetic appeal in pathfinding. Thus, a path object, P_1, is said to have more "working beauty" than a control or reference path object, P_0, provided that $\mathfrak{R}(P_1|P_0) > 0$. While $\mathfrak{R}$ was shown to give commonsense results in accordance with straight lines, smooth turns, and avoiding obstacles without tracking them, values of $\mathfrak{R}$ are not readily intuitive except in a strictly lattice sense. $\mathfrak{R}$ is furthermore mathematically undefined for some path objects. The implication is that $\mathfrak{R}$ is parametric; it uses explicit, internal assumptions about pathfinding and aesthetics.

Coleman [4] subsequently proposed a fractal model, G, which is similar to and mildly correlated with $\mathfrak{R}$ as a relative, nonlinear measure, that is, $G(P_1|P_0) > 0$ implies that P_1 has more "fractal beauty" than a reference path object, P_0. However, G is a more reliable and intuitive estimator according, respectively, to its variance-to-mean ratio and relationship to textured sensory data. Most importantly for the present study, G is nonparametric. It makes no assumptions about pathfinding or even aesthetics. Thus, G tends to provide more reliable, conservative results.

In this paper, we use G to study a new pathfinding regime, the stealth effect, in relation to controls. We examine paths treated with stealth regimes versus "standard" paths, that is, with no beautifying treatments and "aesthetic" paths, that is, with beautifying treatments. While Coleman [4] was completely analytical, the present effort is both analytical and generative.

3. Fractal Dimension

Mandelbrot developed the fractional (or fractal) dimension as a way to analyze irregularly shaped geometric objects which are no-where differentiable (i.e., textured) and self-similar [11, 12, 14]. Mandelbrot observed furthermore that the fractal dimension, D, of a surface, S, is greater than its topological dimension, n [11, 12], that is, $n < D < n + 1$. Mandelbrot suggested that fractals offered a better description of objects found in nature (e.g., coastlines).

The fractal dimension has different interpretations that come under two general mathematical categories: stochastic and geometric [21]. The stochastic interpretation assumes Brownian fluctuations [20] and might be employed, for instance, in time series analysis. In this paper, we use a geometric interpretation based on the Hausdorff dimension [20]:

$$D(S) = \lim_{\varepsilon \to 0} \frac{\log N_\varepsilon(S)}{\log \varepsilon}, \tag{1}$$

S is a surface, ε is a yardstick or ruler, and $N_\varepsilon(S)$ is the number of self-similar objects or subcomponents covered by the ruler. For fractal objects, $\log N_\varepsilon(S)$ will be greater than $\log(1/\varepsilon)$ by a fractional amount.

One way of interpreting the Hausdorff dimension is through the box counting dimension, that is, reticular cell counting. In this case, if the ruler is a uniform grid of square cells, then a smooth surface passes through twice as many cells if the cell length is reduced by a factor of two. A fractal object passes through more than twice as many cells if the cell length is reduced by a factor of two.

For instance, the coastline of Maine, USA , is not straight or smooth but highly textured with inlets, outcrops, and keys. Researchers using the box counting dimension have estimated its fractal dimension to be between 1.11 and 1.37 depending on where and how measurements are taken [23].

Reticular cell counting is intuitive and straightforward computationally. We use it to estimate the fractal dimension by computing the regression slope of $\log(1/\varepsilon)$ versus $\log N_\varepsilon(S)$. We use a slightly modified version of FracTop [9], which reliably computes the fractal dimension using reticular cell counting, where $\varepsilon = \{2, 3, 4, 6, 8, 12, 16, 32, 64, 128\}$ in pixels are the default rulers. The input to FracTop is a 2D image in Portable Network Graphics (PNG) [16] format which we explain later how to generate given a virtual world.

4. Fractal Model: G

The fractal model we describe is from Coleman [4]. We review it here for the sake of completeness.

Let the surface, S, consist of W, $\{B^j\}$, and P. W is a finite state-space in Euclidean R^n. We assume $n = 2$ or $n = 3$. For analysis purposes, however, the perspective is two dimensional. For example, if the game is a first person shooter, the veiw is from above, looking down on walls,

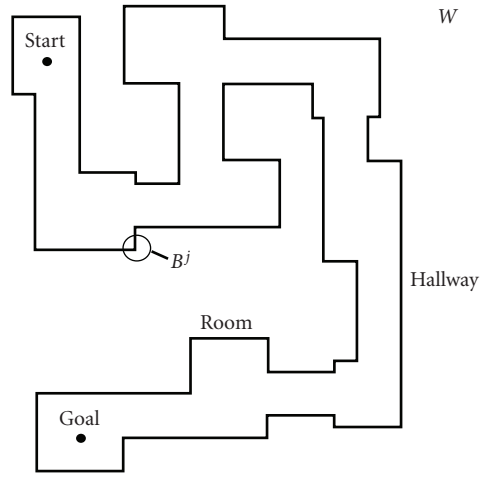


Figure 1: Example of virtual world, W, in 2D perspective with start and goal configurations.

rooms, and hallways. Yet, the NPC perceives the world as a set of rigid-body obstacles, $\{B^j\}$, in two or three dimensions. See Figure 1 as an example.

Let $W : x, y \to t$ for $0 \le x < w,\ 0 \le y < h$, where w and h are width and length features, respectively, of W, and t is a state, namely, $t \in \{0, 1, 2\}$. W contains the set $\{B^j\}$, namely, $B^j = W : B^j \cdot x,\ B^j \cdot y \to 2$.

W also contains A, a "free flying" rigid-body (i.e., the NPC), which has configurations or steps such that $A^i = W : A^i \cdot x,\ A^i \cdot y$. These steps define a path object, P, for $i = 0 \cdots L$ from A^{start} (start $= 0$) to A^{goal} (goal $= L$), where $A^{i+1} = W : A^i \cdot x + \Delta x,\ A^i \cdot y + \Delta y \to 1$ and $\Delta x, \Delta y \in \{-1, 0, 1\}$. All other states of W are "open" or unoccupied, namely, $W : x, y \to 0$. For the worlds we generate, A "tracks" an obstacle if $W : A^i \cdot x \pm k,\ A^i \cdot y \pm k \to 2$, where $k = 1$.

Let $D(P)$ be shorthand notation for the fractal dimension of P for a particular world, W, which includes the open states of W, P, and $\{B^j\}$. Let G be the "fractal beauty" of a path, P_1, in relation to a reference path, P_0, as

$$G(P_1|P_0) = D(P_1) - D(P_0). \tag{2}$$

G is constrained in that W, A^{start}, and A^{goal} are assumed to be the same for both P_1 and P_0. Thus, we say P_1 has more "fractal beauty" than P_0 only if

$$G(P_1|P_0) > 0. \tag{3}$$

P_1 is said to have less fractal beauty than P_0 if $G < 0$. If $G = 0$, then P_1 and P_0 are said to have the same fractal beauty.

5. Stealthy Pathfinding

G does not specify how to find a path. That is the role of pathfinding. In principle, therefore, any suitable pathfinding algorithm suffices. We start with the A^* algorithm [2] as a base. Aside from being generally regarded as the "work horse" of pathfinding for games, A^* is simple, flexible, and straightforward with well-known space and time characteristics [19]. The "standard" A^*, for instance, the one given by Bourg and Seemann [2], does not have an aesthetic objective.

Others have sought to reduce or correct these aesthetic deficiencies through beautifying heuristics [1, 7, 17, 22], that is, if the path score subject to minimization is

$$f(A^{\text{start}}, A^{\text{goal}}) = g(A^{\text{start}}) + h(A^{\text{goal}}), \tag{4}$$

where g is the known cost from the start configuration and h is the heuristic estimate to the goal configuration. (For h, we use Manhattan metric, namely, $h(A^{\text{goal}}) = |\Delta X| + |\Delta Y|$(see [9]) for further information.) By adding a penalty or surcharge to h for turns or zigzags, A^* tends to generate paths with straight lines and smooth turns. Coleman [3] goes further and also penalizes wall tracking within some radius, k, that is, an NPC navigating a game world by following a wall or obstacle may appear to be using the object and not A.I. Thus, it is best to avoid such objects.

Yet in a competitive game world setting, the NPC would not necessarily traverse the middle of a hallway in a straight line or make "pleasant," predictably smooth turns. Indeed, wall tracking is precisely what an NPC might conceivably do if it is seeking to avoid detection, dodge an opponent, or evade a trap.

Whereas the standard A^* is wall-neutral and "aesthetic" A^* is wall-adverse, we define a "stealthy" A^* as one which is obstacle-prone, that is, rather than ignoring obstacles or penalize the NPC for tracking them, the stealthy A^* rewards such paths according to the following schedule if $W : A^i \cdot x \pm k,\ A^i \cdot y \pm k \to 2$, where $k = 1$:

$$H(A^{\text{goal}}, \gamma) = (1 - \gamma) \cdot h(A^{\text{goal}}), \tag{5}$$

where γ is called *stealth effect* and $(1-\gamma)$ is the *discount*. (Note the discount may in fact behave like a surcharge for some values of γ.) Equation (5) supersedes the heuristic component of the A^* algorithm. The nonheuristic component does not change.

We state the following lemmas.

Lemma 1. *$Cor(h, \gamma) = 0$, that is, there is no correlation between the stealth effect, γ, and the heuristic cost, h.*

Proof. By inspection of (5), there is no dependency between the discount and h. □

Lemma 2. *Three possible values of γ give distinct characteristics per the relations below:*

$\gamma = 0$ *standard or obstacle-neutral search,*

$\gamma < 0$ *aesthetic or obstacle-adverse search,*

$\gamma > 0$ *stealthy or obstacle-prone search.*

Proof. If $\gamma = 0$, (5) degenerates to the standard search. If $\gamma < 0$, the discount becomes a surcharge for tracking an obstacle. If $\gamma > 0$, the heuristic cost is discounted. □

Lemma 3. *At the limit, there is no stealth effect and H converges to h, that is,*

$$\lim_{\gamma \to 0} H(\gamma) = H(\gamma = 0) = h. \quad (6)$$

Proof. See Lemma 2. □

6. Experimental Design

Under experimental conditions, G may be regarded as "black box," that is, we input two objects, P_0 and P_1, and we get a result, a statistic called G subject to constraints we mentioned above. The experiment, thus, does not ask whether internally the regression lines for P_0 and P_1 are statistically different (they may or may not be), what kind of regression we are using, how we measure the fractal dimension, and so forth. The G is deliberately and completely blind to these questions. The only concern for experimental purposes is whether there are systematic deviations from expectation, that is, our null hypothesis, which cannot be explained by chance. We use two controls for this purpose.

Lemma 1 suggests we can generate the stealthy paths without modifying the A^* cost heuristic directly. Indeed, per Lemma 2 we use the standard A^* from Bourg and Seemann [2] as one of our experimental controls, in this case, pathfinding without beautifying treatment. The other experimental control, the aesthetic A^*, is from Coleman [3].

Lemma 3 states that paths are distinguishable only for sufficiently large, nonzero γ. However, Lemma 3 does not suggest *how* to choose γ. Thus, we selected $\gamma = 10\%$ for one run and $\gamma = 15\%$ for another run as these seemed to us a reasonable basis for experimental and illustration purposes. Note that a "run" is a series of "trials" which we explain below.

These pathfinding algorithms, standard, aesthetic, and stealthy, are embedded, respectively, in multiroom virtual worlds, W, generated by the Wells [24] random level generator. The Wells level generator takes as input a "level" which defines the width and height of the world. It also takes as input a seed which randomizes the configuration of the world in terms of rooms and interconnecting hallways as $\{B^j\}$. The Wells level generator also creates A^{start} and A^{goal}, respectively, in the first and last rooms. We use the three types of pathfinding (i.e., aesthetic, standard, and stealthy) to find a path from A^{start} to A^{goal} in each world. Finally, for each world we compute $G(P_1|P_0)$, where P_1 is a stealthy path and P_0 a reference or control path, either aesthetic or standard.

To compute G, we convert the virtual world to a PNG [16] image. We generate level "10" worlds which are 50×50 tiles. Each tile is 10×10 pixels and each A^i and B^j occupies a single tile. A^i are ovals 10 pixels in diameter and B^j are squares 10 pixels in length. This is the input to FracTop which calculates the fractal dimension, $D(P)$, using reticular cell counting. Finally, we then compute G according to (2).

Each random multiroom virtual world, W_m, is an independent Bernoulli trial. A trial is successful provided that $G({P_1}^{\text{stealthy}}|{P_0}^{\text{aesthetic}}) < 0 \wedge G(P^{\text{stealthy}}|{P_0}^{\text{standard}}) > 0$. The trial is a failure otherwise. If s is the number of successes in N trials and f is the number of failures where $N = s + f$, then the null hypothesis is $H_0 : s \leq f$. To conservatively estimate the P-value, we use the one-tailed Binomial test, a nonparametric test [6] for $N = 100$ trials in two runs, one for $\gamma = 10\%$ and one for $\gamma = 15\%$.

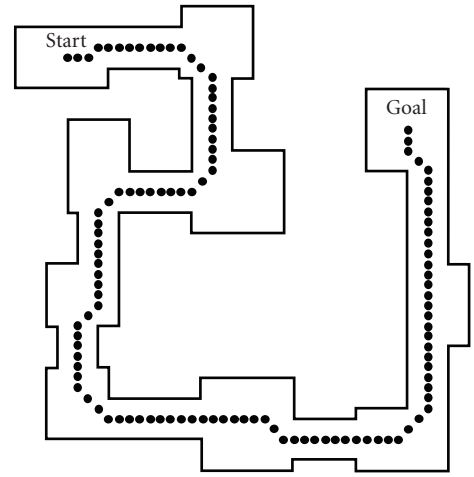


Figure 2: Aesthetic pathfinding with beautifying treatment for trial 18.

We also analyze stealthy paths compared to each other, namely, less stealthy ($\gamma = 10\%$) versus more stealthy ($\gamma = 15\%$) pathfinding. In this case, a trial is successful if $G({P_1}^{\text{stealthy } 15\%}|{P_0}^{\text{stealthy } 10\%}) \neq 0$ and a failure if $G({P_1}^{\text{stealthy } 15\%}|{P_0}^{\text{aesthetic } 10\%}) = 0$. Again, we have $H_0 : s \leq f$.

7. An Example

To make these ideas clearer, we go through a randomly selected trial, number 18. Namely, the Wells random seed is 18. Readers can view the results of all 100 trials of 400 images online at the author's website [5]. Figure 2 shows the multiroom, virtual world and aesthetic pathfinding for this trial from A^{start} to A^{goal}. The • symbols represent "bread crumbs" which constitute the path in the time domain.

Figure 3 shows the same random virtual world with stealthy ($\gamma = 15\%$) pathfinding.

Figure 4 shows the stealthy path for $\gamma = 10\%$. Notice that the difference between 10% and 15% is the little "jog" in the upper-left quadrant. We discuss this further in the conclusion section.

In general, one can easily see the difference between stealthy paths and the control paths. The standard path swerves from wall to wall seeming almost to wander. In a sense, the standard path is making random choices since the wall does not affect the cost heuristic. Yet in the stealthy case, the wall is sought out where possible. This movement gives a visual impression of avoiding opening spaces, that is, middle of the room or hallway. In other words, the aesthetic path is less covert compared to the standard one. The stealthy ones, however, appear more covert than both aesthetic and standard paths.

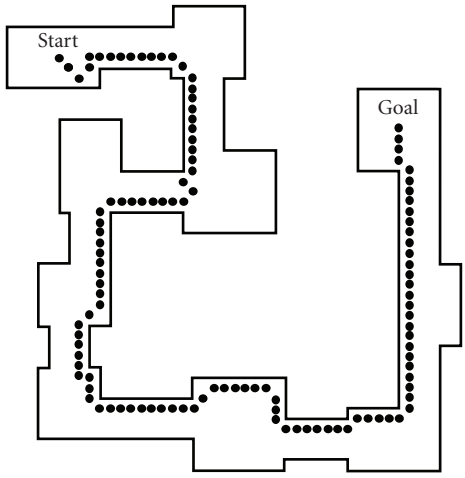


Figure 3: Stealthy pathfinding for trial 18 and γ=15%.

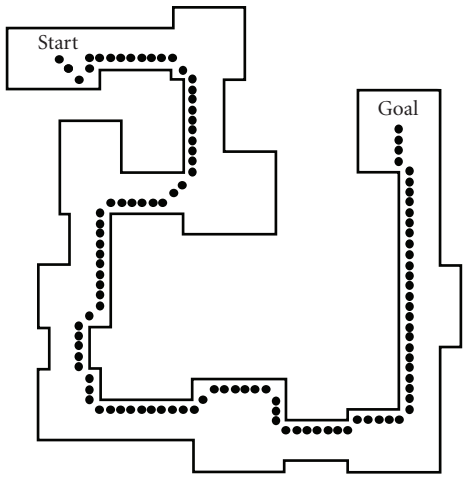


Figure 4: Stealthy pathfinding for trial 18 and $\gamma = 10\%$.

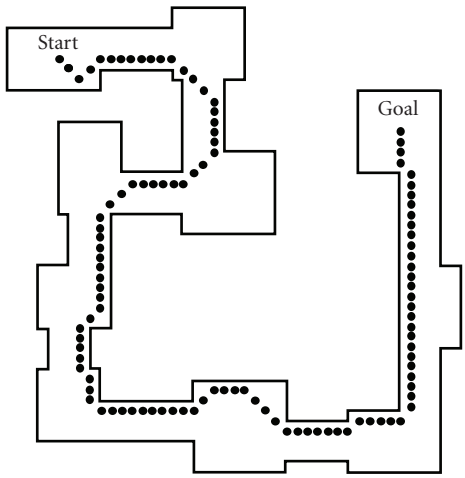


Figure 5: Standard pathfinding (i.e., with no beautifying treatment) for trial 18.

Table 1: Fractal dimensions, $D(P)$, for each path, P, of trial 18.

Figure	P	$D(P)$
2	Aesthetic	1.557638
3	Stealthy (15%)	1.550786
4	Stealthy (10%)	1.549607
5	Standard	1.547505

Table 2: $G(P_1|P_0)$.

	P_0			
P_1	Aesthetic	Stealthy 15%	Stealthy 10%	Standard
Aesthetic	0			
Stealthy 15%	0.006852	0		
Stealthy 10%	0.008031	0.001179	0	
Standard	0.010132	0.003281	0.002101	0

Table 1 gives a quantitative assessment, namely, the fractal dimension, $D(P)$, for each path, P, according to the four objectives shown in Figures 2–5.

Table 2 gives $G(P_1|P_0)$ as the intersection of rows and columns starting in the control or "$D(P_0)$" column gives G for this trial.

We organized Table 2 for readability; namely, the lower triangle is a positive transpose of the upper triangle (not shown). The zeros along the diagonal represent $G(P_1|P_0) = 0$, where $P_1 = P_0$. For instance, the aesthetic path compared to the stealthy path as the reference, $G(P_1^{\text{aesthetic}}|P_0^{\text{stealth 15\%}}) = 1.557638 - 1.550786 = 0.006852$.

From a purely quantitative perspective, Table 2 shows the objectives in order of decreasing fractal beauty and one can readily see that the null hypothesis, H_0, is not supported by this single trial. Both stealthy paths' fractal dimensions are numerically between the aesthetic path and the standard path.

In other words, the numerical relationships are somewhat different from visual impressions. We do not attempt to explain this phenomenon here. We only note that the movement patterns are visually distinct and consistent, and as we observe below, statistically significant from the model's perspective.

8. Results

The raw data consists of 400 results: 100 standard paths, 100 aesthetic paths, 100 paths for $\gamma = 10\%$, and 100 paths for $\gamma = 15\%$. The full data sets may be found online at the author's website [5]. Figure 6 gives the histogram distribution of $G(P_1^{\text{stealthy}}|P_0^{\text{standard}})$ and $G(P_1^{\text{aesthetic}}|P_0^{\text{stealthy}})$ for the $\gamma = 15\%$. run. The proportions of mean G are 0.001675 ± 0.001661 and 0.007203 ± 0.002285, respectively. (The notation "$\pm$" is the standard deviation.)

Figure 7 gives the statistical histogram distribution of $G(P_1^{\text{stealthy}}|P_0^{\text{standard}})$ and $G(P_1^{\text{aesthetic}}|P_0^{\text{stealthy}})$ for the $\gamma = 10\%$ run. The proportions of mean G are 0.001275 ± 0.001503 and 0.007603 ± 0.002257, respectively.

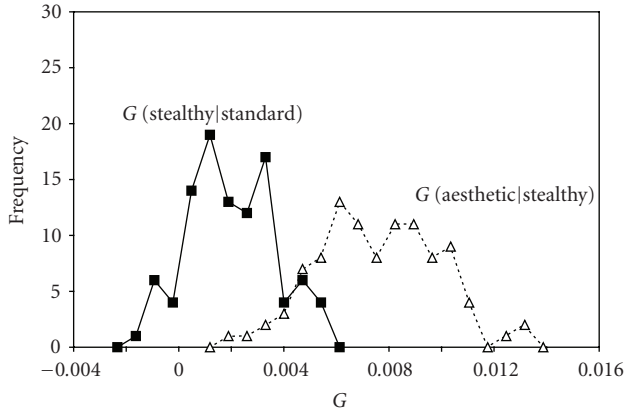


Figure 6: $G(P_1^{\text{stealthy}}|P_0^{\text{standard}})$ and $G(P_1^{\text{aesthetic}}|P_0^{\text{stealthy}})$ for $\gamma = 15\%$.

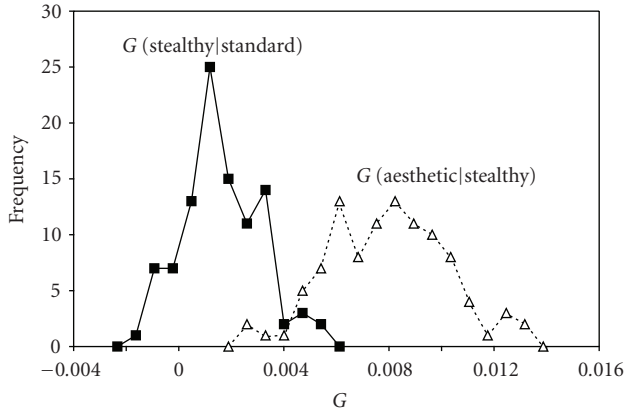


Figure 7: $G(P_1^{\text{stealthy}}|P_0^{\text{standard}})$ and $G(P_1^{\text{aesthetic}}|P_0^{\text{stealthy}})$ for $\gamma = 10\%$.

Table 3: Number of successes and failures and P-value.

γ	Trials	s	f	P
15%	100	84	16	$2.391E-13$
10%	100	78	22	$2.169E-09$

These two charts are generally similar. They both show that stealthy paths tend to have more fractal beauty than standard ones, while aesthetic paths have more fractal beauty than stealthy ones. The distribution is somewhat more dispersed for $\gamma = 15\%$ compared to $\gamma = 10\%$ when the standard path is the control. Yet, this is precisely what Lemma 3 predicts.

Table 3 gives the results in terms of the number of successes (s) and failures (f) and the P-value based on the one-tailed binomial test.

Thus, we can reject the null hypothesis and accept its logical alternative. Namely, stealthy paths are unique in terms of their aesthetic value.

Table 4 addresses the question of how less stealth ($\gamma = 10\%$) versus more stealth ($\gamma = 15\%$) affects pathfinding. The P-value is based on the one-tailed binomial test.

Table 4: Number of successes and failures and P-value.

γ	s	f	P
15% versus 10%	64	36	$1.759E-03$

The data in Table 4 suggests that stealth effects $\gamma = 15\%$ versus $\gamma = 10\%$ are unique among themselves. In other words, there is a measurable, statistically significant difference.

9. Conclusions

We have shown that stealthy pathfinding is a unique aesthetic objective in relation to controls which have beautifying treatment and no such treatment. There is also a small but nevertheless statistically significant difference between the two stealth effects, $\gamma = 10\%$ versus $\gamma = 15\%$. In fact, a closer inspection of the data suggests that the "jog" in Figure 3 is *the* difference. Future research might seek to better understand this more clearly.

We noted that the quantitative pattern measured by the model is somewhat different from visual inspections of the virtual worlds. This discrepancy is consistent but seemingly counterintuitive. Future work might set up further experiments to explore the matter further.

We chose γ on the basis of trial and error. In fact, after collecting the data for $\gamma = 10\%$ versus $\gamma = 15\%$, we subsequently tried other values, for instance, $\gamma = 5\%$ versus $\gamma = 20\%$. We found no differences compared to $\gamma = 10\%$ versus $\gamma = 15\%$, respectively. We speculate that the range of γ effectiveness is constrained by the virtual world size. Future efforts might study γ more systematically in relation to parameters which generate the virtual world.

Acknowledgments

The author thanks Maria Coleman for reading the initial draft and the reviewers for providing valuable commentary and feedback.

References

[1] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical pathfinding," *Journal of Game Development*, vol. 1, no. 1, pp. 7–28, 2004.

[2] D. M. Bourg and G. Seemann, *AI for Game Developers*, O'Reilly, Sebastopol, Calif, USA, 2004.

[3] R. Coleman, "Operationally aesthetic pathfinding," in *Proceedings of the International Conference on Artificial Intelligence (ICAI '07)*, vol. 1, pp. 159–163, CSREA Press, Las Vegas, Nev, USA, June 2007.

[4] R. Coleman, "Fractal analysis of pathfinding aesthetics," *International Journal of Simulation and Modeling*, vol. 7, no. 2, pp. 71–80, 2008.

[5] R. Coleman, "Fractal analysis of stealthy pathfinding: experimental data," May 2008, http://foxweb.marist.edu/users/ron.coleman/faspdata/.

[6] W. J. Conover, *Practical Nonparametric Statistics*, John Wiley & Sons, New York, NY, USA, 2nd edition, 1980.

[7] D. Higgins, "Pathfinding design architecture," in *AI Game Programming Wisdom*, S. Rabin, Ed., pp. 114–121, Charles River Media, Hingham, Mass, USA, 2002.
[8] S. Hunter and J. Jacobus, *Modern Art: Painting, Sculpture, Architecture*, Abrams, New York, NY, USA, 3rd edition, 1992.
[9] H. Jelinek and D. Cornforth, "FracTop v.0.3B," December 2006, http://seal.tst.adfa.edu.au/~s3165516/Fractop.
[10] I. Kant, *The Critique of Judgment*, Oxford University Press, Oxford, UK, 1978, translated by J. C. Meredith.
[11] B. Mandelbrot, "How long is the coast of Britain? Statistical self-similarity and fractional dimension," *Science*, vol. 156, no. 3775, pp. 636–638, 1967.
[12] B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman, San Francisco, Calif, USA, 1982.
[13] I. Millington, *Artificial Intelligence for Games*, Morgan Kaufman, San Mateo, Calif, USA, 2006.
[14] D. Oliver, *Fractal Vision: Put Fractals to Work for You*, Sams, Carmel, Ind, USA, 1992.
[15] M. Pinter, "Toward more realistic pathfinding," October 2006, http://www.gamasutra.com.
[16] Portable Network Graphics, July 2007, http://www.libpng.org/pub/png/.
[17] S. Rabin, "Aesthetic optimizations," in *AI Game Programming Gems*, M. DeLoura, Ed., pp. 264–271, Charles River Media, Hingham, Mass, USA, 2000.
[18] D. Rockmore, "The style of numbers behind a number of styles," *The Chronicle of Higher Education, Section: The Chronicle Review*, vol. 52, no. 40, p. B10, 2006.
[19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ, USA, 2003.
[20] M. Schroeder, *Fractals, Chaos, and Power Laws*, W. H. Freeman, New York, NY, USA, 1992.
[21] P. Soille and J. G. Rivest, "On the validity of fractal dimension measurements in image analysis," *Journal of Visual Communication and Image Representation*, vol. 7, no. 3, pp. 217–229, 1996.
[22] B. Stout, "Smart moves: intelligent pathfinding," September 2006, http://www.gamasutra.com.
[23] B. R. Tanner, E. Perfect, and J. T. Kelley, "Fractal analysis of Maine's glaciated shoreline tests established coastal classification scheme," *Journal of Coastal Research*, vol. 22, no. 5, pp. 1300–1304, 2006.
[24] M. Wells, *J2ME Game Programming*, Thompson, Boston, Mass, USA, 2004.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 837095, 18 pages
doi:10.1155/2009/837095

Research Article

Games and Agents: Designing Intelligent Gameplay

F. Dignum,[1] J. Westra,[1] W. A. van Doesburg,[2] and M. Harbers[1,2]

[1] *Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands*
[2] *TNO Defence, Security and Safety, P.O. Box 23, 3769 ZG Soesterberg, The Netherlands*

Correspondence should be addressed to F. Dignum, dignum@cs.uu.nl

Received 27 April 2008; Revised 20 August 2008; Accepted 2 December 2008

Recommended by Abdennour El Rhalibi

There is an attention shift within the gaming industry toward more natural (long-term) behavior of nonplaying characters (NPCs). Multiagent system research offers a promising technology to implement cognitive intelligent NPCs. However, the technologies used in game engines and multiagent platforms are not readily compatible due to some inherent differences of concerns. Where game engines focus on real-time aspects and thus propagate efficiency and central control, multiagent platforms assume autonomy of the agents. Increased autonomy and intelligence may offer benefits for a more compelling gameplay and may even be necessary for serious games. However, it raises problems when current game design techniques are used to incorporate state-of-the-art multiagent system technology. In this paper, we will focus on three specific problem areas that arise from this difference of view: synchronization, information representation, and communication. We argue that the current attempts for integration still fall short on some of these aspects. We show that to fully integrate intelligent agents in games, one should not only use a technical solution, but also a design methodology that is amenable to agents. The game design should be adjusted to incorporate the possibilities of agents early on in the process.

1. Introduction

The gaming industry has changed dramatically over the past few years. Where the development focus used to be on the graphical possibilities of the games, that is, the naturalness of the image rendering, the near movie realism of the graphics now increasingly contrasts with the rather primitive and unnatural behavior of the characters. With behavior, we do not mean the animation of the character, but its cognitive behavior, that is, the reaction and interaction with other characters, the consistency over time of its goals, and so forth. Therefore, the focus is now shifting toward more natural behavior of the game characters. This shift changes the focus on the techniques to be used as well. Whereas geometric techniques and graphics were the prime focus, now it seems the time to introduce more serious AI techniques, see for example [1]. We see an increasing use of techniques such as fuzzy logic and neural networks to enhance the decision functions of the characters, see [2] for an example. These features are very useful to make individual behaviors look more realistic and in some cases make them blend into a crowd in a natural way.

The game developers community has also recognized the importance of making characters appear intelligent over longer periods of time. Finite state machines are most often used to model the life cycle behavior of a character. Each state describes an important state of the character that determines its choice of available actions. Although this works fine for simple behavior, Orkin [3] realized that more flexible planning is needed for complex behavior. In F.E.A.R., planning techniques based on STRIPS [4] and goal-oriented action planning [5] are introduced. This leads to more natural behavior because the goals of the character are separated from the plans generated to reach the goal. Therefore, the failure of a plan does not directly lead to giving up a goal but rather leads to generating an alternative plan including the new information about the world that led to the failure of the first plan. Interestingly enough though, little reference is made in this work to the research performed in the agent community about dynamic and real-time planning, which would be directly applicable such as [6].

Despite these examples, few commercial games have focused on making characters within games behave more natural on a *cognitive* level. Probably the main exception was

the Soar Quakebot build in SOAR for Quake II. However, this was done (successfully) as an academic project, but the Soar Quakebot was not incorporated in later commercial versions of Quake. This is partly explained by the fact that Quake like most video games does not require more complicated behavior than that of film characters played by actors like Sylvester Stalone or Arnold Schwarzenegger. Not much intelligence is needed to emulate that behavior yet. However, if we want to move beyond the shooting and fighting games toward games in which multiple characters interact naturally over extended episodes or serious games to train people leading teams in stressful situations, we need more cognitive believable behavior of the characters. One of the problems mentioned in [3] is the implementation of believable and natural communication between characters. This can be done by giving them additional information, not available to normal players. Also the use of special environmental characteristics can provide the illusion that characters are cooperating, while they merely all react to the same environmental cue. These features are for instance, used in F.E.A.R. to create a realistic appearance. However, this trick can only be used in well-defined environments and everything has to be preprogrammed.

The more complex the games become and the more elaborate the interactions between the characters during the game, the more difficult it will become to design these characters without the use of specialized tools geared toward implementing intelligent agents in a modular way. We aim for characters that are programed using agent technology that actually incorporates deliberation on actions and cooperation, rather than simulated intelligence through clever tricks. Thus this seems an excellent area for applying intelligent agent technology such as that being developed within computer science faculties at the universities, which already for a decade has developed models, techniques, and tools to design software based on design concepts such as goals, intentions, plans, and beliefs. Some first attempts to connect game engines with this type of agents have been made, for example [7, 8]. However, these are very specific to game engine and agent type, no general solutions have been proposed. The main issue of this paper is thus, given that it makes sense to use ideas from agent research in gaming, as seems to be supported by the growing amount of work in games that incorporates (parts of) agent concepts and technologies, what would be necessary to make use of the agent technology as developed for the multiagent platforms?

The techniques used in agent technology do not seamlessly fit with those used in game technology. Agent technology has hardly bothered about efficiency issues up till now. Most applications are not real-time ones or still have large time scales. Moreover, agent technology usually assumes distributed control and a certain level of autonomy of the agents. This is in stark contrast with game technology in which the game engine dictates the application, and strict time constraints are used in order to render the images naturally and efficiently.

In this paper, we explore the possibilities to join the game to existing agent technologies despite some inherent incompatibilities. We will focus on some of the most apparent problem areas and also show how they can be solved in a structural way. A main assumption of this paper is that we want to use agent platforms and the associated technology to develop the intelligent agents, because platforms such as JADEX, Jack, Jason, and 2APL [9] provide optimal support for developing the agents themselves. This will, therefore, support the design principle of separation of concerns, which is important for complex systems. The way to design a complex system is to separate different concerns and tackle them separately using the most appropriate tools for that part and joining them later. This principle can already be seen in the game technology where the game engine is built from physics engine, animation engine, and so forth, each taking care of one part of the design of the game. So, the challenge is to connect the agents developed on these platforms to the game engines on which the rest of the game is developed. The work reported in this paper is based on our experiences of connecting the 2APL platform to several game engines. These experiences led us to the conviction that in order to fully integrate agents in games, one should not only use a technical solution, but also a design methodology that is amenable to agents. We aim to support this claim as well in this paper. The areas that we will specifically look at are: synchronization, information representation, and communication.

As the agents will run in separate threads from the game engine (in principle using the agent platform), the actions of the agents and the game engine also have to be synchronized explicitly, for example, in order to make the agents behave according to the laws of physics. The problem of synchronization is of course not new; neither will we present completely new solutions. However, the solution should take the peculiarities of the games and agents into consideration.

Information filtering is needed to provide both the agents and the game engine with the right type of information at the right time. Whereas the geometric information might be the most important for the game engine, the agent wants to get its information at a higher knowledge level. For example, instead of knowing the rotation angle of a rectangular object, the agent just wants to know that the door is open. Conversely, the agent might perform an action to move a fire hose toward the house which has to be translated to more geometric actions that can be used by the game engine. The idea of using different knowledge levels to solve different types of problems dates back to Allen Newell [10] who distinguished, for example, a biological, cognitive, rational, and social level. Each knowledge level represents the information in a format that is suited for that particular type knowledge and may also contain its own type of problem solving methods. We actually propagate the same idea and claim that agents need a different (cognitive or rational) knowledge level from the game engine, which uses a biological (or physical) knowledge level.

Communication is the last area that we will consider explicitly. We will mainly look at communication between different characters in a game. In most games, coordination between characters is preprogrammed and thus communication is only needed on a small scale. In multiagent systems, communication is one of the pillars of the whole system

and thus takes a prominent place in both design as well as technology. We will show how communication can be adequately integrated with the game engine to provide the agents with easy communication, while keeping it visible for the game engine.

The rest of the paper is ordered as follows. First, we will discuss the state of the art with respect to games and agents and identify problem areas. In Section 3, we describe some applications of (serious) games that rely on intelligent behavior and are currently difficult to achieve, but will be more readily attainable using our approach. In Section 4, we will describe our vision on connecting games and agents, using three different perspectives to alleviate the problems of Section 2 and enable games described in Section 3. In Section 5, we will discuss the different parts of our approach and indicate their contribution to the type of gaming scenarios described in Section 3. Finally, we will draw some conclusions and sketch directions for future work in Section 6.

2. State of the Art

In this section, we discuss several approaches to the integration of agents in game engines. As said in the introduction, many games already advertise the use of AI; however, the meaning of the term agent differs between game developers and AI researchers. In the next subsection, we discuss the type of agent we will focus on in this paper. Subsequently, we discuss different ways in which agents are connected to game engines.

2.1. Software Agents. In the game developer community, the term software agent usually refers to some character or unit within the game. In contrast, in the area of agent research, many definitions of software agents are used, which can lead to confusion when the term is used by different communities. (For some attempts to get to a common definition and characterize agents, see, e.g., [11]). However, there are some features of agents that are generally accepted in the community, which we will also adhere to in this paper. Software agents should be autonomous, proactive, reactive, and socially able. In this paper, we assume the following, more specific, definition of software agent: a software agent is a piece of software that has its own goals available (is proactive) and will try to achieve them without intervention of a user or other program (is autonomous), while sensing its environment and reacting to possible changes (is reactive). Additionally, agents in a multiagent system (MAS) are not centrally controlled, execute asynchronous, and should be able to communicate with each other, the user(s) and the environment. Software agents might be able to learn and adapt, but we do not consider these features as essential for agents.

The above definition mentions the generally accepted features of agents, but of course is still quite vague. However, it does give an indication of what type of features one generally expects in agents. Without going into a complete classification of agents, we do want to mention a few types of software agents that are already used in the context of gaming. Most important are the virtual agents or believable characters. They are especially useful for user interfaces and as such the emphasis is on natural interaction of the characters with persons, see [12, 13] for examples of these types of agents. The goals of these types of agents exist only implicit in the way that the rules with which they react to the environment are modeled and ordered in a way to resemble the fact that they have a goal. Because these types of agents usually have only one goal (something like assisting the user to understand or use the system), this will work fine. However, this approach fails when the character has more complicated goals or several goals that are competing. This happens particularly when more than one virtual agent is present at the same time and they have to cooperate which is usually avoided in this type of applications.

Much work on using multiple agents with (relatively) simple behavior is done in the area of (agent-based) social simulation (see, e.g., [14]). These agent systems focus on the emergent behavior of the system as a result of the interactions of the agents according to simple rules. Some work where multiple agents have been used in a simulation environment for training is [15]. In this work, the agents represent individuals or groups that interact in a virtual village, region, or country. Their goal is to study how the behavior of groups influences that of other groups or individuals. In these simulations, the agents are not really autonomous. They react to their environment through relatively simple rules. More importantly they do not plan but only execute actions one by one. Planning can be simulated by manipulating the environment in such a way that sequences of actions are forced after the first action is taken. However, it is difficult to program long-term goals in these agents.

In the research on multiagent systems (MASs), the starting point is that each agent represents a point of view or party with its own goal. Therefore, usually each agent runs in its own thread such that it can be autonomous. Also, agents usually contain some mechanism to deliberate about which action to take next in order to reach their own goal. The interactions between the agents emerge from the fact that the goals of the agents are not independent and thus the agents need each other to achieve their goals. Therefore, the design of agent interactions in such a way that all agents can reach their goal is of prime importance. This is illustrated by the amount of game theory-related research reported at the recent MAS conferences [16]. In MAS, the communication facilities play a crucial role. Because not all interactions are preprogrammed, a high degree of flexibility is needed to handle the communication. The de facto standard communication language is the FIPA ACL [17], which is based on speech act theory and can be used to pass information, but also to request or order actions. MAS platforms support communication by providing addresses of all agents, delivering messages in the right order, and so forth. The most widespread MAS platform is JADE(X), which is provided as a library of JAVA classes and fully supports the use of FIPA ACL communication (and thus provides easy interoperability with other FIPA ACL compliant platforms).

Good examples of applications of MAS are logistics and virtual organizations. In these business applications, the benefits of representing the stakeholders by their own agent that pursues the goal of that party while interacting with the other parties (either cooperatively or competitively) becomes obvious. It is this kind of MAS that we are aiming to connect to the games. Taking this type of MAS as a starting point provides a means to design each virtual character with its own goal, while being able to interact with other characters to reach its goal.

Also in MAS, there are several types of agents. We are particularly interested in *intelligent* agents, which will use some form of logic to perform their deliberation. That is, they are able to reason about their own goals and plans, to check which plan is best to achieve their goal given the current situation of the world, and to replan when the situation changed. The most well-known type of intelligent agents is the so-called BDI agent [18], which are specified (and sometimes implemented [19, 20]) in terms of the agent's beliefs, desires, and intentions. We believe that the BDI agents are most suitable to implement consistent long-term intelligent behavior in games. They seem a natural extension to the work started by the use of goal-oriented action planning in gaming as they also make explicit use of goals and planning. However, they also incorporate mechanisms to effectively use communication and other interaction mechanisms in their action deliberation.

Some platforms that are more geared toward the use of agents for cognitive simulation (and thus, like the BDI agents, seem suitable for use within the gaming area) are SOAR [21] and ACT-R [22]. In this paper, we do not commit to a particular platform, but rather try to propose a more generic framework that can be used by most agent types and platforms. We thus will only refer to properties that are shared by most (well-known) agent platforms.

2.2. Connecting Games and Agents. Current work on combining agent systems such as the described above and game environments either uses a server or client-side approach. The server-side approach can be said to be the traditional approach used in game design. In server-side approaches, the decision-making process of the agents is usually completely integrated into the game, resulting in agents that have to make decisions within one time step of the game loop. As such, server-side approaches have not made use of the available agent systems. Examples of this approach are Quake III [23], Never Winter Nights [24], F.E.A.R. [25], and Bos Wars [12]. In contrast, agents in client-side approaches are separate applications using the network information that is usually sent to a client game (a game instance that connects to a server for the world information, such as the one used by the human player). Some examples of this approach are Gamebot [26] connecting to Unreal Tournament 2003, Flexbot [27] connecting agents to Half-life, and Quakebots [28] in Quake II. Most of these types of implementations are made for research purposes.

Intelligent human-like behavior is important in the first person shooter games because in the newest ones, the agents control single avatars just like humans can control their avatar. Thus bots should be intelligent enough to perform in a way that makes their avatar resemble an avatar of a human player. In real-time strategy games, a whole team is controlled by a single agent. The team members are just executing basic instructions received from this top level control. In this setting the emphasis is more on the quality of the strategy and winning the game than on whether the strategy resembles that of human players. Finally, the pace of first person shooter games is higher than in most other game genres, necessitating quicker decision making and use of heuristics. For some games, it is claimed that more sophisticated agent technologies are used, but this is difficult to verify because most games are not open source. Most games also have no publications from the creators and third party publications are often inconsistent. So, in this paper, we use a rather old game (Quake III), but we believe very prototypical for this approach, to illustrate our point, because it was the only open source game that we could access and thus reliably discuss.

In order to get a better understanding of the state-of-the-art approaches, we will analyze Quake III as a prototypical example of the server-side approach and one of the few of which the code is open source and thus inspectable and Gamebots as a good example of the client-side approach. For both approaches, the analysis will be made according to three major aspects: synchronization, information representation, and communication.

2.2.1. A Server-Side Approach: Quake III. In Quake III, the agents are completely integrated in the default game loop in the same way as the physics engine, the animation engine, and rendering engine. The agent's decisions are defined by a sequence of method calls, and the methods return the action that has to be performed at that time step. Direct method calls can be used for many different decision-making processes, for example, hard coding approaches, directly specifying what to return with a certain input; fuzzy logic, mapping the right output to a certain set of input variables; or finite state machines, identifying the situation the agent is in and executing the corresponding method call. Independent of the particular decision-making strategy, the whole process is completely synchronized. This limits the complexity of behavior because in a synchronized process all decisions have to be made within one time step, and complex decisions would slow down a game too much.

Figure 1 gives an impression of the implementation of agents in Quake III. On the lowest level in the figure, a translation from the raw engine data to a representation more suitable for agents has been created, called the area awareness system (AAS). The heart of the AAS is a special 3D representation of the game world that provides all the information relevant to the bot. The AAS informs the bot about the current state of the world, including information relevant to navigation, routing, and other entities in the game. The information is formatted and preprocessed for fast and easy access and usage by the bot. For instance, to navigate the bot receives information from the AAS about the

locations of all static items, and it can ask the AAS whether a certain location is reachable. The AAS is responsible for route planning. The first level also executes the actual actions of the agent and facilitates the decision process of the agents. However, the agents are highly dependent on the data they can extract from the AAS, for example, an agent cannot decide to take another route to a certain item. To illustrate the importance of the linkage between the engine and the agents, this part constitutes over 50% of the entire agent code.

On the second and third levels of the architecture, the information from the AAS can be used to check whether the bot's goals are reached or how far off they are. Depending on the character that a bot plays, the fuzzy logic control determines which of the possible paths the bot should start navigating.

Little communication between agents takes place in a normal game of Quake III; it is only used to assign roles in team play situations. Communication is implemented by using the chat system for sending simple text messages. More cooperation between agents would require improved communication facilities. Moreover, currently it is assumed that communication is always successful, which is usually not guaranteed in realistic multiagent scenarios.

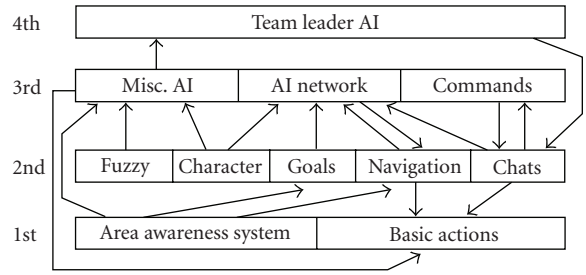


Figure 1: The Quake III bot architecture as described in the developer documentation. This figure shows the close coupling between the various levels of abstraction (Copied from [28]).

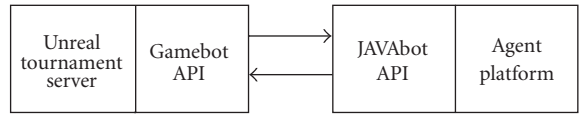


Figure 2: The Gamebot architecture showing that a server-side module is needed to translate the game data to terms the agent can process.

2.2.2. A Client-Side Approach: Gamebots. Gamebots [8] has been created as a research platform for making the connection between agent research and a computer game, namely, the Unreal Tournament environment, and is one of the most used client-side implementations. In client-side approaches, agents are running as completely separate programs from the server and are usually communicating through network sockets. Network communication between agents and other external software programs has been successfully used in other multiagent systems. Gamebots was designed for educational purposes, and therefore, multiple client implementations have been created, for example, one using the scripting language TCL, a SOAR bot, and a JAVA-based implementation.

Figure 2 shows a diagram of the different Gamebot modules in combination with the JAVAbot extension. The Gamebot API forms the extension to Unreal Tournament that is needed to connect a client-side program to the Unreal Tournament. The JAVAbot API is the client side of the coupling. Having a general JAVA API on this side facilitates the connection to most agent platforms because they are usually also JAVA based. Information is sent from the game engine to the agents through the Gamebot and JAVAbot APIs by two types of messages: synchronous and asynchronous messages. The synchronous messages are sent at a configurable interval. They provide information about the perceptions of the bot in the game world and a status report of the bot itself. Asynchronous messages are used for events in the game that occur less often and are directly sent to the agent (but do not interrupt the large synchronous message).

Gamebots is actually not a pure client-side solution because the server is also modified to supply a special world representation to the bot. There are some pure client agent implementations, but they are usually only created for cheating purposes. In this case, the processing of the data is done in the bot itself because it pretends to be a human client game. Doing this filtering on the server is more efficient because only the useful information needs to be communicated. This server modification, the Gamebots network API, performs a similar task and for similar reasons as the area awareness system in Quake III. This clarifies why the Gamebot API is specific for Unreal Tournament; it needs to know the internal representation of the game world in order to make the translation (efficiently).

The Gamebot API does not provide information about the complete environment, but only about objects that are perceivable by the bot. Thus, if a bot wants to gather information about the complete environment, it has to (physically) explore it. To navigate, for example, the agent receives information about predefined navigation nodes in the game map, but only the currently observable nodes are returned. The agent thus does not know what exists around the corner, let alone that it can reason about it. Due to the representation choices made in Gamebots, information about the environment has to be stored at the agent side of the system. This results in large differences between the agent's representation of the environment and the actual environment of the game engine. For complex bots, the information provided by the Gamebot API quickly becomes too limited to make intelligent decisions. For example, the agent cannot know the spawning location of a certain power-up, and therefore, it cannot plan to go there.

There is no facility for communication between agents in the Gamebots API because Gamebots was not designed for adding a multiagent system with interacting agents to the game. It is allowed to add multiple agents to one game, but there are no facilities for direct interaction between these agents. It is possible to create a separate communication

system between the agents by bypassing the API and the engine. However, this solution is not only inelegant, but also restrains the game environment to have any influence on the communication. An advantage of separating the game engine and the agents in different processes is that there are no strict time limits on the reasoning process of the agents. A disadvantage of using a fixed API is that the agent receives information it does not need and it cannot access information that it might need.

2.3. Multiagent Interaction. In the previous paragraph, we have seen two examples of ways to connect agents to games. These approaches are limited to a technical way of connecting agents to a game. On the level of game design, few games have tried to leverage these approaches from the start of the game design to add multiple agents and create a more compelling game play. Current games are generally not created with multiagent interaction in mind; interaction is not implemented at all or added as an extra feature in a later phase. For games in which interaction is simple, this is not problematic. For example, Quake III has a game mode in which two teams strive to capture each others flag. The player plays one character in a team, while all the other characters, from his own and the opposing team, are computer-controlled agents. Although the interaction between agents and between agents and the player is limited, the game conveys the feeling of a dynamic interactive world. The same can be said about the communication between characters in F.E.A.R. Although the communication looks quite natural it is actually added to the interaction scene afterward. It thus serves more to enhance reality than that it has a function in the gameplay! See [3] for a description of the problems encountered.

If the interaction in a game becomes more complex and the multiagent interaction is not an intricate part of the design process, some unexpected or unbelievable behavior might occur. For instance, users who were testing the game "Elder Scrolls: Oblivion" by Bethesda games [29] noticed that if they gave one character a rake and the goal "rake leaves" and another a broom and the goal "sweep paths" this worked smoothly. But when they swapped the items, so that the raker was given a broom and the sweeper was given the rake, in the end one of them killed the other so he could get the proper item. If the communication between agents in this game would have been possible, they could have communicated about their goals, and solved their problem. In the academic community, much work has been done on sharing, exchanging, and rejecting goals [30]. So far, this has not been absorbed by the game developer community.

Current games also do not facilitate multiple agents requiring complex decision making. In order to generate agent behavior, complex computation may be required. For instance, in a real-time strategy game, an opponent agent needs to observe the playing field, assess the state of his own units, make an assessment of the strategy of its opponents, generate a strategy, form a plan to execute that strategy, coordinate plans with other agents within the same faction, and in some cases evaluate actions in order to learn from them for future battles. Depending on the algorithms used, this can take considerable processing time. Current games make high demands on computer processors in order to display graphics, simulate physics, create 3D audio, and perform network communication, amongst others. Many games are, therefore, forced to minimize the processing time used for individual agents. If each agent has its own reasoning process running in parallel to generate behavior, this can spiral out of control quickly. This is certainly the case in games with many characters in a scenario. Current games, therefore, often forego the generation of complex behavior and script the behavior of nonplaying characters. For instance, in a first-person shooter game, two computer-controlled players happen to be within equal distance of a power-up. In the current game AI design approaches, such players enter a scripted line of reasoning, resulting in the decision to retrieve the power-up. This will lead them toward the same area in the game and within the shooting range of each other. This behavior is an example of nonrealistic behavior due to oversimplification in a script.

A human player expects the entire game world to persist even when not present in a particular area. Many games have an optimized design that allows a game to be compressed to events, behaviors, and reactions that directly surround the player, and therefore, only the ones visible to the player. So when a nonplaying character falls out of the scope of the player, the game engine no longer simulates the interaction between a nonplaying character and its environment. Thus the game is optimized and the demands on computer hardware are reduced. However, simulating only parts of the game world might result in unrealistic behavior. For instance, in large first-person shooter games, the positions of guards are reset (or their behavior no longer updated) when the player has reached a certain distance. Each time the player returns to the initial area, the guards will be at the same places or even have become alive again while they were killed before. The example shows that simulating an agent depending on the position of the play can lead to discontinuities in the game world.

Conclusion. Most state-of-the-art games use a server-side model with tightly integrated agents. As we have seen, this approach restricts the reasoning time of the agents considerably. An asynchronous solution is more suitable and will be used as a starting point in the next sections. Translating the raw game data to information more suitable for the agents is done in most computer games, but usually in a very restrictive way. In Section 5.2, we propose a more flexible solution. Many of the current games do not use communication at all, and if they do, only for simple tasks and in an ad hoc fashion. Modern games are not created with multiagent interaction in mind. This results in games without or with very simple interaction, or in unexpected behavior in more complicated scenarios. We propose to make the agent interaction an intricate part of the whole development process.

3. Using Intelligent Agents

In this section, we describe some applications of (serious) games that really leverage intelligent agent technology in a way that is currently not practiced. These examples serve to illustrate the usefulness of our approach. The examples in Section 3.1 mainly focus on problems related to information representation. Section 3.2 about multiagent systems stresses the importance of finding solutions for communication issues. The area of synchronization is addressed throughout the whole section.

3.1. Serious Gaming. Besides the purpose of entertainment, games are also used for training and education. These so-called serious games are for example used for the training of pilots, soldiers, and commanders in crisis situations. The training scenarios often involve complex and dynamic situations that require fast decision making. By interacting with these games, the player learns about the consequences of his actions from the reactions of the environment and other (nonplayer) characters to his behavior. Explanations can enhance the player's understanding of a situation [31]. Several approaches of self-explaining agents have been proposed [32–34]. In addition to performing interesting behavior, such agents are able to explain the underlying reasons for it afterward. By understanding the motivations of the other characters in the game, the player learns how his behavior is interpreted by others.

An example of a serious game to which explanation capabilities could be added is virtual training for leading firefighters. In such training, the player (training to become a leading firefighter) has to handle an incident in the game, and is surrounded by virtual characters representing his team members, police, bystanders, or victims. A possible scenario is a fire in a building. During the training session, the player commands his team members to go inside a building and extinguish a fire. The player's team enters the building, but after a while he still does not see the fire shrink from the outside. To better understand the situation, he might ask the virtual characters to explain their behavior. Their possible answer is that they saw a victim inside the building, and decided to save the victim first before extinguishing the fire.

The scenario just given is described on a high level. The virtual characters get commands from the player such as *go to the building, find the fire*, and *extinguish the fire*. When they explain their behavior, they refer to abstract concepts such as priorities between different tasks (saving a victim has priority over extinguishing a fire). However, the abstract decisions that the characters make result into actions that have to be executed and visualized in the virtual environment. Instead of the description *go to the building*, more specific information is required on the implementation level, for example, the coordinates of the agent's starting position, exact path, and final position. So in order to perform actions in the virtual world, the high-level descriptions generated by an agent's reasoning process have to be translated to low-level descriptions required by the game engine.

Besides acting in the environment, agents sense their environment and information goes from the game engine to the agent. The low-level information that is made available by the engine is not immediately useful to the agents. Instead of the exact positions of all the entities and objects in the game at every time step, agents use abstractions, for example, someone is going inside a building, exploring a building takes some time, and the entity in the building is a victim who needs help. The low-level information provided by the game engine needs to be translated to concepts that are useful for the agent. For instance, information about the course of the coordinates of a character could be translated to the more abstract description that the character *enters a building*, and if a state holds for a certain amount of time steps, this could be translated to the high-level concept *for a while*. This concept has to be flexible, as the agent might decide to take an action at time "t," but the game engine can only process its action a few steps later. After translating the available low-level information to concepts that agents use, an agent itself can select which of the high-level information will influence its future actions.

For the generation of explanations about agent behavior, a high-level representation of the agent's reasoning process is needed. For instance, agents implemented in a BDI programming language appropriate for the addition of explanation capabilities. Concepts such as goals, beliefs, and plans are explicitly represented in BDI agents and thus available for reasoning and the generation of explanations. Moreover, it has been demonstrated that BDI agents are suitable for developing virtual nonplayer characters for computer games [35]. A nonplaying character however needs to act in and sense its virtual environment, in which other representations of the game world are used. The example illustrates the need of a middle layer in serious gaming, where a translation between the two representation levels takes place.

3.2. Multiagent Systems. Multiple intelligent nonplaying characters bring additional challenges to game design. Currently there are few facilities that allow efficient multiagent behavior. Issues that should be addressed are for example how an agent determines whether there are other agents in the game. If so, how can it communicate with these other agents? How does it know that a message has reached the intended agent? How is information filtered such that it allows an agent to reason about social concepts, for example, about groups, group goals, and roles within a group?

In the firefighting scenario sketched in the previous subsection, the team members of the leading firefighter (player) are intelligent agents (nonplayers). Although they have to execute the commands of the player, they still need intelligence of their own. In the first place because they might take initiatives by themselves; in the scenario the nonplaying characters decided to save the victim first instead of extinguishing the fire as the commander had told them. Second, because they act in a team and have to coordinate their actions with each other. For instance, if the group has to decide whether to go left or right, they have to communicate to each other in order to make a common decision. Or, only one of the characters needs to carry an axe for opening doors,

but the others have to know that one of the team members is responsible for this task.

Suppose that a team of firefighters goes into a building with the goal to extinguish a fire. One of the members is responsible for opening locked doors and another has to extinguish the fire. If the first carries an axe and the second an extinguisher, this will work smoothly. However, the situation in which the door opener carries an extinguisher and the fire extinguisher and axe is more complex and requires communication. The door opener has to be aware of the other character, come up with the idea to communicate with it, send the right message, wait—long enough—for the result, and finally connect the right action to it. The next action of the door opener depends on the information it receives from the fire extinguisher.

We believe that the communication between different agents in a game should go through the game engine instead of taking place on the agent platform because the effect of communication has influence on the game world itself and not only on the agents. For instance, if the two agents in the scenario successfully communicated and decided to exchange their tools, this needs to happen physically in the virtual environment as well. If communication would not go through the game engine, there is a danger that processes in the game world and between the agents are no longer synchronized. For example, if the agents agree to swap items, they would both send a message to the game engine and believe that the items will be successfully exchanged in the game world. This however is not obvious. The actual swap in the virtual world is managed by the game engine, for example, one agent puts down its tool, has its hands free to receive the other tool, and the other agent picks up the tool from the ground. For such a process, it is crucial that the game engine receives the messages from both agents at the same time, or at least connects them to each other. This can be better realized if the game engine is included into the communication loop.

In turn, physical changes in the world have effect on communication as well. For example, if the door opening agent asks a team member to take over, it expects this member to come and take his axe. By perception, the agent derives whether its colleague perceived the message and decided to assist, or if it should communicate more. The colleague might have a good reason to refuse, for example, it has to assist a third agent already. It could communicate this to the requesting agent. The timing of this communication and the action to help the third agent should be synchronized; otherwise the requesting agent might for example unjustly belief that it is being ignored. Such timing is facilitated by including communication into the game loop.

Further issues concerning careful time management include a translation of time for the game engine to time for the agents. For instance, if the door opening agent sends the message *what tool are you carrying?* To the other agent, it expects a reaction. It is not realistic to expect a response directly in the next time step, the game engine could give priority to other processes first and the other agent might need some time to reason about the question. However, the agent also should not wait indefinitely because it could be that the message never arrived, or that the other agent misunderstood the content, and so forth. So after a certain amount of time, the agent has to react, for example, by sending the same message again, or by sending a message *did you understand my previous message?* In the middle layer, a translation of time for the game engine (a number of time steps) to time for the agents (time in which a reaction could be expected) has to be made.

The examples in this subsection aim to make clear that communication is more than just an exchange of information. After sending a question or a command, the sender expects an answer or action. If it does not see an effect of its communication action for whatever reason, the sender will react on that. Decisions of agents depend on the information they receive by communication *and* perception, and their communication actions have effect on the game world *and* the behavior of other agents. Therefore, the communication processes and the actions in the game world have to be well synchronized.

4. Connecting Games and Agents, Our Vision

In Section 2, we have shown current approaches to integrate agents in game engines. It is clear that those solutions are pragmatic but do not really give room to fully use all aspects of agent technology in the game environment. In Section 3, we have illustrated how agent technology can contribute to the use of game for serious purposes and a more compelling interaction between NPC characters. To overcome issues with synchronization, information representation, and communication, we analyze the connection between game and agent technology from three different perspectives, that is, the infrastructural, conceptual, and design perspectives.

For our solution, we look at the connection between the agents and the game engine starting from infrastructural point of view. The main requirement is that on the one hand the game engine should have some control over the actions of the agents in order to control the overall game play and preserve physical realism. For instance, if an agent wants to move in a straight line to a position in the game world, but there is a wall in between the agent and that point, then the game engine will prevent the agent from moving to the point it wants to get to, that is, the agent cannot just move through walls. On the other hand, the agents should be autonomous to a certain level. For instance, if an agent is walking to a way point, but is reconsidering his decision and wants to turn back, it should not first have to walk to the way point and only there be able to turn back. Also, we want the agents to be able to keep reasoning full time and not being restricted to specific time slots allocated by the game engine.

An important consideration in the connection between the agents and the game engine is which information is available to the agent and when and how does it get that information. Moreover, we have to consider when agents can perform actions in the game and which actions are available to the agent. With respect to the latter, one should think

more in terms of abstractions than in terms of forbidden actions. For example, can an agent open a door or should it manipulate a door object position to another position? Often the translation between these types of actions is provided for the avatars steered by the user. However, it is not clear that the same set of translations applies for the nonplaying characters in the game. For example, current animation engines are capable of performing rudimentary path planning. This means that actions become available to characters to move through a room without bumping into any object with one command. These commands might not be available for the human players, but are very efficient for the nonplaying characters.

The above considerations all relate to the connection of a single agent to the game engine. In general, one would like to connect a complete multiagent system to the game in which the agents also can communicate and coordinate their actions. In order to fully profit from agent technology, one would want especially to have the agents using their own high-level communication protocols that facilitate coordination. These communication facilities are standard provided by the agent platforms on which the agents reside normally. As we have seen, the facilities for communication within the game engines are rather primitive and/or ad hoc. So they are not very suitable for this type of communication, unless we extend them considerably.

The next question thus becomes how to connect the agent platforms to the game engine. Several solutions are possible. First, one can integrate the functionality of these platforms in the game engine. In this case, the agents can be built as if they are running on an agent platform. Second, one can distribute the functionality over the game engine and the agents. This means that some rudimentary functionality is incorporated in the game engine, but the agents have to get some more elaborate communication functionalities to compensate for the loss of some features. For example, they might have to keep track of the other agents they can communicate with (storing agent names and addresses). A last option is to let the agents run on their own platform and connect the platform to the game engine. One problem with this option is that the platform runs in parallel to the game engine and all types of interactions between the agents are not available to the game engine. This might potentially lead to a loss of control and inconsistencies between the agents and the game engine.

We will opt for a position in the middle. We will transfer some of the communication functionalities to the game engine to preserve consistency and control. However, we also will keep the agents running within their own platform. This is mainly done for some other facilities provided by the platforms, such as efficient sharing of reasoning engines by the agents and monitoring and debugging interfaces for the agents. The last parts are important for designing and implementation, but can be decoupled in the runtime version of the game. In order to address all issues, we divide the connection into three stances: an infrastructural, a conceptual, and a design stance.

As indicated above, the infrastructural connection requires adjustments on both the agent as well as on the game engine side. Therefore, although the connection principles might be platform independent, the actual implementation will not be completely platform independent. The standard way to ameliorate this point is to create a middleware API. Basically, connecting agent (platforms) to game engines is not different from connecting any other software together. So, in the end, we also will make use of the means available to connect independent threads of software. However, what is different is the perspective. In most applications that connect software, there will be a single thread of control that is well defined. In our case, we want a kind of shared control that is different from traditional software solutions. It means that our infrastructural solutions should take this perspective of shared control already in mind and be as flexible as possible in order to define the way control is shared on higher levels. So, in our middleware, one can define the standard constructions that we assume to exist on both sides, but the way they work together is kept as flexible as possible. The exact sharing of control is defined in the infrastructural stance. We describe the infrastructural stance in more detail in Section 5.1.

The translations between information representations that are needed to connect the agents to the game are described using a conceptual stance. Most important will be the translation of actions of the agent into actions within the game engine and translations of changes in the world into percepts that can be handled by the agent. We aim to use the high-level architecture (HLA) standard for this purpose. This stance is described in Section 5.2.

Finally, it is important to incorporate the agents explicitly in the design method of the games. The type of data that has to be generated or kept depends crucially on the ways that the agents need to use them. Therefore, if the world is first created and the agents are only added in the end, they might not have enough information available to act intelligently. For example, if an agent has to take cover it should know the distinction between an iron bar fence and stone wall of the same dimensions. If the only data available is that there is an obstacle of certain dimensions, this information can hardly be deduced. Designing the environment with the possible actions and perceptions of the agents in mind will drastically change the way the world is created. In Section 5.3, we will show that the agent-oriented OperA framework is a good starting point for such a design methodology.

In Table 1, we summarize how the different issues that we focused on are dealt with within the different stances. In this table, we denote the technique that is used in a particular stance to deal with an issue. Please note that the issues are not all of the same type. Synchronization, for example, is a technical issue that is, therefore, not really discussed in the design stance. In contrast, communication is such a general issue that it has elements that are dealt with in all the different stances.

5. Three Stances to View the Connection

In this section, we will discuss the three stances (infrastructural, conceptual, and design stance) in our approach more

extensively. For each of them, we will indicate their contribution to gaming scenarios as described in the previous section. As argued before, the topics of synchronization, information filtering, and communication play a fundamental role in coupling games and agents. So they all will be covered in this section as well. Synchronization is mainly addressed in the subsection about the infrastructural stance. Information filtering receives most attention in the subsection about the conceptual stance. Communication involves several aspects; it is, therefore, discussed in all of the three subsections.

5.1. Infrastructural Stance. In our approach, we view the game engine and agents as asynchronous processes because, as discussed in Section 2, agents that are part of the game loop are restricted in their reasoning by time. Therefore, we believe that a synchronous approach is not suitable for intelligent agents with complex reasoning processes. Although we are investigating a coupling between two specific types of asynchronous processes, infrastructurally our case is similar to other asynchronous couplings.

There are four basic tasks that need to be performed by the infrastructure. First, information about the game environment needs to be provided to the agents to allow them to reason about the game. Second, the actions that the agents have selected to perform in the game need to be transferred to the game engine to allow them to be executed. Third, the communication between agents can be effected by the game environment and thus needs to flow from the agents, through the game engine, back to the agents. Last, the infrastructure needs to provide a central time. The latter is relatively simple and done by sending timed events to both game engine and agents.

When an agent requires information from the game engine, a distinction is made between information about static and dynamic game entities. Static entities have properties that are fixed for the duration of the game, for example, buildings, mountains, and roads. Dynamic entities contain properties that change continuously. For example, victims have changing health, firefighters change position, and fire spreads through a building. For static entities, the engine can inform that the entity is static and include the requested properties. After such a message, the agent normally does not need to update this information anymore. This thus provides for some efficiency in the information flow.

For dynamic entities, the game engine sends a message when entities become (un)perceivable for the agent. The conditions for perceivability are defined conceptually. In the filtering layer is decided which events are relevant for that specific agent. This in contrast to fixed APIs used in current work where all agents receive the same event types. After being subscribed to a dynamic entity, the game engine will keep sending updates about these properties. This mechanism prevents the agent from being flooded by information about all possible entities and their properties, while not limiting it to predefined aspects of the game world. One could see the decision-making process that selects which events are selected as part of the agent but this is not a necessity.

Table 1: Contribution of each stance to the three challenges of connecting agents to games.

	Infrastructure	Conceptual	Design
Synchronization	Event queues	Timestamps	—
Filtering	—	HLA	Ontology
Communication	Communication queue	ACM	Interaction patterns

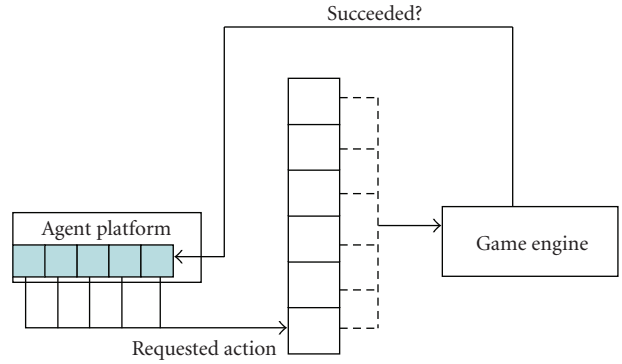


Figure 3: Event queues in the infrastructure allow both the agent platform and the game engine the flexibility to select events based on their own criteria.

In order to execute actions in the game world, an agent sends a request to the game engine. However, the agent's actions might be of a different type than the game engine's actions, for instance, open a door versus move object x to position y, z. Moreover, the timing of the request might not correspond to the game loop, so directly executing these actions in the game engine is in general not possible. Therefore, we propose to implement the actions of the agents in the game world by a queue structure which contains a description of the action plus possible timing constraints. Figure 3 shows a diagram of the information flow of the action requests. A requested action is inserted at the end of the queue to keep an ordering of the requested actions. At the beginning of each new time step of the game loop, the engine selects actions to perform. One possible approach would be to always select the actions at the top of the queue. The game engine however is able to select actions based on other criteria. For example, certain actions might be preferred by the game engine or a higher priority might be given to actions by a certain agent.

Normally agents expect an external action to behave like a method call and the agent waits for the result. But because actions in the game world are not always executed right away, do not always succeed, and sometimes have unexpected results, we separate the request of performing the actions from the result. The result of the execution of the actions is sent back from the game engine to the agent in a separate message. Agents do not have to stop their reasoning process to wait for this message. When the message arrives, the information can be used for further reasoning. This is

significantly different from normal multiagent programing. There are different ways to cope with this delayed feedback. There is no guarantee that the engine responds within a certain time limit. An agent could be programed in such a way that it assumes that the action failed if no response is received within a fixed amount of time. Or an agent can assume that all actions succeed and if a negative response is received from the engine, this information is corrected. More elaborate reasoning about this information is also possible.

When comparing this approach to the Gamebots model, it is obvious that there are some similarities. Gamebots also uses separate asynchronous processes and some adjustments can be made on the timing of information passing. However, there are some advantages to using the approach suggested in this section. The main difference is that there is a lot more flexibility on the kind of information that is passed from the engine to the agent because of the usage of a subscription model instead of the fixed API. For example, in the subscription model, the agent could subscribe to a very specific property such as the health information of another character. With the fixed information passing used in Gamebots, all predefined information is continuously sent, and therefore, such specific properties are omitted. The timing is also more flexible on both sides of the system. When using Gamebots, information is either sent to the agent at a fixed time interval or directly for synchronous or asynchronous, respectively. It is not possible to change the timing or the amount of information that is sent, although the interval between messages is configurable. In the subscription model, the agent can request information whenever it is convenient. On the game side, the engine selects the requested actions from the input queue at its own time and with its own selection criteria. In the Gamebots approach, the actions requests all have to be executed immediately in the next time step.

Communication between agents is organized in a similar way to action requests and information passing. Sending a message to another agent is treated as a type of action request, where the action consists of delivering information to another agent. Communication plays an important role in multiagent systems, which is why we prefer using a separate queue for communication requests. Again the game engine can have its own preferences about the selection of messages from this queue. The game engine determines how a communication request is handled. For example, if an agent shouts, the engine determines which agents receive this message. Similar to action requests, the agents also receive feedback about the result of sending the message. Because the game environment can influence the success and effect of the communication, it is clear that it should pass through the game engine and cannot be organized through the multiagent system platform (as is normally done).

5.2. Conceptual Stance. The second stance in our framework connects the character's mental capabilities (implemented in the software agent) to their physical counterpart (implemented in the game engine), in a similar fashion as the pineal gland was supposed to connect the mind and the body in Rene Descartes' dualist worldview [36]. In this section, we will discuss the mapping of agent reasoning symbols to game engine data.

5.2.1. Conceptual Agreement. The most important aspect of this stance consists of a translation between concepts in the game engine and the agent. For example, when an agent wants to execute the action *go to the building*, this should be translated in the game engine to *find object called building*, *check object can be reached*, *plan path to object*, *follow path to object*, and vice versa. In order to create this mapping, we need to define a consistent common representation. This representation functions as an agreement or contract between the game engine and the agent. While each has a different internal representation of the concept, both have to respect the meaning of the concept as defined in the agreement.

The agreement will cover the way the world can be perceived by the agent (game engine to agent mapping) and the way the world can be acted upon by the agent (agent to game engine mapping). These agreements are called the object perception model (OPM) and the object interaction model (OIM). They are inspired by HLA. HLA is a simulation interoperability standard [37]. HLA was designed to allow different simulations to connect and participate in a shared scenario. However, it was not designed to connect agents to simulations or games. One aspect that is required for the case of connecting an agent is filtering of data. An agent should only receive data that is relevant for the agent. For example, if an agent is fighting a fire in a building, it is of little use to receive a message that there is a player on the other side of the game world that lost his helmet. In HLA, there is only control over data distribution among participants by the use of a publish-subscribe approach. However, in the case of agents, the need for information is highly dynamic and based on the situation at hand and the line of reasoning by the agent. Therefore, the condition under which subscriptions should change needs to be represented. In the case of agents, we will extend the HLA approach with more control over data distribution. This extended control will create a more dynamic publish-subscribe approach in which a party is only subscribed to certain information in relevant situations.

First, we will describe the object perception model. The OPM represents both the entities that can be perceived (ontological representation) and the condition in which they can be perceived (qualification representation). In HLA, the common ontological representation is defined in the federate object model (FOM) which is an instantiation of the object model template (OMT). In the case of agents, we will not need many of the data types defined in the OMT and we can use a general syntax such as XML. For example, a firefighter in our scenario can observe other characters. The following XML description indicates which features of the characters it can perceive:

```
<class name="Character">
  <property>
    <name>ID</name>
    <type>number</type>
  </property>
  <property>
    <name>Distance</name>
    <type>meters</type>
  </property>
  <property>
    <name>Direction</name>
    <type>Orientation</type>
  </property>
  <property>
    <name>Tool</name>
    <type>Tool</type>
  </property>
</class>
```

Stating that one can perceive the character's distance and the tool it carries. An agent can for instance subscribe to perception messages about other characters. Only when it is relevant should the agent actually receive these messages. This is accomplished with the Poss() operator. It means that only messages will be sent when the situation satisfies some constraints. For instance, the conditions in which the characters can be perceived can be described as follows:

$$\text{Poss}(\text{Perceive}(\text{Character}, \text{ID})) \iff (\text{Dist}(\text{Character}, \text{ID}) < 150 \wedge \text{LineofSight}(\text{Character}, \text{ID}) \wedge \text{Direction}(\text{Character}, \text{ID}, \text{towards})$$

So, a character can only be perceived if it is closer than 150 meters and one looks in the right direction. Both the agent and the game engine need to interpret the OPM based on this common representation.

In the case of the game engine, the part of the game loop that sends world data to the agents contains a list of agents that have the capability "perception." This capability is described in the OIM (which we will introduce hereafter). It also contains a list of objects of the type "Character." It compares the x, y, z positions and checks if the distance is smaller than 150 meters, checks if there is a line of sight between each agent and each character and checks the relative orientation of agent and character. If these actions satisfy the perception rule in the OPM, the game engine sends an asynchronous message to the queue of the perceiving agent. The message contains the "Character" object with the properties as defined in the OPM.

The mapping of concepts between game engine and agent can be facilitated by software tools that automate some of such mappings. For example, Kynapse from Kynogon [38] is able to analyze geometric data and extract path planning information from this data. This in fact is an automated step to translate game engine information to concepts with which an agent can reason. For information other than path planning, additional tools could be developed.

In the case of an agent created in an agent language such as 2APL, it will interpret the OPM straightforward as an incoming event of the type perceive with a number of parameters.

Event (Perceive (Character, ID, Distance, Direction, Tool), TIMESTAMP). The timestamp indicates the time the event was received. These types of events are stored in the so-called event base of the 2APL agent. It can use reasoning rules to decide what to do with these perceptions. For example, it can update its belief base each time such an event is received, but it can also restrict updates to characters that are closer than 50 meters or of which the distance changed more than 100 meters.

Second, we will describe the object interaction model. The OIM represents the capabilities of the agent to interact with the world. It denotes the possible interactions, the conditions under which they are possible, and the effects of an action. In HLA, interactivity between simulations is achieved through sending specific interaction events. These interactions are messages specifying events that happen in a simulation. Based on the subscriptions of a simulation, it will receive all corresponding events. In the case of agents and games, we again need more precise control over which interactions are relevant for an agent. This helps reduce processing load on the agent side and optimize the game on the game engine side. We again take inspiration from HLA and define the interaction relevant properties of objects using XML. We then extend this with rules specifying constraints and consequences concerning the actions. We continue the firefighter example and describe an agent (which can be viewed as an object that can interact) that can open doors:

```
<Agent name="Door-opener">
  <general>
    <property>
      <name>HoldsOpeningTool</name>
      <type>Tools</type>
    </property>
  </ general>
  <physical>
    <property>
      <name>height</name>
      <type>meters</type>
    </property>
  </physical>
  <sensor name="eyes">
    <property>
      <name>Range</name>
```

```
        <type>meters</type>
      </property>
    </sensor>
    <capability name="Open door">
      <property>
        <name>target</name>
        <type>Door</type>
      </property>
    </capability >
</Agent>
```

We force agreement on the circumstances before and after the action in a similar fashion as done in [39] by specifying pre- and postconditions of the action:

$$\text{PRE: Poss(OpenDoor(Agent, Door))} \iff \text{Closed(Door)} \wedge \text{Distance(Agent, Door)} < 1 \wedge \text{Holds(Agent, Axe)}$$
$$\text{POST: Done(OpenDoor(Agent, Door))} \implies \text{Open(Door)} \wedge \text{Poss(Backdraft(Door))}.$$

So the agent can open a door if the door is closed and it stands near to the door and is holding an axe. If a door is opened, its state is changed to open and the agent is automatically subscribed to messages that indicate a back draft explosion occurred.

Similar to perception, both the agent and the game engine will need to interpret the OIM. For the game engine, this means that the "Door-opener" agent will be subscribed to asynchronous messages (as described in the previous section) about "Door" objects in its vicinity. This is interpreted from the appearance of door objects both in the agent properties and in the interaction rules. Additionally, the game engine processes code to execute "OpenDoor" actions sent by the "Door-opener" agent (i.e., changing the status of the door to open) while it ignores such actions from other agents. It changes the physical representation of the door by turning it ninety degrees. Following this, the game engine recalculates fire and heat intensity and the oxygen level in the room behind the door. If a door is opened in a room that is very hot but contains little oxygen, the game engine will produce a message indicating that a back draft explosion occurred.

The link to the agent side has to be made through the capabilities of the agent. In 2APL, this is an easy process because the capabilities of an agent are given explicitly in the agent program with their pre- and postconditions. For example,

$$\{\text{Closed(Door)}, \text{Distance(Agent,Door)} < 1, \text{Holds(Agent,Axe)}\}\ \text{OpenDoor}\ \{\text{Open(Door)}\}$$

By forcing agreement on the concepts used between agents and the game engine, each can have their own internal representation while there is an agreement on what can be communicated and on what level of abstraction.

5.2.2. Communication. In a multiagent setting where agents need to coordinate their actions, they must communicate. Communication between agents can be achieved in similar fashion as actions and perception of the agent. The action of an agent now is the sending of a message, while the perception consists of the reception of a message. Sending a message consequently requires describing the pre- and postconditions. Receiving a message is controlled by an agent subscribing on messages and by the game engine when it determines that an agent can sense an action. In this case, we do not only specify the agreement between agent and game engine, but also between agents. So there are three or more parties that need to conform to the agreement instead of two in the previous case. We will call this agreement the agent communication model (ACM).

The definition of the ACM will contain the type of things that can be communicated (communication content representation) between agents. This representation is only relevant to the agents in the game. The ACM also specifies when communication can take place (qualification representation). This specifies the impact of the environment upon communication. For instance, if an agent is far away, it may not be able to communicate. These factors are relevant for the game engine that is responsible for simulating the environment.

There already exists a formalism that provides a communication content representation. It provides a way to communicate such things as beliefs among agents or propose an action or communicate with multiple agents. Within the FIPA standard [17], these communicative acts are already defined. We propose to use the FIPA standard to establish a game-specific message structure. For example, in our firefighting game, agent A may propose to agent B that A opens the door to the building:

```
(propose
        :sender (agent-identifier:name A)
        :receiver (set (agent-identifier:name B))
        :content
                "((action A (open door))"
        :ontology Fire-fighting
        :in-reply-to proposal2
        :language fipa-sl)
```

The message is translated to the concepts internal to both the agent and the game engine. The game engine will, upon reception of the above message, send this message to agent B and automatically subscribe agent A to the communication messages of agent B. This is because agent A and B can now be said to be in a dialogue and it is likely that agent A would like to receive an answer. The game could progress such that agent B replies affirmatively and the game engine receives an action from agent A to open the door and an action from agent B to go through the door. The game engine will now have enough information to know that this is a coordinated action and that the order of actions (as implied by the dialogue) is to process the door opening action first

and the movement action second. The game engine therefore takes these messages from the incoming actions queue and processes these together (coordinated) and in the right order.

To describe the impact of the environment on communication, we have to augment the linguistic representation with information about the environment. Since communication is a form of action, the same qualification representation needs to be made explicit. These qualification rules will also need to specify the ramifications of communication. This allows us on the one hand to specify what is needed when agents want to communicate (e.g., that they are close together) and on the other hand the (side) effects of communication (e.g., if other agents than the message recipient are nearby they too may receive the message):

PRE: Poss(Send(Propose(Action,Agent)))
$\iff$ Dist(Agent)<5
POST: Done(Send(Propose(Action,Agent)))
$\wedge$ Dist(Agent')<5 $\Longrightarrow$
Poss(Receive(Propose(Action,Agent)))

5.2.3. Time. Time is an important aspect in the connection between game engine and agent. Both need to agree on a reference of time. In our approach, the game engine provides the time by sending periodic time messages to all agents. The meaning of these time messages is defined in the OPM:

```
<class name="time">
  <property>
    <name>value</name>
    <Type>Seconds</type>
  </property>
</class>
```

The game engine will translate its own data in milliseconds to seconds and send the messages. The agent will translate these time messages into meaningful symbols that are relevant to the agents updating a belief it formed an hour ago to an "old belief" or "stale belief." Additional to these time messages a game designer is free to add additional facilities, such as allowing agents to query how much time passed between two events. Such a service could provide the translation from milliseconds in the game engine to concepts such as "just now," "a while," and "long ago."

The above contracts (i.e., OPM, OIM, and ACM) will be derived from the game design process in the design stance. For instance, it is established that a game interaction takes place in a scene called "building." The game designer can then start to construct the contracts that describe the concepts of that building that are relevant to both agents and the game engine.

5.3. Design Stance. In the previous sections, we discussed how agents could be connected to game engines infrastructurally and conceptually. However, creating these connections does not automatically mean that they are used in a proper way. Game design uses several methodologies [40], but all consider aspects such as rules, play, and culture. We will follow [41] and distinguish the following channels.

(i) *The abstract rules governing the game play.* For example, this determines the strength of weapons or what is needed to open doors, and so forth.

(ii) *The storyline.* This determines the overall narrative. For example, in Quake, the story is about capturing a flag.

(iii) *The user interface.* How is the game environment represented and how does the user interact with it.

(iv) *Look and feel.* What emotions are generated by the game, what kind of feeling it gives. For example, are enemies extraterrestrial beings or soldiers?

Current practice in game design assumes that the human players are intelligent. The game rules are meant to regulate how the users can interact with the game and ensure that the storyline is kept. At this moment, the only place where AI plays a significant role is on intelligent path planning. All characters have to do some form of path planning to get around in the world and this is a nice modular task that can be enhanced by some more realistic or intelligent path planning. Looking at the different channels, we see that it mainly influences the look and feel channel as it makes the characters move more natural. It thus has no fundamental influence on the game play.

This will be quite different when the characters are played by software agents that can be autonomous, adaptive, and intelligent and moreover can communicate with each other. Once these features are added, it is unclear whether the same game rules still ensure the same game play. Once characters can reason about the world, start cooperating and adapting to the players, the game might fundamentally change of character, and it is not directly clear if it will change for the better!

If we want to add software agents that can behave more intelligent and adaptive, we should also design the game rules such that the game profits from this behavior. Thus, we should take the capabilities of the characters already into account when designing the game rules! For example, a game rule that determines that in a firefighter training, at least one of three doors is locked to make the firefighting more difficult becomes useless if the characters learn how to open a locked door as quick as a nonlocked door. This becomes even more apparent if we consider that agents might also communicate (in a more or less unrestricted way). Adding communication capabilities to agents means that they can start to cooperate and thus circumvent some of the rules in the game. For example, one character can start extinguishing the fire while the other saves a victim. The one that goes inside the building to save the victim can be determined by which of the two knows the building better. This can be easily determined through communication, but is hard to preprogram. It does mean that the agents will be able to achieve more than when used independently. This kind of elements should thus also be modeled in the game rules channel. In general, one would

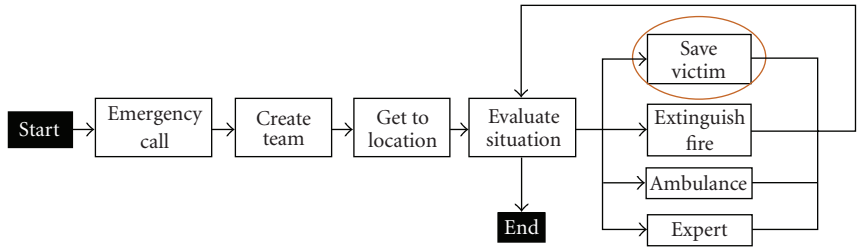


Figure 4: Interaction structure in OperA.

have to take into account who can communicate with whom and whether communication always succeeds. In a realistic environment, a character might only be able to communicate by "talking" to characters in the same physical space while other characters are overhearing the conversation.

In multiagent systems, communication mechanisms provide standard ways of dealing with these issues. However, they do not assume the agents operate in a game environment. Thus the mechanisms would have to be adjusted to the game engine.

The above points illustrate that, if we assume that characters are played by intelligent software agents that can communicate, the game rules should be designed in such a way that the storyline will still be guaranteed. Moreover, if we assume the characters to act intelligently, they should also have the means to do so, that is, they should have the right information available at the right time. For example, if a character has to avoid being seen, it does not make sense to duck behind an obstacle which happens to be an iron fence. However, if geometric features of the obstacles are the only available information, it will be hard to create intelligent behavior based on them.

This pleads for the fact that we should take possible intelligent behavior and the requirements for this behavior on the world already into account in an early design stage. One could also argue to start modeling the agents using an agent-oriented software methodology. This at least ensures a proper modeling of the agents and their interactions in the game. However, agent-oriented methodologies hardly take the environment in which the agents operate into account. Therefore, the modeling of the actual world and the intricate interactions that are needed in the game environment are not supported sufficiently.

This leaves only one way open, which is a new design methodology that allows designing the game environment and the agents concurrently. We believe that there are good starting points for creating such a methodology if we use an agent-oriented methodology that also takes the agent's organization into account.

Roughly the methodology should start with designing the game rules and storyline at a high level. At this level, the specific actions that take place are not fixed yet, but only the required landmarks that the game should pass through. In the next stage, the designer should determine which agents would possibly play a role in the different scenes that lead to these landmarks. Note that the order of scenes might still vary. He can then decide which requirements have to be fulfilled by the agents to perform their actions in the different scenes and what kind of information should they have available if they want to exhibit some intelligent behavior. Besides these requirements, he also has to give the constraints on the actions, for example, opening a door requires an axe, to ensure that "intelligent" behavior does not lead to completely unexpected and unwanted behavior.

The requirements on the availability of information lead to requirements on the conceptual contracts. The boundaries on the actions lead to requirements on the capabilities of the agents, for example, the precondition of opening a door is to carry an axe.

5.3.1. A Design Methodology: OperA. We propose to use OperA [42] as a starting point of a framework to model games incorporating agents. OperA provides a model for agent organizations that enables the specification of organizational requirements and objectives, and at the same time allows participants to act according to their own capabilities and demands. It still needs to be extended with a more elaborate environment model to capture the game world aspects. In this paper, we will focus on the specification part of the agents. Role descriptions in OperA define the activities and services that have to be performed to achieve the game objectives. These objectives are distributed over the objectives of the agents. Role descriptions also define the rights and capabilities of the agents.

By clearly defining these capabilities in an early design stage, we can guarantee that they are implemented in the game world (through a conceptual translation). Table 2 shows an example of a role description for an agent of the type "leading firefighter." From this description, the objectives of this type of agent become clear and it already gives some idea about the information needed by the agent to realize these objectives. A part of the game rules is specified by the norms. The rights of the agents also define a part of the game rules and need to be translated to actual capabilities in the game engine.

In OperA, the overall storyline is specified by the interaction structure. The main purpose of this structure is to specify an ordering between separate scenes in the game and to make sure that required states are always reached. The ordering is not always linear; scenes can be executed multiple times. The actors that participate in the scenes of

Table 2: Role definition in OperA.

Role: leading firefighter	
Objectives	Fire_under_control, victims_save
Subobjectives	{get_to_disaster_location, situation_assessment, plan_of_attack, extinguish_fire, rescue_victims}
Rights	Command_team_members, order_ambulance, get_experts
Norms	OBLIGED inform(headquarters, plan_of_attack) BEFORE NOW+10 IF DO safe(victim) or DO extinguish(fire) THEN PERMITTED damage(building) OBLIGED ensure_safety(team) OBLIGED safe(victims) BEFORE extinguish(fire)

Table 3: Interaction scene in OperA.

Interaction scene: save victim	
Roles	Leading_firefighter(1), door_opener(1), fire_extinguisher(1), ambulance(2), victim(3)
Results	r1 = ∀ T ∈ victim, safe(T)
Interaction patterns	PATTERN(r1) = {DONE(T, at(H,T)) BEFORE DONE(B, secure_area), DONE(B, secure_area) BEFORE DeadlineH, DONE(M, stabilise(H) BEFORE Dead(H)) DONE(T, transport_to_ambulance(H)) }
Norms	PERMITTED (E, blow_obstacles) OBLIGED (M,stabilise(T) BEFORE Dead(T)) OBLIGED (B, extinguish_fire BEFORE transport(H))

the game and the way they interact are defined in the scene level description. Figure 4 shows a graphical representation of a possible interaction structure. The transitions between the different scenes are specified in the interaction structure to make sure that a scene is entered and terminated in such a way that the storyline is guaranteed.

A scene is a formal description of the interaction space between different agents for a specific part of the game. In these scenes, the types and number of participating agents are defined, and the interaction between the agents themselves and the environment. The result of a scene is specified and optionally norms can be added.

Table 3 shows a possible description of the "save victim" scene from the interaction structure above. For each role that is possibly active in this scene, we specify the number of agents that fulfill that role. For example, there is one leadingfirefighter and there are three victims. Most importantly, the desired results of the scene and the interaction patterns between the different roles are specified. In an interaction scene, separate norms and permissions can be specified that determine specific game rules for the interaction.

Starting from the interaction patterns of the scenes, different messages and other forms of communication can be specified, and a platform-independent design can be created. The ordering of these communicative actions can be strictly defined by a protocol or, more flexible, an interaction diagram. From this platform-independent model, we can move on to the platform-specific phase, in which the agent interface and the interaction specifications are implemented. If certain inherent limitations on the communication are known in advance, these limitations should already be taken into account during the platform-independent design phase. If they surface during the implementation phase, it is usually better to go back and adjust the platform-independent design. In the design phase, the agents' knowledge about the environment and themselves should be modeled. This information can later be used in the platform-specific design to create the data models.

After we specified the agent roles and interactions, decisions have to be made about the agent implementation. The requirements that the agents need to fulfill have to be taken into account in this step. For example, if the agents have to be able to explain themselves [43], it is necessary that they use high-level concepts in their reasoning, such as beliefs, intensions, and goals. A logical decision in case of this requirement would be to implement the agents in a BDI programing language. Another example could be learning or adapting agents; the appropriate agent type needs to be selected to allow for the expected adaptability. Also the learning algorithm itself, the elements that are adapted and the feedback type have to be chosen.

The multiagent interaction also has to be specified more precisely. In the interaction scenes, we already define a high-level definition of the interaction. As we have seen in Section 3.2 (multiagents systems), certain tradeoffs have to be made on the amount of communication. In the design phase, a clear definition has to be made of what information is passed on and when. Designing decisions also have to be made about the activity of characters that are not playing an active role in the current scene.

Besides specifying the technical requirements, some quality requirements have to be kept into consideration as well. An important quality requirement for computers is that the agents and other parts of game respond fast enough. The behavior of agents should be believable. Games should be esthetically pleasing and a certain atmosphere in the design is desired. These quality requirements are mostly related to the look and feel channel and the user interface channel. Sometimes they can be translated to a technical specification, but most of the time they have to be considered during the whole development process without being captured by a precise technical requirement.

6. Conclusion

There is consensus among game developers that intelligent characters for games can make games better. However, there is a difference in the approach to bring intelligence about between the game developers and the artificial intelligences researchers. Consequently, using agent technology in combination with game technology is not trivial. Because agents are more or less autonomous they should run in their own thread and can only be loosely coupled to the game engine. Synchronizing the agents with the game thus becomes an important point. Once the agents are synchronized not all problems are solved. Because agents usually function on a more abstract level than the game world representation allows. A translation is needed between the game world information and processes to the beliefs and actions of the agents. Finally, agents should be able to communicate not only with the game world but also with each other. Thus there is a need for communication mechanisms that connect both the agents and the game world. We have seen that current combinations of games and agents only deliver limited or ad hoc solutions for all these issues.

In this paper, we argue that improving the AI in games by using agent technology to its full extent involves solving the issues above. Furthermore, solving the synchronization, information representation, and communication issues requires more than constructing a technical solution for the loosely coupling of some asynchronous processes. Although this aspect is a fundamental part of the coupling, we also need to provide support on a conceptual and design level. Using a conceptual stance allows for connecting the agent concepts to the game concepts such that agent actions can be connected to actions that can be executed through the game engine and that agents can reason intelligently on the information available from the game engine.

We also argue that coupling agents to games requires a design methodology including agent notions from an early stage in the design process in order to allow a full integration of agent characteristics in the game and to profit from specific agent characteristics such as communication, cooperation, reasoning, proactive behavior, and adaptivity.

In Section 5, we have shown how each of the three stances can contribute to the use of agents in games. We have also shown some standards and tools that could be used in each of the three stances. We have successfully tested the synchronization principles explained in the infrastructural stance by coupling the Pilgrim game engine (under development at TNO, Soesterberg, The Netherlands) with the 2APL agent platform.

We have shown that the HLA standard is a good starting point to describe the filtering in the conceptual stance. The ease of the translation between the game engine and the agent concepts, of course, also depends on the specific platforms used. The Pilgrim game engine appeared very suitable for this approach because all game objects have a property tree describing all the properties of that object, thus allowing for an easy translation to a common representation (OPM). In a similar fashion, the properties of the 2APL agents are available in a declarative format and could easily be converted to the common representation (OIM). Finally, the agent-based methodology OperA seems to offer a good starting point for combining agent-oriented and game-oriented methodologies.

As a future work, we hope to build some support tools to facilitate the modeling and implementation of games with agents, making use of the framework sketched in this paper.

Acknowledgments

Thanks to Bernard Maassen whose implementation of the connection between Pilgrim and 2APL started off this paper. This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

References

[1] S. Rabin, *AI Game Programming Wisdom 3*, Charles River Media, Brookline, Mass, USA, 2006.

[2] A. Ayesh, J. Stokes, and R. Edwards, "Fuzzy Individual Model (FIM) for realistic crowd simulation: preliminary results," in *Proceedings of IEEE International Conference on Fuzzy Systems (FUZZ '07)*, pp. 1–5, London, UK, July 2007.

[3] J. Orkin, "Three states and a plan: the AI of F.E.A.R.," in *Proceedings of the Game Developers Conference (GDC '06)*, San Jose, Calif, USA, March 2006.

[4] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[5] J. Orkin, "Applying goal-oriented action planning to games," in *AI Game Programming Wisdom 2*, Charles River Media, Brookline, Mass, USA, 2003.

[6] M. E. Pollack and J. F. Horty, "There's more to life than making plans: plan management in dynamic, multiagent environments," *AI Magazine*, vol. 20, no. 4, pp. 71–83, 1999.

[7] M. Lees, B. Logan, and G. K. Theodoropoulos, "Agents, games and HLA," *Simulation Modelling Practice and Theory*, vol. 14, no. 6, pp. 752–767, 2006.

[8] R. Adobbati, A. N. Marshall, A. Scholer, et al., "Gamebots: a 3D virtual world test-bed for multi-agent research," in *Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montreal, Canada, May 2001.

[9] R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Eds., *Multi-Agent Programming: Languages, Platforms and Applications*, International Book Series on Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer, Berlin, Germany, 2005.

[10] A. Newell, *Unified Theories of Cognition*, Harvard University Press, Cambridge, Mass, USA, 1994.

[11] S. Franklin and L. Gasser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *Intelligent Agents III. Agent Theories, Architectures, and Languages*, pp. 21–35, Springer, Berlin, Germany, 1997.

[12] "Bos Wars," http://www.boswars.org.

[13] S. Kopp, L. Gesellensetter, N. C. Krämer, and I. Wachsmuth, "A conversational agent as museum guide—design and evaluation of a real-world application," in *Proceedings of the 5th International Working Conference on Intelligent Virtual Agents (IVA '05)*, T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, and T. Rist, Eds., vol. 3661 of *Lecture Notes in Computer Science*, pp. 329–343, Springer, Kos, Greece, September 2005.

[14] L. Antunes and K. Takadama, Eds., *Multi-Agent-Based Simulation VII*, vol. 4442 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2007.

[15] B. G. Silverman, G. Bharathy, M. Johns, R. J. Eidelson, T. E. Smith, and B. Nye, "Sociocultural games for training and analysis," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 37, no. 6, pp. 1113–1130, 2007.

[16] L. Padgham, D. Parkes, S. Parsons, and J. Müller, Eds., *Proceedings of the 7th International Conference on Autonomous Agents and Multi Agent Systems (AAMAS '08)*, IFAAMAS, Estoril, Portugal, May 2008.

[17] FIPA, Foundation for Intelligent Physical Agents, http://www.fipa.org.

[18] M. Wooldridge, *Reasoning about Rational Agents*, MIT Press, Cambridge, Mass, USA, 2000.

[19] M. Dastani, "2APL: a practical agent programming language," *Autonomous Agents and Multi-Agent Systems*, vol. 16, no. 3, pp. 214–248, 2008.

[20] R. Bordini, J. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*, John Wiley & Sons, New York, NY, USA, 2007.

[21] P. S. Rosenbloom, J. E. Laird, and A. Newell, *The Soar Papers: Readings on Integrated Intelligence*, MIT Press, Cambridge, Mass, USA, 1993.

[22] J. R. Anderson, "ACT: a simple theory of complex cognition," *American Psychologist*, vol. 51, no. 4, pp. 355–365, 1996.

[23] Quake, http://www.idsoftware.com/games/quake/quake3-arena.

[24] Never Winter Nights, http://nwn.bioware.com.

[25] F.E.A.R., http://whatisfear.com.

[26] G. A. Kaminka, M. M. Veloso, S. Schaffer, et al., "GameBots: a flexible test bed for multiagent team research," *Communications of the ACM*, vol. 45, no. 1, pp. 43–45, 2002.

[27] A. Khoo, G. Dunham, N. Trienens, and S. Sood, "Efficient, realistic NPC control systems using behavior-based techniques," in *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Palo Alto, Calif, USA, March 2002.

[28] J. M. P. van Waveren, *The quake III arena bot*, M.S. thesis, Faculty ITS, University of Technology Delft, Delft, The Netherlands, 2003.

[29] A. Witzel and J. Zvesper, "Higher order knowledge in computer games," in *Proceedings of the AISB Symposium on Logic and the Simulation of Interaction and Reasoning*, pp. 1–5, Aberdeen, Scotland, April 2008.

[30] S. Kraus, "Negotiation and cooperation in multi-agent environments," *Artificial Intelligence*, vol. 94, no. 1-2, pp. 79–97, 1997.

[31] T. Mioch, M. Harbers, W. van Doesburg, and K. van den Bosch, "Enhancing human understanding through intelligent explanations," in *Proceedings of the 1st International Workshop on Human Aspects in Ambient Intelligence*, pp. 327–337, Darmstadt, Germany, November 2007.

[32] W. L. Johnson, "Agents that learn to explain themselves," in *Proceedings of the 12th National Conference on Artificial Intelligence*, vol. 2, pp. 1257–1263, Seattle, Wash, USA, July-August 1994.

[33] M. van Lent, W. Fisher, and M. Mancuso, "An explainable artificial intelligence system for small-unit tactical behavior," in *Proceedings of the 19th National Conference on Artificial Intelligence and the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI '04)*, pp. 900–907, AAAI Press, San Jose, Calif, USA, July 2004.

[34] D. Gomboc, S. Solomon, M. G. Core, H. C. Lane, and M. van Lent, "Design recommendations to support automated explanation and tutoring," in *Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation (BRIMS '05)*, Universal City, Calif, USA, May 2005.

[35] E. Norling and L. Sonenberg, "Creating interactive characters with BDI agents," in *Proceedings of the Australian Workshop on Interactive Entertainment (IE '04)*, pp. 69–76, Sydney, Australia, February 2004.

[36] R. Descartes, *Treatise on Man*, Harvard University Press, Cambridge, Mass, USA, 1976.

[37] IEEE Std 1516-2000, "IEEE Standard for modeling and simulation (M&S) high level architecture (HLA)—framework and rules," September 2000.

[38] Kynogon, http://www.kynogon.com.

[39] R. Reiter, *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*, MIT Press, Cambridge, Mass, USA, 2001.

[40] P. Vorderer and J. Bryant, *Playing Video Games*, Lawrence Erlbaum, Mahwah, NJ, USA, 2006.

[41] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*, MIT Press, Cambridge, Mass, USA, 2004.

[42] V. Dignum, *A model for organizational interaction: based on agents, founded in logic*, Ph.D. dissertation, Utrecht University, Utrecht, The Netherlands, 2004.

[43] C. de Melo, R. Prada, G. Raimundo, J. P. Pardal, H. S. Pinto, and A. Paiva, "Mainstream games in the multi-agent classroom," in *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT '06)*, pp. 757–761, Hong Kong, December 2006.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 910819, 10 pages
doi:10.1155/2009/910819

Research Article

A Multiagent Potential Field-Based Bot for Real-Time Strategy Games

Johan Hagelbäck and Stefan J. Johansson

Department of Software and Systems Engineering, Blekinge Institute of Technology, P.O. Box 520, 372 25 Ronneby, Sweden

Correspondence should be addressed to Johan Hagelbäck, jhg@bth.se

Received 30 April 2008; Accepted 7 September 2008

Recommended by Abdennour El Rhalibi

Bots for real-time strategy (RTS) games may be very challenging to implement. A bot controls a number of units that will have to navigate in a partially unknown environment, while at the same time avoid each other, search for enemies, and coordinate attacks to fight them down. Potential fields are a technique originating from the area of robotics where it is used in controlling the navigation of robots in dynamic environments. Although attempts have been made to transfer the technology to the gaming sector, assumed problems with efficiency and high costs for implementation have made the industry reluctant to adopt it. We present a multiagent potential field-based bot architecture that is evaluated in two different real-time strategy game settings and compare them, both in terms of performance, and in terms of softer attributes such as configurability with other state-of-the-art solutions. We show that the solution is a highly configurable bot that can match the performance standards of traditional RTS bots. Furthermore, we show that our approach deals with Fog of War (imperfect information about the opponent units) surprisingly well. We also show that a multiagent potential field-based bot is highly competitive in a resource gathering scenario.

1. Introduction

A *real-time strategy* (RTS) game is a game in which the players use resource gathering, base building, technological development and unit control in order to defeat its opponent(s), typically in some kind of war setting. The RTS game is not turn-based in contrast to board games such as Risk and Diplomacy. Instead, all decisions by all players have to be made in real time. Generally the player has a top-down perspective on the battlefield although some 3D RTS games allow different camera angles. The real-time aspect makes the RTS genre suitable for multiplayer games since it allows players to interact with the game independently of each other and does not let them wait for someone else to finish a turn.

In RTS games computer bots often "cheats," that is, they have complete visibility (perfect information) of the whole game world. The purpose is to have as much information available as possible for the artificial intillegence (AI) to reason about tactics and strategies in a certain environment. Cheating is, according to Nareyek, "*very annoying for the player if discovered*" and he predicts the game AIs to get a larger share of the processing power in the future which in turn may open up for the possibility to use more sophisticated AIs [1]. The human player in most modern RTS games does not have this luxury, instead the player only has visibility of the area populated by the own units, and the rest of the game world is unknown until it gets explored. This property of incomplete information is usually referred to as *Fog of War* or FoW.

In 1985, Ossama Khatib introduced a new concept while he was looking for a real-time obstacle avoidance approach for manipulators and mobile robots. The technique which he called *Artificial Potential Fields* moves a manipulator in a field of forces. The position to be reached is an attractive pole for the end effector (e.g., a robot) and obstacles are repulsive surfaces for the manipulator parts [2]. Later on Arkin [3] updated the knowledge by creating another technique using superposition of spatial vector fields in order to generate behaviors in his so called motor schema concept.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance (see, e.g., [4, 5]). The technique is really helpful for the avoidance of simple

obstacles even though they are numerous. Combined with an autonomous navigation approach, the result is even better, being able to surpass highly complicated obstacles [6].

Lately some other interesting applications for potential fields have been presented. The use of potential fields in architectures of multi agent systems is giving quite good results defining the way of how the agents interact. Howard et al. developed a mobile sensor network deployment using potential fields [7], and potential fields have been used in robot soccer [8, 9]. Thurau et al. [10] have developed a game bot which learns reactive behaviours (or potential fields) for actions in the first-Person Shooter game Quake II through imitation.

The article is organised as follows. First, we propose a methodology for multiagent potential field- (MAPFs-) based solution in an RTS game environment. We will show how the methodology can be used to create a bot for a resource gathering scenario (Section 4) followed by a more complex tankbattle scenario in Section 5. We will also present some preliminary results on how to deal with imperfect information, Fog of War (Section 6). The methodology has been presented in our previous papers [11, 12]. This article summarises the previous work and extends it by adding new experiments and new results. Last in this article, we have a discussion and line out some directions for future work.

2. A Methodology for Multiagent Potential Fields

When constructing a multiagent potential field-based system for controlling agents in a certain domain, there are a number of issues that we must take into consideration. It is, for example, important that each interesting object in the game world generates some type of field, and we must decide which objects can use static fields to decrease computation time.

To structure this, we identify six phases in the design of an MAPF-based solution:

(1) the identification of objects;

(2) the identification of the driving forces (i.e., the fields) of the game;

(3) the process of assigning charges to the objects;

(4) the granularity of time and space in the environment;

(5) the agents of the system;

(6) the architecture of the MAS.

In *the first phase*, we may ask us the following questions. What are the *static objects* of the environment? That is, what objects keep their attributes throughout the lifetime of the scenario? What are the *dynamic objects* of the environment? Here we may identify a number of different ways that objects may change. They may move around, if the environment has a notion of physical space. They may change their attractive (or repulsive) impact on the agents. What is the *modifiability* of the objects? Some objects may be consumed, created, or changed by the agents.

In *the second phase*, we identify the driving forces of the game at a rather abstract level, for example, to avoid obstacles, or to base the movements on what the opponent does. This leads us to a number of fields. The main reason to enable multiple fields is that it is very easy to isolate certain aspects of the computation of the potentials if we are able to filter out a certain aspect of the overall potential, for example, the repulsive forces generated by the terrain in a physical environment. We may also dynamically weight fields separately, for example, in order to decrease the importance of the navigation field when a robot stands still in a surveillance mission (and only moves its camera). We may also have *strategic fields* telling the agents in what direction their next goal is, or *tactical fields* coordinating the movements with those of the teammate agents.

The third phase includes placing the objects in the different fields. Static objects should typically be in the *field of navigation*. The potentials of such a field are precalculated in order to save precious run time CPU resources.

In *the fourth phase*, we have to decide the resolution of space and time. Resolution of space means how detailed the navigation in the game world should be. Should for example the agents be able to move to every single point in the world, or should the game world be divided into a grid with tiles of for example 4×4 points in the world? Resolution of time means how often the potential fields should be updated. If the agents are able to move around in the environment, both these measures have an impact on the lookahead. The space resolution obviously, since it decides what points in space that we are able to access, and the time in that it determines how far we may get in one time frame (before it is time to make the next decision about what to do).

The fifth phase is to decide what objects to agentify and set the repertoire of those agents: what actions are we going to evaluate in the lookahead? As an example, if the agent is omnidirectional in its movements, we may not want to evaluate all possible points that the agent may move to, but rather try to filter out the most promising ones by using some heuristic, or use some representable sample.

In *the sixth step*, we design the architecture of the MAS. Here we take the unit agents identified in the fifth phase, give them roles, and add the supplementary agents (possibly) needed for coordination, and special missions (not covered by the unit agents themselves).

3. ORTS

Open real-time strategy (ORTS) [13] is a real-time strategy game engine developed as a tool for researchers within artificial intelligence (AI) in general and game AI in particular. ORTS uses a client-server architecture with a game server and players connected as clients. Each timeframe clients receives a data structure from the server containing the current game state. Clients can then call commands that activate and control their units. Commands can be like "move unit A to (x, y) or attack opponent unit X with unit A." The game server executes the client commands in random order.

Users can define different types of games in scripts where units, structures, and their interactions are described. All

types of games from resource gathering to full real-time strategy (RTS) games are supported.

We will begin by looking at a one-player resource gathering scenario game called *Collaborative Pathfinding*, which was part of the 2007 and 2008 ORTS competitions [13]. In this game, the player has 20 worker units. The goal is to use the workers to mine resources from nearby mineral patches and return them to a base. A worker must be adjacent to a mineral object to mine, and to a base to return resources. As many resources as possible will be collected within 10 minutes.

This is followed by looking at the two-player games, *Tankbattle*, which was part of the 2007 and 2008 ORTS competitions [13] as well.

In *Tankbattle,* each player has 50 tanks and five bases. The goal is to destroy the bases of the opponent. Tanks are heavy units with long fire range and devastating firepower but a long cool-down period, that is, the time after an attack before the unit is ready to attack again. Bases can take a lot of damage before they are destroyed, but they have no defence mechanism of their own so it may be important to defend our own bases with tanks. The map in a tankbattle game has randomly generated terrain with passable lowland and impassable cliffs.

Both games contain a number of neutral units (sheep). These are small indestructible units moving randomly around the map. The purpose of sheep is to make pathfinding and collision detection more complex.

4. Multiagent Potential Fields in ORTS

First we will describe a bot playing the Collaborative Pathfinding game based on MAPF following the proposed methodology. Collaborative Pathfinding is a 1-player game where the player has one control center and 20 worker units. The aim is to move workers to mineral patches, mine up to 10 resources (the maximum load a worker can carry), then return to a friendly control center to drop them off.

4.1. Identifying Objects. We identify the following objects in our application: *Cliffs, Sheep, Base stations,* and *workers.*

4.2. Identifying Fields. We identified five tasks in ORTS: avoid colliding with the terrain, avoid getting stuck at other moving objects, avoid colliding with the bases, move to the bases to leave resources, and move to the mineral patches to get new resources. This leads us to three major types of potential fields: a *field of navigation*, a *strategic field,* and a *tactical field*.

The field of navigation is a field generated by repelling static terrain. This is because we would like the agents to avoid getting too close to objects where they may get stuck, but instead smoothly pass around them.

The strategic field is a dynamic attracting field. It makes agents go towards the mineral patches to mine, and return to the base to drop off resources.

Own workers, bases, and sheep generate small repelling fields. The purpose of these fields is the same as for obstacle avoidance; we would like our agents to avoid colliding with each other and bases as well as avoiding the sheep. This task is managed by the tactical field.

4.3. Assigning Charges. Each worker, base, sheep, and cliffs has a set of charges which generates a potential field around the object. These fields are weighted and summed together to form a total potential field that is used by our agents for navigation.

Cliffs, for example, impassable terrain, generate a repelling field for obstacle avoidance. The field is constructed by copying pregenerated matrixes of potentials into the field of navigation when a new game is started. The potential all cliffs generate in a point (x, y) is calculated as the lowest potential a cliff generates in that point. The potential $p_{\text{cliff}}(d)$ in a point at distance d from the closest impassable terrain tile is calculated as:

$$p_{\text{cliff}}(d) = \begin{cases} \dfrac{-80}{(d/8)^2} & \text{if } d > 0, \\ -80 & \text{if } d = 0. \end{cases} \tag{1}$$

Own worker units generate repelling fields for obstacle avoidance. The potential $p_{\text{worker}}(d)$ at distance d from the center of another worker is calculated as

$$p_{\text{worker}}(d) = 1.429 \cdot \begin{cases} -20 & \text{if } d \leq 6, \\ 16 - 2 \cdot d & \text{if } d \in]6, 8]. \end{cases} \tag{2}$$

Sheep. Sheep generate a small repelling field for obstacle avoidance. The potential $p_{\text{sheep}}(d)$ at distance d from the center of a sheep is calculated as

$$p_{\text{sheep}}(d) = 0.125 \cdot \begin{cases} -20 & \text{if } d \leq 8, \\ 2 \cdot d - 25 & \text{if } d \in]8, 12.5]. \end{cases} \tag{3}$$

Own bases. The own bases generate two different fields depending on the current state of a worker. The base generates an attractive field if the worker needs to move to the base and drop off its resources. Once it has arrived at the base, all the resources are dropped. The potential $p_{\text{attractive}}(d)$ at distance d from the center of the base is calculated as

$$p_{\text{attractive}}(d) = \begin{cases} 240 - d \cdot 0.32 & \text{if } d \leq 750, \\ 0 & \text{if } d > 750. \end{cases} \tag{4}$$

In all other states of the worker, the own base generates a repelling field for obstacle avoidance. Below is the function for calculating the potential $p_{\text{own }B}(d)$ at distance d from the center of the base. Note that this is, of course, the view of the worker. The base will effect some of the workers with the attracting field while at the same time effect the rest with a repelling field. If a point is inside the quadratic area the base occupies, the potential in those points is always 10000 (potential used for impassable points):

$$p_{\text{own }B}(d) = 0.125 \cdot \begin{cases} 6 \cdot d - 258 & \text{if } d \leq 43, \\ 0 & \text{if } d > 43. \end{cases} \tag{5}$$

Minerals, similar to own bases, generate attractive fields for all workers that do not carry maximum loads and a

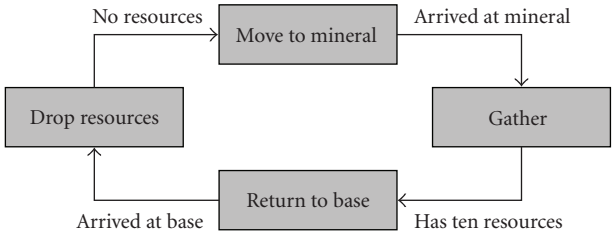


Figure 1: The finite state machine used by the workers in a resource gathering scenario.

repelling field for obstacle avoidance when they do. The potential of the attractive field is the same as the attractive field around the own base in (4).

In the case when minerals generate a repelling field, the potential $p_{\text{mineral}}(d)$ at distance d from the center of a mineral is calculated as

$$p_{\text{mineral}}(d) = 1.429 \cdot \begin{cases} -20 & \text{if } d \leq 8, \\ 20 - 2 \cdot d & \text{if } d \in]8, 10]. \end{cases} \quad (6)$$

4.4. The Granularity of the System. Since the application is rather simple, we use full resolution of both the map and the time frames without any problems.

4.5. The Agents. The main units of our system are the workers. They use a simple finite state machine (FSM) illustrated in Figure 1 to decide what state they are in (and thus what fields to activate). No central control or explicit coordination is needed, since the coordination is emerging through the use of the charges.

4.6. The Multiagent System Architecture. In addition to the worker agents, we have one additional agent that is the interface between the workers and the game server. It receives server information about the positions of all objects and workers which it distributes to the worker agents. They then decide what to do, and submit their proposed actions to the interface agent which in turn sends them through to the ORTS server.

4.7. Experiments, Resource Gathering. Table 1 shows the result from the Collaborative Pathfinding game in 2008 years' ORTS tournament. It shows that an MAPF-based bot can compete with A*-based solutions in a resource gathering scenario. There are however some uncertainties in these results. Our bot has disconnected from the server (i.e., crashed) in 30 games. The reason for this is not yet clear and must be investigated in more detail. Another issue is that Uofa has used the same bot that they used in the 2007 years' tournament, and the bot had a lower score this year. The reason, according to the authors, was "probably caused by a pathfinding bug we introduced" [14]. Still we believe that with some more tuning and bug fixing our bot can probably match the best bots in this scenario.

Table 1: Experiment results from the Collaborative Pathfinding game in 2008 years' tournament.

Team	Matches	Avg. Resources	Disconnected
BTH	250	5630.72	30
Uofa	250	4839.6	0

5. MAPF in ORTS, Tankbattle

In the 2-player Tankbattle game, each player has a number of tanks and bases, and the goal is to destroy the opponent bases. In [11] we describe the implementation of an ORTS bot playing Tankbattle based on MAPF following the proposed methodology. This bot was further improved in [12] where a number of weaknesses of the original bot were addressed. We will now, just as in the case of the Collaborative pathfinding scenario, present the six steps of the used methodology. However, there are details in the implementation of several of these steps that we have improved and shown the effect of in experiments. We will therefore, to improve the flow of the presentation, not present all of them in chronologic order. Instead we start by presenting the ones that we have kept untouched through the series of experiments.

5.1. Identifying Objects. We identify the following objects in our application: Cliffs, Sheep, and own (and opponent) tanks and base stations.

5.2. Identifying Fields. We identified four tasks in ORTS: *Avoid colliding with the terrain, Avoid getting stuck at other moving objects, Hunt down the enemy's forces,* and *Defend the bases*. In the resource gathering scenario we used the two major types: *field of navigation* and *strategic field*. Here we add a new major type of potential field: the *defensive field*.

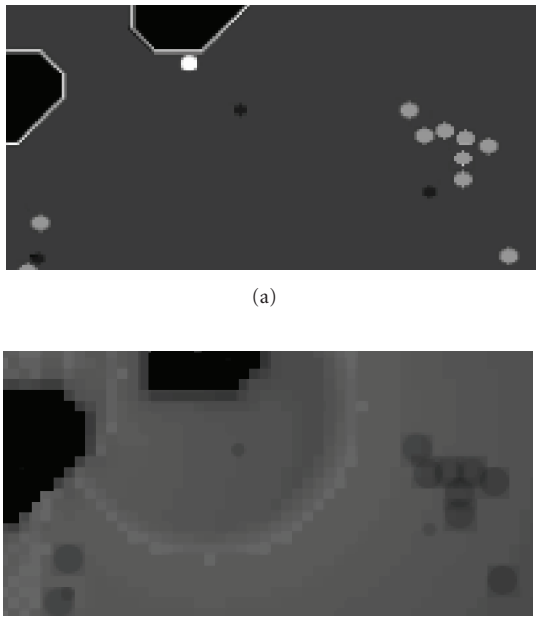
The field of navigation is, as in the previous example of Collaborative pathfinding, a field generated by repelling static terrain for obstacle avoidance.

The strategic field is an attracting field. It makes units go towards the opponents and place themselves on appropriate distances where they can fight the enemies.

The defensive field is a repelling field. The purpose is to make own agents retreat from enemy tanks when they are in cooldown phase. After an agent has attacked an enemy unit or base, it has a cooldown period when it cannot attack and it is therefore a good idea to stay outside enemy fire range while being in this phase. The defensive field is an improvement to deal with a weakness found in the original bot [11].

Own units, own bases, and sheep generate small repelling fields. The purpose is the same as for obstacle avoidance; we would like our agents to avoid colliding with each other or bases as well as avoiding the sheep. This is managed by the tactical field.

5.3. Assigning Charges. The upper picture in Figure 2 shows part of the map during a tankbattle game. The screenshots are from the 2D GUI available in the ORTS server. It shows our agents (light-grey circles) moving in to attack an



(a)

(b)

Figure 2: Part of the map during a tankbattle game. The upper picture shows our agents (light-grey circles), an opponent unit (white circle), and three sheep (small dark-grey circles). The lower picture shows the total potential field for the same area. Light areas have high potential and dark areas have low potential.

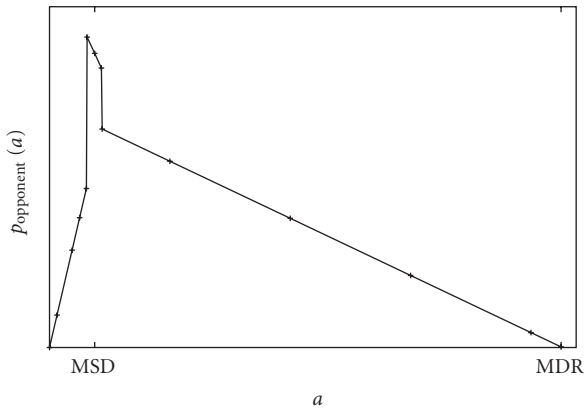


Figure 3: The potential $p_{\text{opponent}}(a)$ generated by opponent units as a function of the distance a.

opponent unit (white circle). The area also has some cliffs (black areas) and three sheep (small dark-grey circles). The lower picture shows the total potential field in the same area. Dark areas have low potential and light areas have high potential. The light ring around the opponent unit, located at maximum shooting distance of our tanks, is the distance from which our agents prefer to attack opponent units. The picture also shows the small repelling fields generated by our own agents and the sheep.

Cliffs. Cliffs generate the same field as in the resource gathering scenario, see Section 4.3.

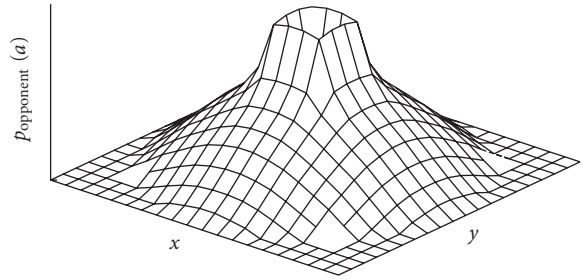


Figure 4: The potential $p_{\text{opponent}}(a)$ generated by the opponent that is in the middle.

The Opponent Units and Bases. All opponent units and bases generate symmetric surrounding fields where the highest potential is in a ring around the object with a radius of *maximum shooting distance* (MSD). MDR refers to the *Maximum Detection Range*, the distance from which an agent starts to detect the opponent unit. The reason why the location of the enemy unit is not the final goal is that we would like our units to surround the enemy units by attacking from the largest possible distance. The potential all opponent units generate in a certain point is then equal to the highest potential *any* opponent unit generates in that point, and not the sum of the potentials that all opponent units generate. If we were to sum the potentials, the highest potential and most attractive destination would be in the center of the opponent unit cluster. This was the case in the first version of our bot and was identified as one of its major weaknesses [11]. The potentials $p_{\text{opp}U}(d)$ and $p_{\text{opp}B}(d)$ at distance d from the center of an agent and with $D = \text{MSD}$ and $R = \text{MDR}$ are calculated as

$$p_{\text{opp}U}(d) = 0.125 \cdot \begin{cases} 240/d(D-2) & \text{if } d \in [0, D-2[, \\ 240 & \text{if } d \in [D-2, D], \\ 240 - 0.24(d-D) & \text{if } d \in]D, R], \end{cases}$$

$$p_{\text{opp}B}(d) = 0.125 \cdot \begin{cases} 360/d(D-2) & \text{if } d \in [0, D-2[, \\ 360 & \text{if } d \in [D-2, D], \\ 360 - 0.32(d-D) & \text{if } d \in]D, R]. \end{cases} \tag{7}$$

$I = [a, b[$ denote the half-open interval, where $a \in I$, but $b \notin I$.

Own units generate repelling fields for obstacle avoidance. The potential $p_{\text{own}U}(d)$ at distance d from the center of a unit is calculated as:

$$p_{\text{own}U}(d) = 0.125 \cdot \begin{cases} -20 & \text{if } d \le 14 \\ 32 - 2 \cdot d & \text{if } d \in]14, 16] \end{cases} \tag{8}$$

Own bases generate repelling fields similar to the fields around the own bases described in Section 4.3.

Sheep generate the same weak repelling fields as in the Collaborative pathfinding scenario, see Section 4.3.

5.4. The Multiagent Architecture. In addition to the interface agent dealing with the server (which is more or less the

Table 2: Experiment results from the original bot.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	0%	(0/100)	0.01	0.00	−46.99
WarsawB	0%	(0/100)	1.05	0.01	−42.56
UBC	24%	(24/100)	4.66	0.92	−17.41
Uofa.06	32%	(32/100)	4.20	1.45	−16.34
Average	14%	(14/100)	2.48	0.60	−30.83

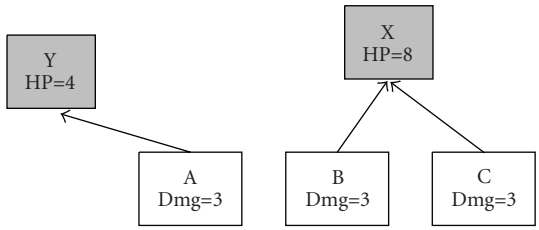


Figure 5: Attacking most damaged unit within firerange.

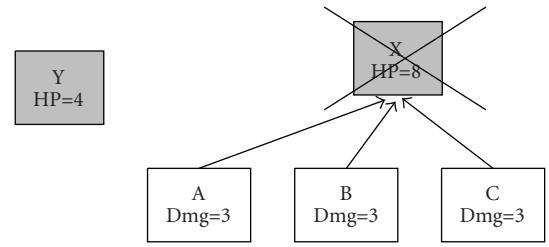


Figure 6: Optimise attacks to destroy as many units as possible.

same as in the collaborative pathfinding scenario), we use a coordinator agent to globally coordinate the attacks on opponent units to maximise the number of opponent units destroyed. The difference between using the coordinator agent compared to attacking the most damaged unit within fire range is best illustrated with an example.

In Figure 5, the own units A, B, and C does 3 damage to opponent units. They can attack opponent unit X (can take 8 more damage before it is destroyed) and unit Y (can take 4 more damage before it is destroyed). Only unit A can attack enemy unit Y. The most common approach in the ORTS tournament [13] was to attack the most damaged enemy unit within firerange. In the example both enemy unit X and Y would be attacked, but both would survive to answer the attacks.

With the coordinator agent attacks would be spread out as in Figure 6. In this case enemy unit X would be destroyed and only unit Y can answer the attacks.

5.5. The Granularity of the System. Each unit (own or enemy), base, sheep, and cliffs has a set of charges which generates a potential field around the object. These fields are weighted and summed together to form a total potential field that is used by our agents for navigation.

In [11] we used pregenerated fields that were simply added to the total potential field at runtime. To reduce memory and CPU resources needed, the game world was split into tiles where each tile was 8×8 points in the game world. This proved not to be detailed enough and our agents often got stuck in terrain and other game objects. The results as shown in Table 2 are not very impressive and our bot only won 14% of the played games.

Some notes on how the results are presented:

(i) *Avg units.* This is the average number of units (tanks) our bot had left after a game is finished.

(ii) *Avg bases.* This is the average number of bases our bot had left after a game is finished.

(iii) *Avg score.* This is the average score for our bot after a game is finished. The score is calculated as

$$\begin{aligned} \text{score} = {} & 5(\text{ownBasesLeft} - \text{oppBasesLeft}) \\ & + \text{ownUnitsLeft} - \text{oppUnitsLeft}. \end{aligned} \tag{9}$$

In [12] we proposed a solution to this problem. Instead of dividing the game world into tiles, the resolution of the potential fields was set to 1×1 points. This allows navigation at the most detailed level. To make this computationally feasible, we calculate the potentials at runtime, but only for those points that are near own units that are candidates to move to in the next time frame. In total, we calculate nine potentials per unit, eight directions, and the potential of staying in the position it is. The results, as shown in Table 3, show a slight increase in the number of games won and a large improvement in the game score.

5.6. Adding an Additional Field. Defensive Field. After a unit has fired its weapon, the unit has a cooldown period when it cannot attack. In the original bot our agents were, as long as there were enemies within maximum shooting distance (MSD), stationary until they were ready to fire again. The cooldown period can instead be used for something more useful and in [12] we proposed the use of a defensive field. This field makes the units retreat when they cannot attack and advance when they are ready to attack once again. With this enhancement, our agents always aim to be at MSD of the closest opponent unit or base and surround the opponent unit cluster at MSD. The potential $p_{\text{defensive}}(d)$ at distance d from the center of an agent is calculated using the formula in

$$p_{\text{defensive}}(d) = \begin{cases} w_2 \cdot (-800 + 6.4 \cdot d) & \text{if } d \leq 125, \\ 0 & \text{if } d > 125. \end{cases} \tag{10}$$

Table 3: Experiment results from increased granularity.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	9%	(9/100)	1.18	0.57	−32.89
WarsawB	0%	(0/100)	3.03	0.12	−36.71
UBC	24%	(24/100)	16.11	0.94	0.46
Uofa.06	42%	(42/100)	10.86	2.74	0.30
Average	18.75%	(18.75/100)	7.80	1.09	−17.21

Table 4: Experiment results from defensive field.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	64%	(64/100)	22.95	3.13	28.28
WarsawB	48%	(48/100)	18.32	1.98	15.31
UBC	57%	(57/100)	30.48	1.71	29.90
Uofa.06	88%	(88/100)	29.69	4.00	40.49
Average	64.25%	(64.25/100)	25.36	2.71	28.50

The use of a defensive field is a great performance improvement of the bot, and it now wins over 64% of the games against the four opponent teams (Table 4).

5.7. Local Optima. To get stuck in local optima is a problem that is well known and that has to be dealt with when using PF. Local optima are positions in the potential field that have higher potential than all their neighbouring positions. An agent positioned in a local optimum may therefore get stuck even if the position is not the final destination for the agent. In the first version of our bot, agents that had been idle for some time moved in random directions for some frames. This is not a very reliable solution to the problem since there are no guarantees that the agents will move out of, or will not directly return to, the local optima.

Thurau et al. [15] describe a solution to the local optima problem called *avoid-past potential field forces*. In this solution, each agent generates a trail of negative potentials on previous visited positions, similar to a pheromone trail used by ants. The trail pushes the agent forward if it reaches a local optimum. We have introduced a trail that adds a negative potential to the last 20 positions of each agent. Note that an agent is not affected by the trails of other own agents. The negative potential used for the trail is set to −0.5.

The use of pheromone trails further boosts the result and our bot now wins 76.5% of the games (see Table 5).

5.8. Using Maximum Potentials. In the original bot, all potential fields generated by opponent units were weighted and summed to form the total potential field which is used for navigation by our agents. The effect of summing the potential fields generated by opponent units is that the highest potentials are generated from the centres of the opponent unit clusters. This makes our agents attack the centres of the enemy forces instead of keeping the MSD to the *closest* enemy. The proposed solution to this issue is that, instead of summing the potentials generated by opponent units and bases, we add the highest potential any opponent unit or base generates. The effect of this is that our agents engage the closest enemy unit at maximum shooting distance instead of trying to keep the MSD to the centre of the opponent unit cluster. The results from the experiments are presented in Table 6.

5.9. A Final Note on the Performance. Our results were further validated in the 2008 ORTS tournament, where our PF-based bots won the three competitions that we participated in (Collaborative Pathfinding, Tankbattle, and Complete RTS). In the Tankbattle competition, we won all 100 games against NUS, the winner of last year, and only lost four of 100 games to Lidia (see Table 7 [14]).

6. Fog of War

To deal with FoW, the bot needs to solve the following issues: remember locations of enemy bases, explore unknown terrain to find enemy bases and units, and handle dynamic terrain due to exploration. We must also take into consideration the increase in computational resources needed when designing solutions to these issues. To enable FoW for only one client, we made a minor change in the ORTS server. We added an extra condition to an IF statement that always enabled Fog of War for client 0. Due to this, our client is always client 0 in the experiments (of course, it does not matter from the game point of view if the bots play as client 0 or client 1). The changes we made to deal with these issues come below.

6.1. Remember Locations of the Enemies. In ORTS, a data structure with the current game world state is sent, each frame from the server to the connected clients. If Fog of War is enabled, the location of an enemy base is only included in the data structure if an own unit is within the visibility range of the base. It means that an enemy base if has been spotted by an own unit and that unit is destroyed, the location of the base is no longer sent in the data structure. Therefore our bot

Table 5: Experiment results from avoid-past potential field forces.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	73%	(73/100)	23.12	3.26	32.06
WarsawB	71%	(71/100)	23.81	2.11	27.91
UBC	69%	(69/100)	30.71	1.72	31.59
Uofa.06	93%	(93/100)	30.81	4.13	46.97
Average	76.5%	(76.5/100)	27.11	2.81	34.63

Table 6: Experiment results from using maximum potential, instead of summing the potentials.

Team	Win %	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	100%	(100/100)	28.05	3.62	46.14
WarsawB	99%	(99/100)	31.82	3.21	47.59
UBC	98%	(98/100)	33.19	2.84	46.46
Uofa.06	100%	(100/100)	33.19	4.22	54.26
Average	99.25%	(99.25/100)	31.56	3.47	48.61

Table 7: Results from the ORTS Tankbattle 2008 competition.

Team	Total win %	Blekinge	Lidia	NUS
Blekinge	98	—	96	100
Lidia	43	4	—	82
NUS	9	0	18	—

has a dedicated global map agent to which all detected objects are reported. This agent always remembers the location of previously spotted enemy bases until a base is destroyed, and distributes the positions of detected enemy tanks to all the own units.

The global map agent also takes care of the map sharing concerning the opponent tank units. However, it only shares momentary information about opponent tanks that are within the detection range of at least one own unit. If all units that see a certain opponent tank are destroyed, the position of that tank is no longer distributed by the global map agent and that opponent disappears from our map.

6.2. Dynamic Knowledge about the Terrain. If the game world is completely known, the knowledge about the terrain is static throughout the game. In the original bot, we created a static potential field for the terrain at the beginning of each new game. With Fog of War, the terrain is partly unknown and must be explored. Therefore our bot must be able to update its knowledge about the terrain.

Once the distance to the closest impassable terrain has been found, the potential is calculated as

$$p_{\text{terrain}}(d) = \begin{cases} -10000 & \text{if } d \leq 1, \\ \dfrac{-5}{(d/8)^2} & \text{if } d \in]1, 50], \\ 0 & \text{if } d > 50. \end{cases} \tag{11}$$

6.3. Exploration. Since the game world is partially unknown, our units have to explore the unknown terrain to locate the hidden enemy bases. The solution we propose is to assign an attractive field to each unexplored game tile. This works well in theory as well as in practice if we are being careful about the computation resources spent on it.

The potential p_{unknown} generated in a point (x, y) is calculated as follows.

(1) Divide the terrain tile map into blocks of 4×4 terrain tiles.

(2) For each block, check every terrain tile in the block. If the terrain is unknown in ten or more of the (at most 16) checked tiles the whole block is considered unknown.

(3) For each block that needs to be explored, calculate the Manhattan Distance md from the center of the own unit to the center of the block.

(4) Calculate the potential p_{unknown} each block generates using (12) below.

(5) The total potential in (x, y) is the sum of the potentials each block generates in (x, y):

$$p_{\text{unknown}}(\text{md}) = \begin{cases} \left(0.25 - \dfrac{\text{md}}{8000}\right) & \text{if md} \leq 2000, \\ 0 & \text{if md} > 2000. \end{cases} \tag{12}$$

6.4. Experiments, FoW Bot. In this experiment set we have used the same setup as in the Tankbattle except that now our bot has FoW enabled, that is, it does not get information about objects, terrain, and so forth that is further away than 160 points from all of our units. At the same time, the opponents have complete visibility of the game world. The results of the experiments are presented in Table 8. They show that our bot still wins 98.5% of the games against the opponents, which is just a minor decrease compared to having complete visibility.

It is also important to take into consideration the changes in the needs for computational resources when FoW is

Table 8: Experiment results when FoW is enabled for our bot.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	100%	(100/100)	29.74	3.62	46.94
WarsawB	98%	(98/100)	32.35	3.19	46.70
UBC	96%	(96/100)	33.82	3.03	47.67
Uofa.06	100%	(100/100)	34.81	4.27	54.90
Average	98.5%	(98.5/100)	32.68	3.53	49.05

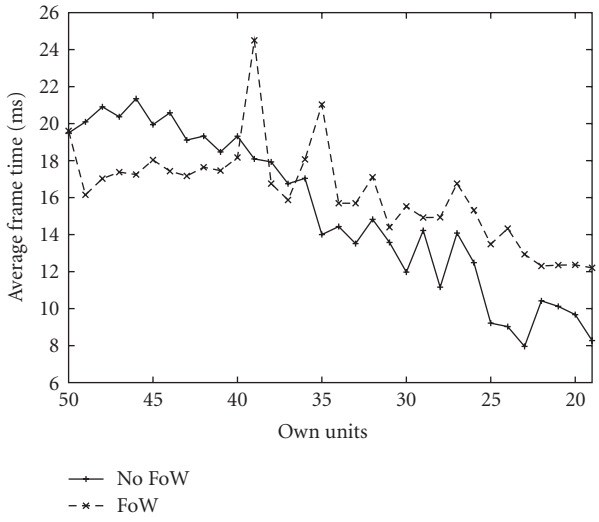


Figure 7: The average frame time used for bots with perfect and imperfect information about the game world.

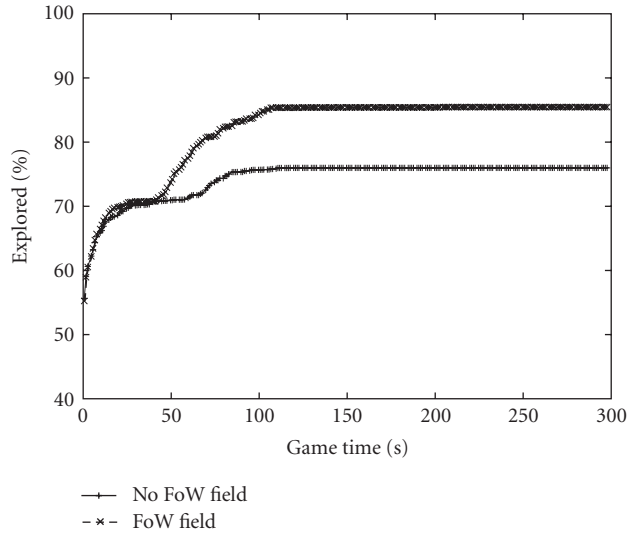


Figure 8: The average explored area given the current game time for a bot using the field of exploration, compared to one that does not.

enabled, since we need to deal with dynamic terrain and exploration field. To show this we have run 100 games without FoW against team NUS and 100 games with FoW enabled. The same seeds are used for both. For each game we measured the average time in milliseconds that the bots used in each game frame and the number of own units left. Figure 7 shows the average frame time for both bots in relation to number of own units left. It shows that the FoW-enabled bot used *less* CPU resources in the beginning of a game, which is probably because some opponent units and bases are hidden in unexplored areas and less potential field-based on opponent units have to be generated. Later in the game, the FoW bot requires more CPU resources probably due to the exploration and the dynamic terrain fields.

In the next set of experiments we show the performance of the exploration field. We ran 20 different games in this experiment, each in which the opponent faced both a bot with the field of exploration enabled, and one where this field was disabled (the rest of the parameters, seeds, etc. were kept identical). Figure 8 shows the performance of the exploration field. It shows how much area that has been explored given the time of the game. The standard deviation increases with the time since only a few of the games last longer than three minutes.

In Table 9, we see that the use of the field of exploration (as implemented here) does not improve the results dramatically. However, the differences are not statistically significant.

Table 9: Performance of the bot with and without field of exploration in 20 matches against NUS.

Version	Won	Lost	Avg. units	Avg. bases
With FoE	20	0	28.65	3.7
Without FoE	19	1	27.40	3.8

7. Discussion

We have shown that the bot can easily be modified to handle changes in the environment, in this case a number of details concerning the agents, the granularity, the fields, and also FoW. The results show that FoW initially decreases the need for processing power and in the end, it had a very small impact on the performance of the bot in the matches. However, this has to be investigated further. In Figure 8, we see that using the field of exploration in general gives a higher degree of explored area in the game, but the fact that the average area is not monotonically increasing as the games go on may seem harder to explain. One plausible explanation is that the games where our units do not get stuck in the terrain will be won faster as well as having more units available to explore the surroundings. When these games end, they do not contribute to the average and the average difference in explored areas will decrease. Does the field of exploration contribute to the performance? Is it at all important to be

able to explore the map? Our results (see Table 9) indicate that—it in this case—may not be that important. However, the question is complex. Our experiments were carried out with an opponent bot that had perfect information and thus was able to find our units. The results may have been different if also the opponent lacked perfect information.

It is our belief that MAPF-based bots in RTS games have great potential even though the scenarios used in the experiments are, from an AI perspective, quite simple RTS scenarios. In most modern commercial RTS games, the AI (and human player) has to deal with base building, economics, technological development, and resource gathering. However, we cannot think of any better testbed for new and innovative RTS games AI research than to test it in competitions like ORTS.

8. Conclusions and Future Work

In Section 4 we introduced a methodology for creating MAPF-based bots in an RTS environment. We showed how to deal with a gathering resources scenario in an MAPF-based bot. Our bot won this game in the 2008 years' ORTS competition, but would have ended up somewhere in the middle in 2007 years' tournament. The bot had some problems with crashes, and more work can be done here to further boost the result.

This was followed by Section 5 where we showed how to design an MAPF-based for playing a tankbattle game. The performance of the first version of our bot was tested in the 2007 years' ORTS competition organized by the University of Alberta. The results, although not very impressive, showed that the use of MAPF-based bots had potential. A number of weaknesses of the first version were identified, solutions to these issues were proposed and new experiments showed that the bot won over 99% of the games against four of the top teams from the tournament. This version of the bot won the 2008 years' tournament with an almost perfect score of 98% wins.

Some initial work has been done in this direction. Our bot quite easily won the full RTS scenario in the 2008 years' ORTS tournament, but more has to be done here. The full RTS scenario in ORTS, even though handling most parts of a modern RTS game, is still quite simple. We will develop this in the future to handle a larger variety of RTS game scenarios.

Another potential idea is to use the fact that our solution, in many ways, is highly configurable even in runtime. By adjusting weights of fields, the speed of the units, and so forth in real time, the performance can be more or less changed as the game goes on. This can be used to tune the performance to the level of the opponent to create games that are more enjoyable to play. One of our next projects will focus on this aspect of MAPF-based bots for RTS games.

Acknowledgments

We would like to thank Blekinge Institute of Technology for supporting our research, the anonymous reviewers, and the organisers of ORTS for providing an interesting application.

References

[1] A. Nareyek, "AI in computer games," *Queue*, vol. 1, no. 10, pp. 58–65, 2004.

[2] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[3] R. C. Arkin, "Motor schema based navigation for a mobile robot: an approach to programming by behavior," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '87)*, vol. 4, pp. 264–271, Raleigh, NC, USA, March 1987.

[4] J. Borenstein and Y. Koren, "The vector field histogram: fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[5] M. Massari, G. Giardini, and F. Bernelli-Zazzera, "Autonomous navigation system for planetary exploration rover based on artificial potential fields," in *Proceedings of the 6th Conference on Dynamics and Control of Systems and Structures in Space (DCSSS '04)*, Riomaggiore, Italy, July 2004.

[6] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[7] A. Howard, M. Matarić, and G. Sukhatme, "Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem," in *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems (DARS '02)*, Fukuoka, Japan, June 2002.

[8] S. J. Johansson and A. Saffiotti, "An electric field approach to autonomous robot control," in *Robot Soccer World Cup V (RoboCup '01)*, Springer, London, UK, 2002.

[9] T. Röfer, R. Brunn, I. Dahm, et al., GermanTeam 2004: the german national Robocup team.

[10] C. Thurau, C. Bauckhage, and G. Sagerer, "Learning human-like movement behavior for computer games," in *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB '04)*, Los Angeles, Calif, USA, July 2004.

[11] J. Hagelbäck and S. J. Johansson, "Using multiagent potential fields in real-time strategy games," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '08)*, L. Padgham and D. Parkesm, Eds., vol. 2, pp. 631–638, Estoril, Portugal, May 2008.

[12] J. Hagelbäck and S. J. Johansson, "The rise of potential fields in real time strategy bots," in *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE '08)*, Stanford, Calif, USA, October 2008.

[13] M. Buro, "ORTS—A Free Software RTS Game Engine," July 2007, http://www.cs.ualberta.ca/~mburo/orts/.

[14] M. Buro, "ORTS RTS game AI competition," August 2008, http://www.cs.ualberta.ca/~mburo/orts/AIIDE08/.

[15] C. Thurau, C. Bauckhage, and G. Sagerer, "Imitation learning at all levels of game-ai," in *Proceedings of the 5th International Conference on Computer Games, Artificial Intelligence, Design and Education (CGAIDE '04)*, pp. 402–408, University of Wolverhampton, Reading, UK, November 2004.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 129075, 10 pages
doi:10.1155/2009/129075

Research Article

Combining AI Methods for Learning Bots in a Real-Time Strategy Game

Robin Baumgarten,[1] Simon Colton,[1] and Mark Morris[2]

[1] *Department of Computing, Imperial College London, London SW7 2AZ, UK*
[2] *Introversion Software Ltd., Glastonbury BA6 BWF, UK*

Correspondence should be addressed to Robin Baumgarten, robin.baumgarten06@doc.ic.ac.uk

Received 31 May 2008; Accepted 21 October 2008

Recommended by Kok Wai Wong

We describe an approach for simulating human game-play in strategy games using a variety of AI techniques, including simulated annealing, decision tree learning, and case-based reasoning. We have implemented an AI-bot that uses these techniques to form a novel approach for planning fleet movements and attacks in DEFCON, a nuclear war simulation strategy game released in 2006 by Introversion Software Ltd. The AI-bot retrieves plans from a case-base of recorded games, then uses these to generate a new plan using a method based on decision tree learning. In addition, we have implemented more sophisticated control over low-level actions that enable the AI-bot to synchronize bombing runs, and used a simulated annealing approach for assigning bombing targets to planes and opponent cities to missiles. We describe how our AI-bot operates, and the experimentation we have performed in order to determine an optimal configuration for it. With this configuration, our AI-bot beats Introversion's finite state machine automated player in 76.7% of 150 matches played. We briefly introduce the notion of ability versus enjoyability and discuss initial results of a survey we conducted with human players.

1. Introduction

DEFCON is a multiplayer real-time strategy game from Introversion Software Ltd., for which a screenshot is provided in Figure 1. Players compete in a nuclear war simulation to score as many points as possible by hitting opponent cities. The game is divided into stages, beginning with placing resources (nuclear silos, fleets of ships, airbases, and radar stations; see Figure 1) within an assigned territory, then guiding fleet manoeuvres, bombing runs, fighter attacks, and finally missile strikes.

The existing single-player mode contains a computer opponent that employs a finite state machine with five states which are carried out in sequence:

(1) placement of ground units and fleet,

(2) scouting by planes and fleet to uncover structures of the opponent,

(3) assaults on the opponent with bombers,

(4) a full strike on the opponent with missiles from silos, submarines, and bombers,

(5) a final state, where fleets of ships approach and attack random opponent positions.

Once the state machine has reached the fifth state, it remains in that state for the remainder of the game. This results in a predictable strategy that may appear monotonous to human players.

We have designed and implemented a novel two-tiered bot to play DEFCON: on the bottom layer, there are enhanced low-level actions that make use of in-match history and information from recorded games to estimate and predict opponent behavior and manoeuvre units accordingly. The information is gathered in influence maps (see also [1]) and is used in synchronous attacks, a movement desire model, fleet formation, and target allocation. On top of these tactical means, we have built a learning system that is employed for the fundamental long-term strategy of a match. The operation of this system is multifaceted and

Figure 1: Screenshot of DEFCON.

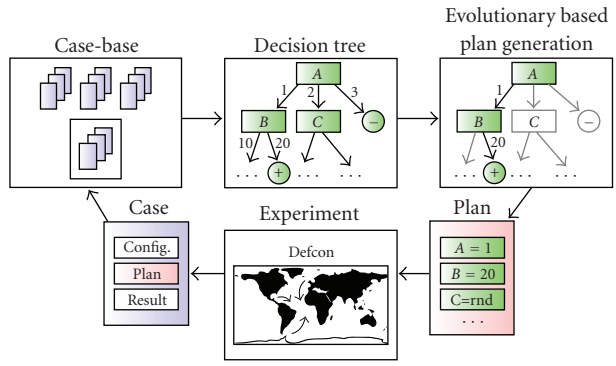


Figure 2: Overview of the system design.

relies on a number of AI techniques, including simulated annealing, decision tree learning, and case-based reasoning. In particular, the AI-bot maintains a case-base of previously played games to learn from, as described in Section 2. It uses a structure placement algorithm to determine where nuclear silos, airbases, and radars should be deployed. To do this, the AI-bot retrieves games from the case-base, ranks them using a weighted sum of various attributes (including life span and effectiveness) of the silos, airbases and radars in the previous game, and then uses the ranking to determine placement of these resources in the game being played, as described in Section 3.

Our AI-bot also controls naval resources, organized into fleets and metafleets (i.e., groups of fleets). Because these resources move during the game, the AI-bot uses a high-level plan to dictate the initial placement and metafleet movement/attack strategies. To generate a bespoke plan to fit the game being played, the AI-bot again retrieves cases from the case-base, and produces a plan by extracting pertinent information from retrieved plans via decision tree learning, as described in Section 4.

During the game, the AI-bot carries out the metafleet movement and attack plan using a movement desire model which takes its context (including the targets assigned to ships and opponent threats) into account. The AI-bot also controls low-level actions at game-time, such as the launching of plane bombing runs, attempting to destroy incoming missiles, and launching missile attacks from fleets. As described in Section 5, we implemented various more sophisticated controls over these low-level actions. In particular, we enabled our AI-bot to synchronize the timing of planes when they attack opponent silos. We also implemented a simulated annealing approach to solve the problem of assigning bombing targets to planes and opponent cities to missiles.

We have performed much experimentation to fine-tune our AI-bot in order to maximize the proportion of games it wins against Introversion's own player simulator. In particular, in order to determine the weights in the fitness function for the placement of silos and airbases, we have calculated the correlation of various resource attributes with the final score of the matches. We have also experimented with the parameters of the simulated annealing search for assignments. Finally, we have experimented with the size of the case-base to determine if/when overfitting occurs. With the most favorable setup, in a session of 150 games, our AI-bot won 76.7% of the time. We describe our experimentation in Section 6.

The superiority of our AI-bot leads to the question of whether higher ability implies higher enjoyability for human players. To this end, we have proposed a hypothesis and conducted an initial survey, which we describe in Section 7. In Sections 8 and 9, we conclude with related work and some indication of future work.

2. Learning from Previous Games

2.1. Learning Cycle Overview. The design of the learning bot is based on an iterative optimization process, similar to that of a typical evolutionary-based process. An overview of the cycle is depicted in Figure 2.

Given a situation requesting a plan, a case-base of previous plan—game pairs is used to select matching plans according to a similarity measure described in the next subsection. This subset of plans is then used in a generalization process to create a decision tree, where each node contains an atomic plan item, as described in Section 4.1. The new plan is generated as a path in the decision tree, derived by starting at the root node and then descending to a leaf node by choosing each branch through a fitness-proportionate selection. The fitness function for the quality of this plan is the game itself, where the AI-bot plays according to the plan, described in Sections 3, 4, and 5. Together with the plan itself, the obtained game data and outcome form a new case. As the decision tree learning requires both positive and negative examples, the new case is retained regardless of the actual outcome of the match.

2.2. A Case-Base of Plans. We treat the training of our AI-bot as a machine learning problem in the sense of [2], where an agent learns to perform better at a task through increased exposure to the task. To this end, the AI-bot is able to store previously played games in a case-base, and retrieve games in order to play its current match more effectively. After a game is played, the AI-bot records the following information as an XML data-point in the case-base:

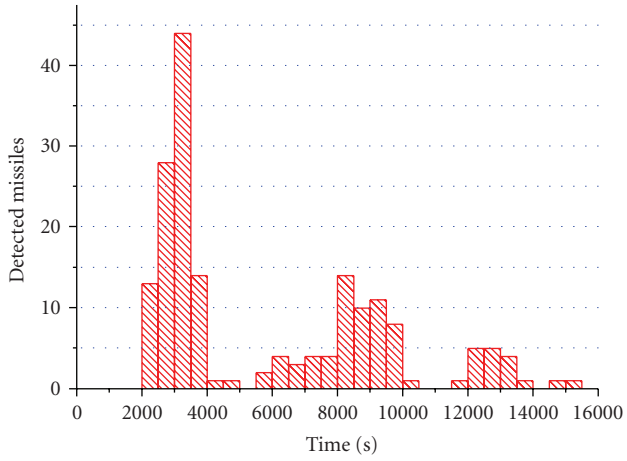


Figure 3: Launched opponent missiles during a match, showing wave-pattern of attacks. The time shown is in-game time.

(i) the starting positions of the airbases, radar stations, fleets, and nuclear silos for both players;

(ii) the metafleet movement and attack plan which was used (as described in Section 4);

(iii) performance statistics for deployed resources which are for nuclear silos the number of missiles attacked and destroyed and planes shot down by each silo, for radar stations the number of missiles identified, and for airbases the number of planes launched and the number of planes which were quickly lost;

(iv) an abstraction of the opponent attacks which took place; we abstract these into waves, by clustering using time-frames of 500 seconds and a threshold of 5 missiles fired (these settings were determined empirically, see Figure 3 for a typical attack distribution);

(v) routes taken by opponent fleets;

(vi) the final scores of the two players in the game.

Cases are retrieved from the case-base using the starting configuration of the game. There are 6 territories that players can be assigned to (North America, Europe, South Asia, etc.), hence there are ${}^6P_2 = 30$ possible territory assignments in a two-player game, which we call the starting configuration of the game. This has a large effect on the game, so only cases with the same starting configuration as the current game are retrieved. For the rest of the paper, we assume that a suitable case-base is available for our AI-bot to use before and during the game. How we populate this case-base is described in Section 6. Cases are retrieved from the case-base both at the start of a game—in order to generate a game plan, as described in Section 4.3—and during the game, in order to predict the fleet movements of the opponent. At the start of a game, cases are retrieved using only the starting territory assignments of the two players.

3. Placement of Silos, Airbases, and Radars

Airbases are structures from which bombing runs are launched; silos are structures which launch nuclear missiles at opponent cities and defend the player's own cities against opponent missile strikes and planes; and radar stations are able to identify the position of enemy planes, missiles, and ships within a certain range. As such, all these structures are very important, and because they cannot be moved at game time, their initial placement by the AI-bot at the start of the game is a key to a successful outcome. The AI-bot uses the previously played games to calculate airbase, silo, and radar placement for the current game. To do this, it retrieves cases with the same starting configuration as the current game, as described above. For each retrieved game, it analyzes the statistics of how each airbase, silo, and radar performed.

Each silo is given an effectiveness score as a weighted sum of the normalized values for the following:

(a) the number of enemy missiles it shot at;

(b) the number of enemy missiles it shot down;

(c) the number of enemy planes it shot down;

(d) the time it survived before being destroyed.

With respect to the placement of silos, each case is ranked using the sum of the effectiveness of its silos. Silo placement from the most effective case is then copied for the current game. The same calculations inform the placement of the radar stations, with the effectiveness given by the following values:

(a) the number of enemy planes detected;

(b) the number of enemy planes detected before other radars;

(c) the number of enemy ships detected;

(d) the time it survived before being destroyed.

Finally, the placement of airbases is determined with these effectiveness values:

(a) the number of planes launched;

(b) the number of units destroyed by launched planes;

(c) the time it survived before being destroyed.

To find suitable weights in the weighted sum for effectiveness, we performed a correlation analysis for the retaining/losing of resources against the overall game score. This analysis was performed using 1500 games played randomly (see Section 4.3 for a description of how randomly played games were generated). In Figure 4, we present the Pearson product-moment correlation coefficient for each of the AI-bot's own resources.

We note that—somewhat counter-intuitively—the loss of carriers, airbases, bombers, fighters, battleships, and missiles is correlated with a winning game. This is explained by the fact that games where fewer of these resources were lost will have been games where the AI-bot did not attack enough (when attacks are made, resources are inevitably

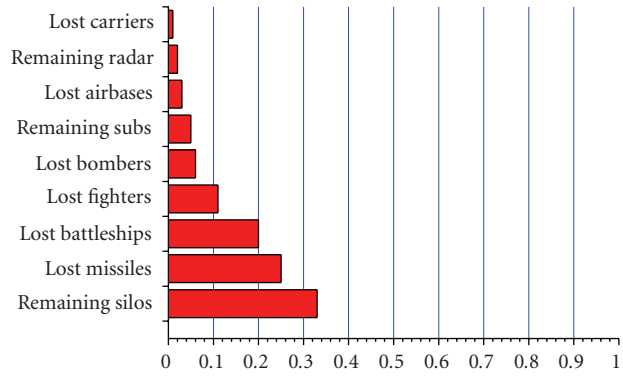


Figure 4: Pearson product-moment correlation coefficient for loss/retention of resources.

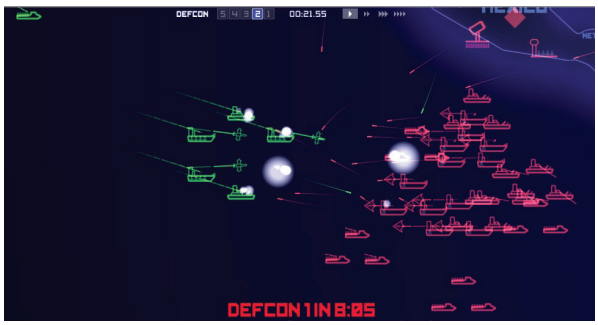


Figure 5: Two fleets attacking each other in DEFCON.

lost). For our purposes, it is interesting that the retention of silos is highly correlated with winning games. This informed our choice of weights in the calculation of effectiveness for silo placement: we weighted value (d), namely, the time a silo survived, higher than values (a), (b), and (c). In practice, for silos, we use 1/10, 1/3, 1/6, and 2/5 as weights for values (a), (b), (c), and (d), respectively. We used similar correlation analyses to determine how best to calculate the effectiveness of the placement of airbases and radar stations.

4. Planning Ship Movements

An important aspect of playing DEFCON is the careful control of naval resources (submarines, battleships, and aircraft carriers). We describe here how our AI-bot generates a high-level plan for ship placement, movement, and attacks at the start of the game, how it carries out such plans (see Figure 5 for a sea attack), and how plans are automatically generated.

4.1. Plan Representation. It is useful to group resources into larger units so that their movements and attacks can be synchronized. DEFCON already allows collections of ships to be moved as a fleet, but players must target and fire each ship's missiles independently. To enhance this, we have introduced the notion of a metafleet which is a collection of a number of fleets of ships. Our AI-bot will typically have a small number of metafleets, (usually 1 or 2) with each one independently targeting an area of high opponent population. The metafleet movement and attack plans describe a strategy for each metafleet as a subplan, where the strategy consists of two large-scale movements of the metafleet. Each subplan specifies the following information.

(1) In what general area (sea territory) the ships in the metafleet should be initially placed, relative to the expected opponent fleet positions.

(2) What the aim of the first large-scale movement should be, including where (if anywhere) the metafleet should move to, how it should move there, and what general target area the ships should attack, if any.

(3) When the metafleet should switch to the second large-scale movement.

(4) What the aim of the second large-scale movement should be, including the same details as for (2).

4.2. Carrying out Metafleet Movements at Game-Time. Sea territories—assigned by DEFCON at the start of a game—are split into two oceans, and the plan dictates which one each metafleet should be placed in. The exact positions of the metafleet members are calculated at the start of the game using the case-base, that is, given the set of games retrieved, the AI-bot determines which sea territory contains on average most of the opponent's fleets. Within the chosen sea territory, the starting position depends on the aim of the first large-scale movement and an estimation of the likelihood of opponent fleet encounter which is calculated using the retrieved games. This estimation uses the fleet movement information associated with each stored game in the case-base. The stored information allows the retrieval of the position of each fleet from the stored game as a function of time. For any given position, the closest distance from that position which each fleet obtains during the game can be calculated. The likelihood of enemy fleet encounter is then estimated by the fraction of games in which enemy fleets get closer to the observed position than a predefined threshold.

There are five aims for the large-scale movements, namely,

(a) to stay where they are and await the opponent's fleets in order to engage them later,

(b) to move in order to avoid the opponent's fleets,

(c) to move directly to the target of the attack,

(d) to move to the target avoiding the opponent's fleet,

(e) to move towards the opponent's fleet in order to intercept and engage them.

The aim determines whether the AI-bot should place the fleets at (i) positions with high-opponent encounter likelihoods (in which case, large-scale movements (a) and (e) are undertaken), (ii) positions with low-opponent encounter likelihoods (in which case, large-scale movements (b) and (d) are undertaken), or (iii) positions which are as close to the attack spot as possible (in which case, large-scale

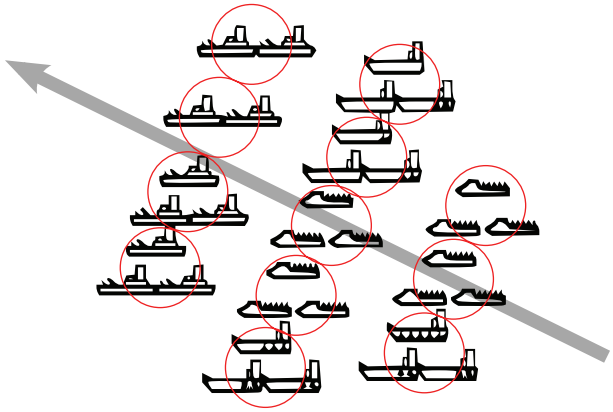


Figure 6: Fleet formation in DEFCON with front direction shown. Each circle indicates a separate fleet of up to three ships.

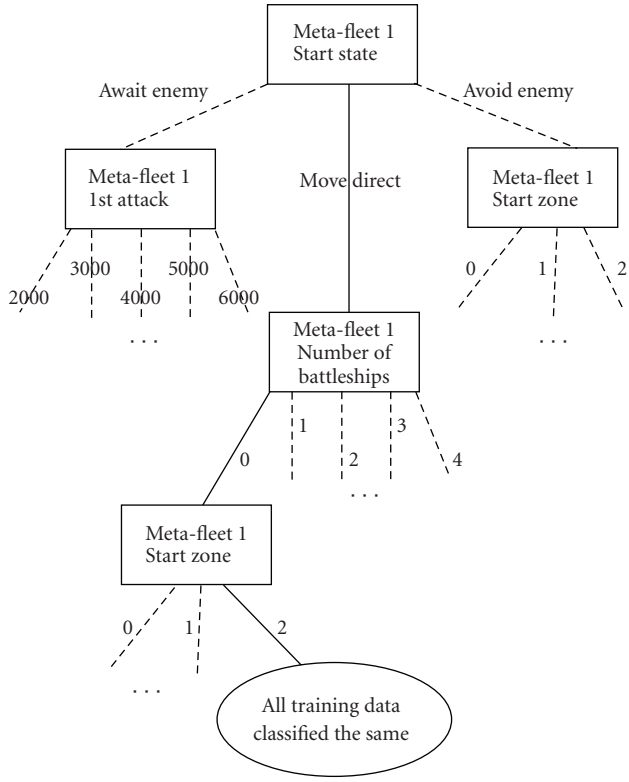


Figure 7: Full selected path through the decision tree. Chosen path is highlighted in bold, not chosen branches are truncated. Remaining plan items are filled in randomly, in this case this is second large-scale movement, first attack time, and number of carriers in metafleet.

movement (c) is undertaken). To determine a general area of the opponent's territory to attack (and hence to guide a metafleet towards), our AI-bot constructs an influence map [1] built using opponent city population statistics. It uses the map to determine the centers of the highest population density, and assigns these to the metafleets.

We implemented a central mechanism to determine both the best formation of a set of fleets into a metafleet, and the direction of travel of the metafleet given the aims of the large-scale movement currently being executed. During noncombative game-play, the central mechanism guides the metafleets towards the positions dictated in the plan (see Figure 6), but this does not take into account the opponent's positions.

Hence, we also implemented a movement desire model to take over from the default central mechanism when an attack on the metafleet is detected. This determines the direction for each ship in a fleet using (a) proximity to the ship's target if this has been specified (b) distance to any threatening opponent ships, and (c) distance to any general opponent targets. A direction vector for each ship is calculated in light of the overall aim of the large-scale metafleet movement. For instance, if the aim is to engage the opponent, the ship will sail in the direction of the opponent's fleets.

The movement desire model relies on being able to predict where the opponent's fleets will be at certain times in the future. To estimate these positions, our AI-bot retrieves cases from the case-base at game-time, and looks at all the various positions the opponent's fleets were recorded at in the case. It then ranks these positions in terms of how close they are to the current positions of the opponent's fleets. To do this, it must assign each current opponent ship to one of the ships in the recorded game in such a way that the overall distance between the pairs of ships in the assignment is as low as possible. As this is a combinatorially expensive task, the AI-bot uses a simulated annealing approach to find a good solution, which is described in more detail in Section 5. Once assignments have been made, the five cases with the closest assignments are examined and the fleet positions at specific times in the chosen retrieved games are projected onto the current game to predict the future position of the opponent's fleets. The five cases are treated as equally likely, thus fleets react to the closest predicted fleet positions according to the aim of their large-scale movement, for instance, approach or avoid it.

4.3. Automatically Generating Plans. As mentioned above, at the start of a game, the starting configuration is used to retrieve a set of cases. These are then used to generate a bespoke plan for the current game as follows. Firstly, each case contains the final score information of the game that was played. These are ranked according to the AI-bot's score (which will be positive if it won, and negative if it lost). Within this ranking, the first half of the retrieved games are labeled as negative, and the second half are labeled as positive. Hence, sometimes, winning games may be labeled negative and, at other times, losing games may be labeled positive. This is done to achieve an example set that generates a more detailed decision tree using the *ID3* algorithm, as described below.

These positive and negative examples are used to derive a decision tree which can predict whether a plan will lead to a positive or a negative game. The attributes of the plan in the cases are used as attributes to split over in the decision tree, that is, the number of metafleets, their starting

sea territories, their first and second large-scale movement aims, and so on. Our AI-bot uses the *ID3* algorithm [2] to learn the decision tree. This algorithm builds a decision tree by iteratively choosing the attribute with the highest information gain (i.e., the amount of noise reduction when splitting the dataset according to the attribute) to split the data. *ID3* is a greedy algorithm that grows the decision tree top-down until all attributes have been used or all examples are perfectly classified. By maximizing the entropy, that is, making sure that there are roughly the same number of negative and positive examples, *ID3* generates a deeper tree, because it takes more steps to perfectly classify the data. As we see below, this is beneficial, as each path in the tree represents a partial plan, with longer paths dictating more specific plans.

We portray the top nodes of an example tree in Figure 7. In this example, we see that the most important factor for distinguishing positive and negative games is the starting sea territory for metafleet 1 (which can be in either low, mid, or high enemy threat areas). Next, the decision tree uses the aim of the second large-scale metafleet movement, the attack time, and the number of battleships in metafleet 1.

Each branch from the top node to a leaf node in these decision trees represents a partial plan, as it will specify the values for some—but not necessarily all—of the attributes which make up a plan. The AI-bot chooses one of these branches by using an iterative fitness-proportionate method, that is, it chooses a path down the tree by looking at the subtree below each possible choice of value from the node it is currently looking at. Each subtree has a set of positive leaf nodes, and a set of negative leaf nodes, and the subtree with the highest proportion of positive leaf nodes is chosen (with a random choice between equally high-scoring subtrees). This continues until a leaf node is reached. Having chosen a branch in this way, the AI-bot fills in the other attributes of the plan randomly. The number of randomly assigned attributes depends on the size of the case-base, for 35 cases this is about 3 attributes.

5. Synchronizing Attacks

In order for players not to have to micromanage the playing of the game, DEFCON automatically performs certain actions. For instance, air defence silos automatically target planes in attack range, and a battleship will automatically attack hostile ships and planes in its range. Players are expected to control where their planes attack, and where missiles are fired (from submarines, bombers, and silos).

5.1. Attacks as Assignment Problems. As mentioned above, the AI-bot uses an influence map to determine the most effective general radius for missile attacks from its silos, submarines, and bombers. Within this radius, it must assign a target to each missile. This is a combinatorially difficult problem, so we frame it as an instance of the assignment problem [3], and our AI-bot searches for an injective mapping between the set of missiles and the set of cities using a simulated annealing heuristic search. To do this, it calculates the fitness of each mapping as the overall population of the cities mapped onto, and starts with a random mapping. Using two parameters, namely the starting temperature S and the cool-down rate c, a pair of missiles is chosen randomly, and the cities they are assigned to are swapped. The new mapping is kept only if the fitness decreases by no more than S times the current fitness. When each missile has been used in at least one swap, S is multiplied by c and the process continues until S reaches a cut-off value.

Figure 8: Synchronized attack in DEFCON.

For most of our testing, we used values $S = 0.5$, $c = 0.9$, and a cut-off value of 0.04, as these were found to be effective through some initial testing. We also experimented with these values, as described in Section 6. Note that the mapping of planes to airbases for landing is a similar assignment problem, and we use a simulated annealing search process with the same S and c values for this.

Only silos can defend against missiles, and silos require a certain time to destroy each missile. Thus attacks are more efficient when the time frame of missile strikes is kept small, so we enabled our AI-bot to organize planes to arrive at a target location at the same time.

To achieve such a synchronized attack, our AI-bot makes individual planes take detours so that they arrive at the time that the furthest of them arrives without detour (see Figure 8 for an example of a synchronized attack using this method). Basic trigonometry gives two possible detour routes and our AI-bot uses the influence map to choose the route which avoids enemy territory the most.

6. Experimentation

We tested the hypothesis that our AI-bot can learn to play DEFCON better by playing games randomly, then storing the games in the case-base for use as described above. To this end, for experiment 1, we generated 5 games per starting configuration (hence 150 games in total), by randomly choosing values for the plan attributes, then using the AI-bot to play against Introversion's own automated player. Moreover, whenever our AI-bot would ordinarily use retrieved cases from the case-base, uninformed (random) decisions were made for the fleet movement method, and the placement algorithm provided by Introversion was used. The

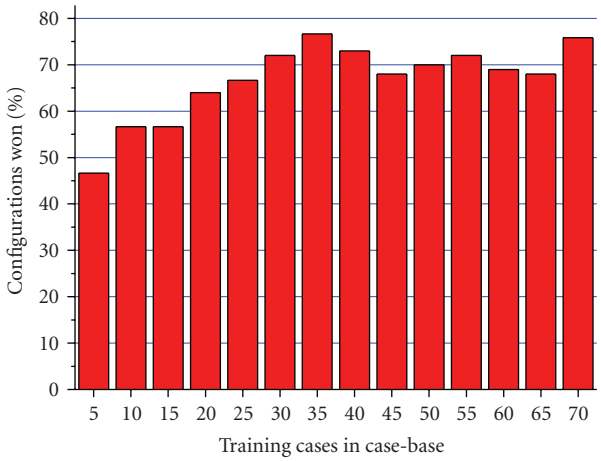


Figure 9: Number of winning configurations versus the size of training data.

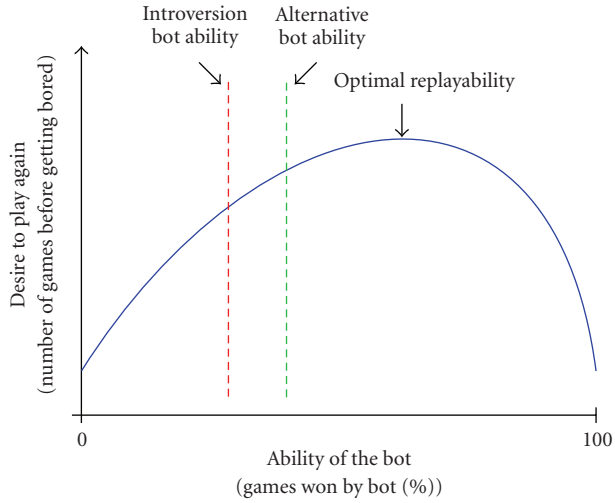


Figure 10: Graph for enjoyability as a function of bot ability (hypothetical).

Table 1: Performance versus simulated annealing parameters.

S	c	Games won (%)	Mean score differential
0	0	53.3	13.3
0.3	0.75	73.3	22.7
0.5	0.9	76.7	33.2
01.01.00	0.95	69.0	34.9
01.01.00	0.99	73.3	33.0

five games were then stored in the case-base. Following this populating of the case-base, we enabled the AI-bot to retrieve and use the cases to play against Introversion's player 150 times, and we recorded the percentage of games our AI-bot won. We then repeated the experiment with 10, rather than 5 randomly played games per starting configuration, then with 15, and so on, up to 70 games, with the results portrayed in Figure 9.

We see that the optimal number of cases to use in the case-base is 35, and that our AI-bot was able to beat Introversion's player in 76.7% of the games. We analyzed games with 35 cases and games with 40 cases to attempt to explain why performance degrades after this point, and we found that the decision tree learning process was more often using idiosyncracies from the cases in the larger case-base, hence overfitting. We describe some possible remedies for overfitting in Section 8.

Using the 35 cases optimum, we further experimented with the starting temperature and cool-down rate of the simulated annealing search for mappings. As described in Section 5, our AI-bot uses the same annealing settings for all assignment problems, and we varied these from no annealing ($S = c = 0$) to very high annealing ($S = 1.0$, $c = 0.99$). As portrayed in Table 1, we recorded both the proportion of wins in 150 games and the score differential between the players. We see that our initial settings of $S = 0.5$, $c = 0.9$ achieved the highest proportion of wins, whereas the setting $S = 1.0$, $c = 0.95$ wins fewer games, but does so more convincingly.

In a final set of experiments, we tried various different metafleet planning setups to estimate the value of learning from played games. We found that with random generation of plans, our AI-bot won 50% of the time, and that by using hand-crafted plans developed using knowledge of playing DEFCON, we could increase this value to 63%. However, this was not as successful as our case-based learning approach, which—as previously mentioned—won 76.7% of games. This gives us confidence to try different learning methods, such as getting our AI-bot to play against itself, which we aim to experiment with in future work.

7. Discussion

We achieved the initial goal of building an AI-bot that can consistently beat the one written by Introversion and included with the DEFCON distribution. Its capability to learn and improve from previous experience makes it more competitive, and thus we can assign a higher *ability* to our bot. This observation leads us to the question of whether increased ability of a computer opponent translates into increased enjoyability for human players.

7.1. Ability versus Enjoyability. We define *enjoyability* in the context of computer games as *the desire to play again* after a match, that is, the number of games a user wants to play before he/she gets bored or frustrated and thus stops enjoying the game. We hypothesize that there is a correlation between ability and enjoyability.

In Figure 10, we portray a hypothetical graph with the ability of the bot (in terms of the percentage of games won against an opponent) plotted against the desire to play again (in terms of the number of games played before the player gets bored). Bots with a fixed strength are therefore points on the ability-axis, indicated by the dotted vertical lines. If our hypothesis is valid, a function similar to the blue graph might emerge. The rationale behind this is that too low or too high

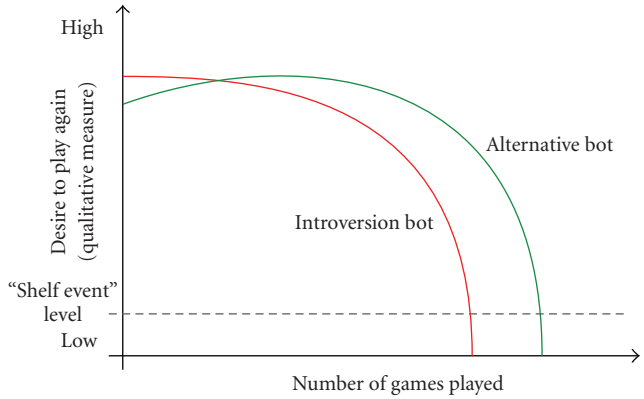


Figure 11: Graph for enjoyability as a function of time (hypothetical).

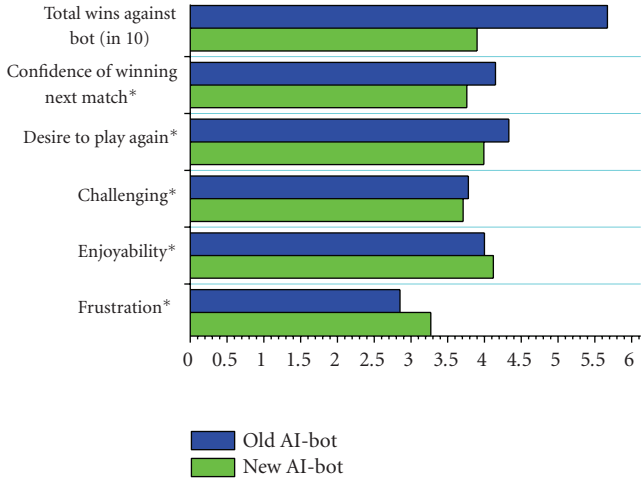


Figure 12: Results of the conducted survey, averaged over 10 games. Values marked with an asterix range from 1 = very low to 6 = very high.

bot ability will bore or frustrate the player and negatively affect his/her desire to play again. The maximum of this function is the ability of a bot that optimizes the enjoyability of playing against it.

Figure 11 shows a hypothetical progression of enjoyability (as a qualitative measure; the player is asked to rate his desire to play again after each match) over time (i.e., over the number of games played against a bot). As the Introversion bot is fairly predictable, we would expect the player to get bored of it after he/she learns how to defeat it quite quickly, indicated by a sharp drop of enjoyability. An alternative, more challenging bot might start with a lower enjoyability as it seems too hard to beat. However, the player's desire to play again should then rise as he/she gets better and learns how to win against the alternative bot. The raised difficulty is expected to delay the drop in enjoyability as it takes longer to consistently win against the bot. We indicate in Figure 11 the idea of a shelf event, that is, a user getting so upset (e.g., through frustration or boredom) with a game that he stops playing it and puts it on his shelf forever. This event can be associated with a very low desire to play again and is indicated as a line in the graph.

7.2. Player Survey. To approach the question of enjoyability versus ability in strategy games, we conducted a pilot survey with students, none of which had played DEFCON before. The sample size of 10 is too small to yield statistically significant results, but it provided valuable responses for further improvement of our AI-bot and helped us to remedy inaccuracies in the test protocol. The test was carried out as a blind test, that is, half the subjects played against the original AI-bot and the other half against our AI-bot. All other game parameters such as starting territories and game mode were identical. After each of the 10 successive matches against their computer opponent, the players were asked to rate their enjoyment, frustration, difficulty, desire to play again and confidence of winning the next match. The results are portrayed in Figure 12.

The results indicate that the novices were overburdened with the increased strength of the new AI-bot, as they won 39% of the games against our AI-bot, while the test group

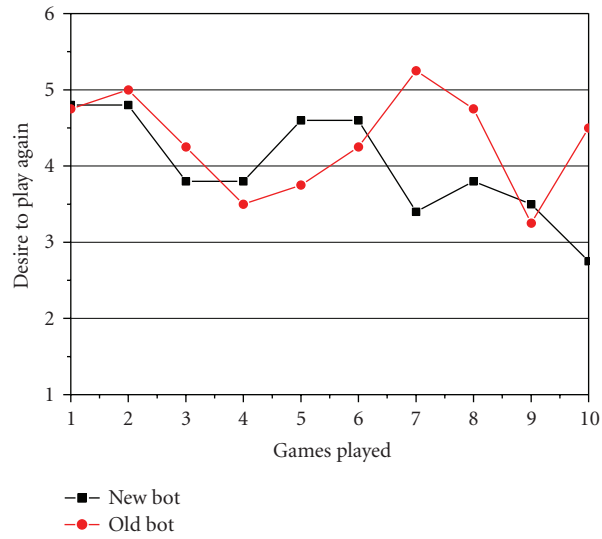


Figure 13: Comparison of survey results on the *desire to play again* (range from 1 = very low to 6 = very high) after each played game. Introversions bot is shown in red, while the new bot is shown in black.

won 56% of the games against the original DEFCON bot. This was reflected in the answers of the questionnaire, where the people playing the original bot were less frustrated (Figure 14) and more confident of winning (Figure 15), which resulted in an often higher desire to play again; see Figure 13. Also, we found that the choice of the starting configuration had a very strong impact on the perceived difficulty of the bots. For instance, in the test games, no player won as *Europe* against the AI-bot playing *South America*. On the other hand, the bots always lost playing *North Asia* against *Europe*. Therefore, great care has to be taken when choosing starting configurations for subsequent surveys.

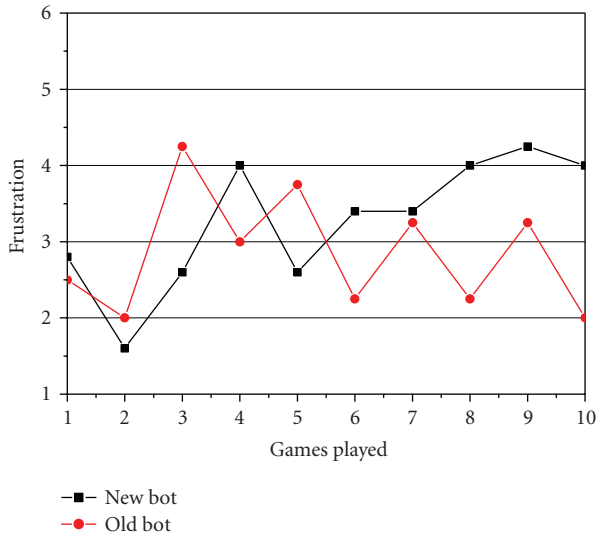


Figure 14: Comparison of survey results on the *frustration* (range from 1 = very low to 6 = very high) after each played game. Introversions bot is shown in red, while the new bot is shown in black.

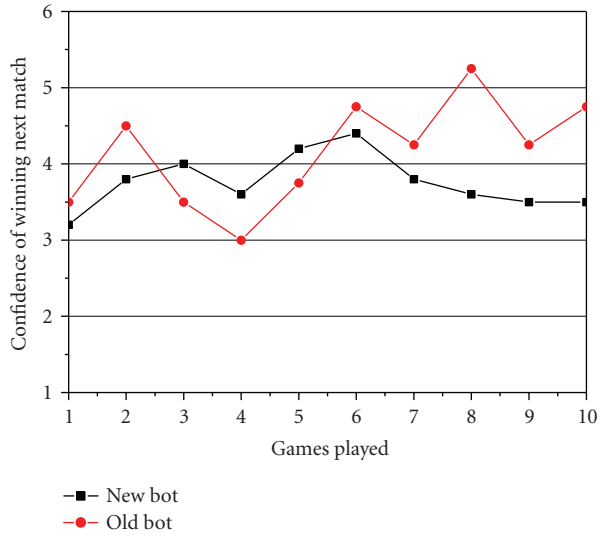


Figure 15: Comparison of survey results on the *confidence of winning the next match* (range from 1 = very low to 6 = very high) after each played game. Introversions bot is shown in red, while the new bot is shown in black.

Regarding our initial hypothesis from Section 7.1, we cannot draw any conclusive results yet, as the sample size and the time-span (i.e., number of games played in the survey) were too small to establish trends. This requires further testing and a survey with a bigger sample size, as described in Section 9. However, our initial survey does indicate that—at least for DEFCON—enjoyability and ability are not correlated in a simple positive manner, which is an interesting finding.

8. Related Work

The use of case-based reasoning, planning, and other AI techniques for board games is too extensive to cover, hence it is beyond the scope of this paper. Randall et al. have used DEFCON as a test-bed for AI techniques, in particular learning ship fleet formations [4]. Case-based reasoning has been used in [5] for plan recognition in order to predict a player's actions when playing the Space Invaders game. They used a simplified planning language which did not need preconditions and applied plan recognition to abstract state-action pairs. Using a plan library derived from games played by others, they achieved good predictive accuracy. In the real-time strategy game Wargus, a dynamic scripting approach was shown to outperform hand-crafted plans, as described in [6]. Moreover, hierarchical planning was tested in an Unreal Tournament environment by [7], who showed that this method had a clear advantage over finite state machines.

A comparison of artificial neural networks and evolutionary algorithms for optimally controlling a motocross bike in a video game was investigated in [8]. Both methods were used to create riders which were compared with regard to their speed and originality. They found that the neural network found a faster solution but required hand crafted training data, while the evolutionary solution was slower, but found solutions that had not been found by humans previously. In commercial games, scripting and/or reactive behaviors have in general been sufficient to simulate planning, as full planning can be computationally expensive. However, the Dark Reign game from Activision uses a form of finite state machines that involves planning [9], and the first-person shooter game F.E.A.R employs goal-oriented action planning [10].

8.1. Other Applications. Although the developed bot is in itself already an application of the used techniques, the underlying concept of combining artificial intelligence methods to benefit from synergy effects is applicable to many problems, including, but not restricted to, other strategy computer games that have similar requirements of optimizing and planning actions to be able to compete with skilled humans.

In particular, the combination of case-bases and decision trees to retrieve, generalize, and generate plans is a promising approach that is applicable to a wide range of problems that exhibit the following properties.

(i) *Discrete attributes.* The problem state space must be discrete or discretizable. This is required for decision tree algorithms to build trees. Attributes with a low cardinality are preferable, as a high number of possible values can cause problems with the decision tree learning algorithm.

(ii) *Recognizable opponent states.* Problem instances must be comparable through a similarity measure, which is required for retrieving cases. In a game domain, it should be based on opponent attributes or behavior to allow an adaptation to take place.

(iii) *Static problem domain.* The interpretation of a plan has to be constant, or else the similarity measure might retrieve irrelevant cases that show similarity to an obsolete interpretation. This also means that, for a hierarchical planner, lower-level plans should not change much when reasoning on high-level plans, as the case-base is biased towards previously successful plans.

(iv) *Availability of training sets.* The problem has to be repeatable or past instances of problem-solution pairs have to be available to train the case-base.

8.2. Future Work. There are many ways in which we can further improve the performance of our AI-bot. In particular, we aim to lessen the impact of over-fitting when learning plans, by implementing different decision tree learning skills, filling in missing plan details in nonrandom ways, and by trying other logic-based machine learning methods, such as Inductive Logic Programming [11]. We also hope to identify some markers for success during a game, in order to apply techniques based on reinforcement learning. There are also a number of improvements we intend to make to the control of low-level actions, such as more sophisticated detours that planes make to synchronize attacks.

With improved skills to both beat and engage players, the question of how to enable the AI-bot to play in a multiplayer environment can be addressed. This represents a significant challenge, as our AI-bot will need to collaborate with other players by forming alliances, which will require opponent modelling techniques. We aim to use DEFCON and similar video games to test various combinations of AI techniques, as we believe that integrating reasoning methods has great potential for building intelligent systems. To support this goal, we are developing an open AI interface for DEFCON, which is available online at http://www.introversion.co.uk/defcon/bots/.

The initial results of the survey and the discussion of ability versus enjoyability raise another important point for the future direction of our research. Usually it is a practice in academic AI research to strive for an algorithm that plays at maximum strength, that is, it tries to win at all costs. This is apparent in the application of AI techniques to playing board games. Chess playing programs, for example, usually try to optimize their probabilities of winning. However, this behavior may be undesirable for opponents in modern video games. It is not the goal of the game to make the player lose as often as possible, but to make him/her enjoy the game. This may involve opponents that act nonoptimally, fall for traps, and make believable mistakes. This behavior is another aspect we hope to improve in our bot in the future. It also suggests further player studies, as it is imperative to evaluate the enjoyability and believability of a bot through player feedback.

9. Conclusion

We have implemented an AI-bot to play the commercial game DEFCON, and showed that it outperforms the existing automated player. In addition to fine-grained control over game actions, including the synchronization of attacks, intelligent assignment of targets via a simulated annealing search, and the use of influence maps, our AI-bot uses plans to determine large-scale fleet movements. It uses a case-base of randomly-planned previously played games to find similar games, some of which ended in success while others ended in failure. It then identifies the factors which best separate good and bad games by building a decision tree using ID3. The plan for the current game is then derived using a fitness-proportionate traversal of the decision tree to find a branch which acts as a partial plan, and the missing parts are filled in randomly. To carry out the fleet movements, ships are guided by a central mechanism, but this is superceded by a movement-desire model if a threat to the fleet is detected.

References

[1] P. Sweetser, "Environmental awareness in game agents," in *AI Game Programming Wisdom 3*, S. Rabin, Ed., vol. 3, Charles River Media, Boston, Mass, USA, 2006.

[2] P. Tozour, "Influence mapping," in *Game Programming Gems*, vol. 2, Charles River Media, Boston, Mass, USA, 2001.

[3] T. Mitchell, *Machine Learning*, McGraw-Hill, New York, NY, USA, 1997.

[4] M. R. Wilhelm and T. L. Ward, "Solving quadratic assignment problems by 'simulated annealing'," *IIE Transactions*, vol. 19, no. 1, pp. 107–119, 1987.

[5] T. W. G. Randall, P. I. Cowling, and R. J. S. Baker, "Learning ship combat strategies in the commercial video game DEFCON," in *Proceedings of the 8th Informatics Workshop for Research Students*, Bradford, UK, June 2007.

[6] M. Pagan and P. Cunningham, "Case-based plan recognition in computer games," in *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR '03)*, vol. 2689 of *Lecture Notes in Artificial Intelligence*, pp. 161–170, Trondheim, Norway, June 2003.

[7] P. H. M. Spronck, *Adaptive Game AI*, SIKS Dissertation Series, Universitaire Pers Maastricht, Maastricht, The Netherlands, 2005.

[8] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical plan representations for encoding strategic game AI," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '05)*, AAAI Press, Marina del Rey, Calif, USA, June 2005.

[9] B. Chaperot and C. Fyfe, "Motocross and artificial neural networks," in *Proceedings of the 3rd Annual International Conference in Game Design and Technology (GDTW '05)*, Liverpool, UK, November 2005.

[10] I. Davis, "Strategies for strategy game AI," in *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Computer Games*, pp. 24–27, Palo Alto, Calif, USA, March 1999.

[11] J. Orkin, "Agent architecture considerations for real-time planning in games," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '05)*, AAAI Press, Marina del Rey, Calif, USA, June 2005.

[12] S. Muggleton, "Inductive logic programming," *New Generation Computing*, vol. 8, no. 4, pp. 295–318, 1991.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 456169, 9 pages
doi:10.1155/2009/456169

Research Article

Enhancing Artificial Intelligence on a Real Mobile Game

Fabio Aiolli and Claudio E. Palazzi

Department of Pure and Applied Mathematics, University of Padova, Via Trieste 63, 35131 Padova, Italy

Correspondence should be addressed to Claudio E. Palazzi, cpalazzi@math.unipd.it

Received 30 May 2008; Accepted 8 September 2008

Recommended by Kok Wai Wong

Mobile games represent a killer application that is attracting millions of subscribers worldwide. One of the aspects crucial to the commercial success of a game is ensuring an appropriately challenging artificial intelligence (AI) algorithm against which to play. However, creating this component is particularly complex as classic search AI algorithms cannot be employed by limited devices such as mobile phones or, even on more powerful computers, when considering imperfect information games (i.e., games in which participants do not a complete knowledge of the game state at any moment). In this paper, we propose to solve this issue by resorting to a machine learning algorithm which uses profiling functionalities in order to infer the missing information, thus making the AI able to efficiently adapt its strategies to the human opponent. We studied a simple and computationally light machine learning method that can be employed with success, enabling AI improvements for imperfect information games even on mobile phones. We created a mobile phone-based version of a game called *Ghosts* and present results which clearly show the ability of our algorithm to quickly improve its own predictive performance as far as the number of games against the same human opponent increases.

1. Introduction

Mobile phones have brought into our lives the possibility and the willingness to be always reachable by anybody; they have almost become an extension of ourselves, making us *Homo Mobilis* for how much we tend to never separate from them [1]. Service providers are riding this wave by continuously offering new appealing services. Among the others, mobile games represent a great and ever-increasing source of revenue in the mobile service market. Indeed, market studies report incomes for the wireless gaming industry in the order of billions of US dollars worldwide and with a 40% growth each year [2–4].

Gaming has always been one of people's favorite digital applications. Nowadays, three main reasons have enabled its success even in the mobile market [5]. First, the incredible proliferation of mobile phones, which have surpassed in number base line phones in countries such as Finland and Italy, thus creating hundreds of millions of potential customers. Second, technological advances have transformed mobile phones from a cordless version of a regular phone into a hand-held computer able to deliver quality audio/video and quickly run complex algorithms, as those required by recent games. Third, the increasing availability of wireless connectivity (i.e., GPRS, UMTS, Bluetooth, Wi-Fi) provides the possibility to play online with other people and allows to create new business models where the game is bought online and directly downloaded in the mobile phone.

While the trend toward a massive use of mobile games is out of dispute, several technical problems remain unsolved; the market success of future (and present) mobile games also passes through providing valid answers to them. For instance, a challenging research issue is related to the availability of an adequate AI algorithm. Indeed, users often play alone against the AI; this could happen to avoid the connectivity cost to play against another mobile user, or because a mobile phone may have gaming capabilities but no connectivity, or just because the user prefers so. Therefore, the game has to be endowed with an AI that is fun to play against: neither too trivial, nor too tough. Unfortunately, classic searching techniques may not be feasible for the considered context for two main reasons: (i) a game could be played on mobile phones with limited computational power, (ii) these techniques might not function when considering

certain games. Elaborating on this second point, we have to remember that games can be classified into two main categories, *perfect information games* (e.g., Chess) and *imperfect information games* (e.g., Poker), depending on whether participants have or do not have a complete knowledge of the game state at any moment. When players have just a very limited knowledge of the game state, resorting to traditional state space searches, may result in an AI which behaves similarly to a random decision maker.

As a case study for this problem, we have created a mobile phone-based version of an imperfect information game named *Ghosts,* a simple board game played by two opponents. The board game has been invented by Alex Randolph and is sold in Germany by Drei Magier Spiele. Its original (German) name is: "*Die guten und die bösen Geister,*" that is, "good and bad ghosts;" for brevity, we simply name it *Ghosts*. The game is particularly interesting for our study as players do not have a complete knowledge of the game state: they can both see the position of game pieces on the board, but they cannot see the type of the opponent's ones. Depending on this information, different tactics will be adopted (i.e., attack the opponent's piece, leave it alone, run away from it). Therefore, in order to win, a player has also to infer the type of each of the opponent's pieces. This information can be extracted from the player's behavior, also keeping in mind that different players can adopt different strategies, for instance, by resorting more or less frequently to bluffing.

To this aim, as main research contribution of this work, we have developed a new kind of AI solution that is able to adapt the gaming strategy to the current human player inferring unknown information about the game state. Our solution employs machine learning techniques to mimic the human's ability in intuiting the opponent's intentions after several game sessions and is hence particularly helpful with imperfect information games. Simply, it associates the behavior features of a player with a presumed type of a piece; by observing how a player acts in different game state configurations, the AI becomes able to classify tactics employed by that player and to adapt to them.

Even if this AI solution represents our main scientific contribution, while creating our mobile version of the game *Ghosts*, we have not overlooked at two practical problems that are crucial in the successful deployment of a real mobile game: (i) compatibility with the highest number of mobile phones in the market and (ii) connectivity among players' mobile phones [5]. To this aim, we have evaluated possible alternatives and finally adopted the state-of-the-art technology that allows the widest compatibility and connectivity among existing mobile phones.

The rest of the paper is organized as follows. Since we developed a real mobile game with an original AI solution, in Section 2, we discuss development issues that are at the basis of implementative choices and, in Section 3, we overview related AI scientific works. Then, in Section 4, we technically present our developed mobile game. As our novel AI approach represents the main research contribution of this work, we devote Section 5 to discuss its details. Section 6 describes the experimental scenario and reports the corresponding outcomes. Finally, in Section 7, we provide a conclusion and future directions for this work.

2. Technical Development Background

In this section we provide background information about two technical issues which are very important to take into account in order to develop a successful mobile game, namely, compatibility and connectivity.

2.1. Compatibility. The current mobile gaming panorama is affected by a significant fragmentation problem that precludes making a game available to the entire mobile market [6]. This is caused by both the absence of a standard in terms of software platform for mobile phones [5] and by the specific characteristics (e.g., screen size) of different phones, even when produced by the same manufacturer. Indeed, if game designers wanted to fully exploit the features of a given device, they should renounce to have that game also running on entry-level phones. Another practiced solution is that of developing multiple versions of the same game to have one version specifically designed for each class of mobile phone; clearly this solution has a cost.

Currently, we can identify three software platforms that emerge as the most popular ones when considering mobile games with connectivity capabilities: Symbian [7], Binary Runtime Environment for Wireless (BREW) [8], and Java Micro Edition (Java ME) [9]. The first one is a proprietary operating system that has been developed by a consortium among Nokia, Sony Ericsson, Siemens, Panasonic, and Samsung. As these brands represent a very wide portion of the global mobile market, Symbian can be considered a very popular operating system. Symbian applications have also the advantage of being fast as they are generally written in C++ and can make use of specific features of the considered mobile phone. This may require specific programming skills and raises compatibility issues with other software platforms.

BREW is a development platform for mobile phones and it is based on C++. As a main advantage, BREW is located between the application layer and the operating system of the mobile phone; this way, it offers a simple interface to the programmer to handle different system/networking details. Unfortunately, BREW is not a free platform and this significantly limits its popularity.

Finally, Java ME is a Java application environment designed for devices with limited capabilities in terms of processing power, memory, and graphics. Its Connected Limited Device Configuration (CLDC) is composed by a set of libraries specifically designed to run Java applications on limited devices; this set of libraries is extended by the Mobile Information Device Profile (MIDP) that embodies a set of APIs for the GUI, the data storage, and the networking functionalities. The latest release, MIDP 2.0, provides specific APIs also to generate 3D graphics for games. Mobile applications written in Java ME are named MIDlet and, although they do not run as fast as applications purposely designed for a particular device or platform, their

main advantage is that, potentially, they can be run on any Java ME-compatible device.

All these platforms have both pros and cons, thus demonstrating the need for a solution that will enable the automatic porting of any mobile game on any mobile phone. However, this is not the aim of this work; we simply note that currently Java ME with MIDP 2.0 is the solution that provides the widest portability of the developed software.

2.2. Connectivity. Different communication technologies are available today on most of mobile phones (e.g., GPRS, UMTS, Bluetooth, Wi-Fi); thereby, being able to exploit them has become an important aspect in the success of a mobile game [5]. The current mobile scenario is dominated by 2G and 3G (GPRS and UMTS, resp.). Phone service providers have done huge investments on this technology; therefore, this communication means present the advantage of being available almost anywhere. Yet, its bandwidth, latency, and cost often block users from using it.

Bluetooth connectivity is also very popular today as only really cheap mobile phones are produced today without it. Bluetooth was designed to implement personal area networks (PANs) and, thereby, its bandwidth and latency also allow to support multimedia applications [10]. Transmissions happen only in a ~10 m range, which implies that players have to be one in front of (or beside) the other to play together; yet, this proximity in the real world is often part of the fun of playing together. Finally, Bluetooth communications are not billed.

Wi-Fi is another communication technology that can be free of charge (or available at a low fixed cost). Its transmission range is in the order of 100 m but can be used as well to play online with other people all around the world by simply connecting to the Internet through an access point in proximity [11]. Unfortunately, Wi-Fi capabilities are currently present only on expensive mobile phones; plus, while walking in a street, there might not be around any freely accessible Wi-Fi access point thus impeding its use.

The optimal solution would be that of having the game enabled to work on any of the aforementioned connectivity means and choosing, at any moment, the "best" among the available ones (e.g., the fastest, the cheapest, the most reliable) [12, 13]. However, if a mobile game producer decides to create a game with only one connectivity option, we deem that the chosen one should be Bluetooth as it is available on almost any new mobile phone and its use is free of charge. The combination of these two characteristics makes users willing to use it for their leisure.

3. Artificial Intelligence Related Work

In this paper, we present a novel AI approach for imperfect information games; we hence deem important to devote this section to provide a discussion about related work in AI.

The first self-learning gaming program, that is, Checkers, was created in 1959 and represented a very early demonstration of the fundamental concept of AI in games [14]. Nowadays, all video games include some AI that may act as a virtual opponent or as a component of the game itself. Yet, the AI of current games show only little advancements if compared to its ancestors; only for few specific games the AI has achieved great improvements (e.g., Chess [15]).

As already stated in Section 1, games can be categorized into two main classes: games where the players possess perfect information about the current game state (e.g., Chess, Tic-tac-toe) and games where players can rely on imperfect information only (e.g., Poker, Rock-paper-scissors). The AI of perfect information games can easily evaluate a given game state by just searching all possible continuations to a fixed depth. For this kind of games, the main problem in developing an AI is related to the capability of precomputing correct evaluations of each game state and then storing and retrieving them in an efficient manner [16–18].

Instead, with imperfect information games, deep search may not be feasible and storing precomputed evaluations might not result in significant improvements in the AI's strength [19, 20]. In this case, techniques like temporal difference learning are also unsuitable as the intermediate states of a game are only partially determined [21]. Alternative solutions have hence to be explored to enhance the level of the AI. For instance, simulation search [22–25] evaluates the possible next moves by self-playing a multitude of simulated game sessions, considering the current state as the starting point and utilizing different values for the indeterministic parameters (i.e., dice rolls, cards held by the opponent player, etc.). To generate real-time responses during the game, these simulations can be run before the game and statistics can be stored to be promptly available during the game. Unfortunately, the branching factor of certain games may considerably limit the effectiveness of this technique.

To this aim, in Section 5 we discuss a new machine learning approach that mimics the human's ability in evaluating important information about the current game state that goes beyond, for instance, the board position [26]. In essence, our mechanism models the opponent's behavior over several game sessions so as to be able to exploit the weaknesses and the typical behaviors of the considered human player.

4. A Representative Case of Imperfect Information Game: *Ghosts*

For our study, we need a simple, yet representative exemplar of imperfect information game. *Ghosts* embodies a perfect case as, not only it belongs to this class of games, but it is also governed by few simple rules, which makes it easier to appreciate the improvement produced by the employment of our AI solution.

The rules of *Ghosts* are listed hereinafter. Two players have to place 8 ghosts each at the back of a 6×6 board as shown in Figure 1. Each player has 4 good ghosts and 4 bad ghosts, but the information about which are good and which are bad is hidden to the opponent player. On each turn, a player moves one of her/his ghosts one square vertically or horizontally; if by doing so the ghost is moved onto an opponent's ghost, the latter is captured by the former. In

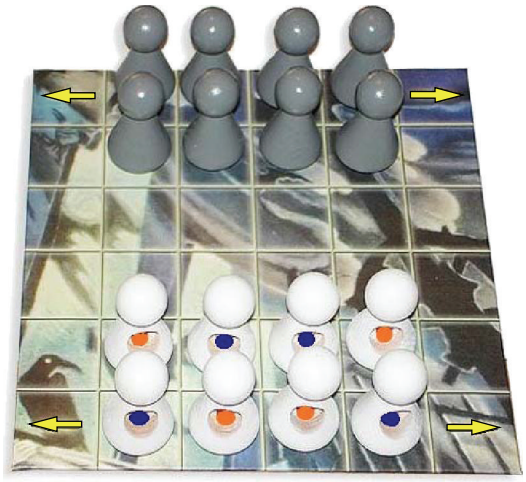


Figure 1: Initial setup of *Ghosts*.

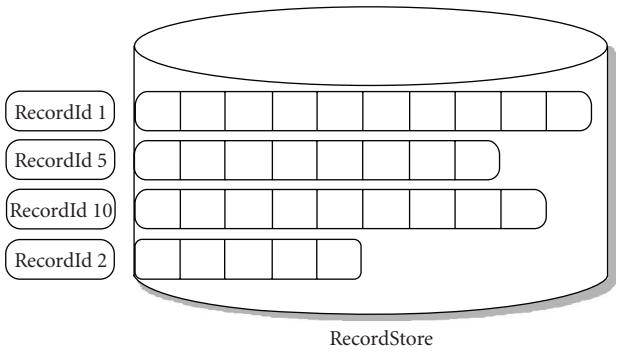


Figure 2: The record management system.

order to win, different possibilities are available to a player: (i) having all of her/his bad ghosts captured by the opponent player, (ii) capturing all the good ghosts of the opponent player, (iii) moving one of her/his good ghosts off the board from one of the opponent's corner squares. Clearly, one of the interesting aspects of *Ghosts* is its bluffing element which is differently utilized by different human players.

4.1. Development of Ghosts for Mobile Phones. The development of the digital version of *Ghosts* for mobile phones has passed through three different phases: creating the GUI and the basic logic of the game, enabling communication among two mobile phones, adding the AI with our player profiling capability. We discuss in this section the first two of these phases, which are related to the practical implementation of the game; we leave the third one to the next section as its research contribution deserves a deeper and different kind of discussion.

To create the game, we decided to use Java ME. This choice was motivated by several reasons. First, developing mobile games through the APIs of MIDP 2.0 is quite simple. Second, Java ME comes also with an emulator that facilitates software development before trying it on real mobile devices. Third, we wanted to create a game with the highest portability on different devices; indeed, we tested our final game on various mobile phones (Nokia 7610, 6630, 6670, 6260, N70, N95) and it perfectly worked in all cases.

Unfortunately, memorizing data related to a mobile game cannot be done in a trivial way as we would do with regular computer-based games. The problem is related to the employed mobile device; indeed, in our phone, we had to resort to a specific data structure named Record Management System (RMS). More in detail, Figure 2 shows that RMS is modeled on the basis of a simple database where stored data are organized in records.

The connectivity of the game has been ensured through Bluetooth. We are currently adding the possibility to exploit Wi-Fi and we plan to enable GPRS and UMTS in the future; however, having to choose just one connectivity means to start with, we preferred Bluetooth since it is available on most of the mobile phones on the market, its use is free of charge, and it does not require networking infrastructure (i.e., access points, game servers) to work.

With Bluetooth connectivity, one of the involved devices has to act as a server and the other(s) can be client(s). In our game, which phone acts as a server and which one as a client is explicitly chosen by the two players when starting the game session. With the help of Figure 3 we describe the various phases to initiate a game session among two mobile phones. First, our *Ghosts* application has to be started on both mobile phones to enable the local device. A choice is made between activating the game as a server or as a client; then, in the former case, the service is created and waits for a client, whereas in the latter case, the client has to search for devices within its transmission range. Once a device is found, the client searches for a specific service (i.e., the game server) on that device; if the service is discovered then the connection is established and messages (e.g., game moves) can be transmitted in turn from one device to the other in order to play the game (our game prototype running on a mobile phone is shown in Figure 4).

5. Enhancing the AI with Player Profiling Capabilities

The last component of our mobile game is represented by the AI algorithm that allows players to play in solo mode; it is also the most interesting from a research point of view.

Problems in game AI are typically grouped with respect to certain characteristics. One of the most intriguing is related to how much information is available to the players. This leads us to distinguish between perfect information games and imperfect information games. The latter is more interesting as it represents a more challenging case.

Ghosts embodies a good case study as it falls in the class of imperfect information games, yet, it is simple enough to be analyzed. In *Ghosts*, a player does not have any information about the type of the opponent's ghosts (i.e., good or bad); thereby, any search state space-based technique, for example, min-max algorithms, will fail. In other words, without any other heuristic judgment, the behavior of a machine driven player could not be better than any trivial random player.

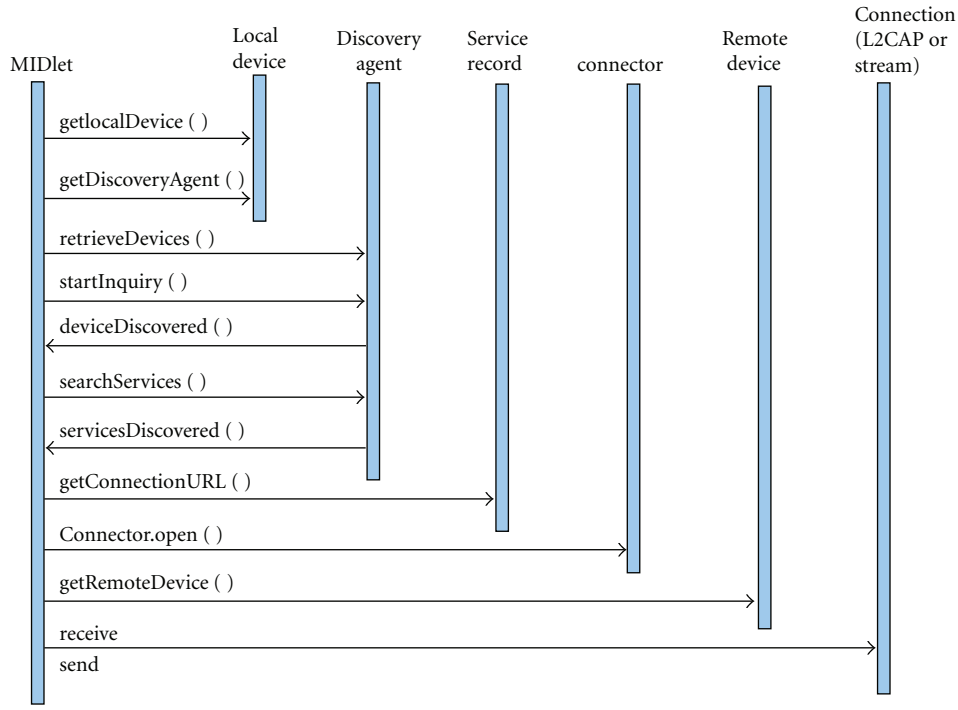


Figure 3: Connection steps through Bluetooth.

Figure 4: Our final *Ghosts* game running on a mobile phone.

Indeed, in order to plan its moves, an AI algorithm would certainly benefit from some other source of information about the type of the opponent's ghosts (the missing information). We propose to get this additional information from the playing style of a player. The basic assumption we make is that different players have different playing behaviors (being aggressive, bluffing, etc.) and they tend to move pieces of a certain type in a similar way when facing similar game situations. Specifically, our claim is that by knowing the playing style of a player, it is possible to recognize the type of a given ghost by its moves. We can then use a machine learning algorithm [21] to compile a behavior profile of good and bad pieces of a player. During future game sessions, this knowledge can be used to predict the type of a ghost in the board and possibly to define higher level game heuristics, like min-max algorithms, based on these predictions. This kind of prediction can be easily seen as a classification problem in machine learning, as discussed in the following section.

5.1. Machine Learning for Classification. One of the most important problems in machine learning is classification. In this setting, given a set of preclassified instances, one wants to predict the class of new instances never seen before. In the simplest case we have two labels, say $\{-1, +1\}$, denoting that an instance belongs $(+1)$, or not belongs (-1), to a given concept (or class). Instances of the problem are described by d-dimensional vectors, each component containing the value of a given attribute (feature) relative to the instance. In general, it is assumed that the instance-class pairs are drawn according to a fixed (but unknown) probability distribution. Moreover, the set of preclassified examples, that is, the training set, is assumed to be drawn according to the same probability distribution.

Various algorithms have been designed to solve this problem; they can be grouped into two big families: discriminative and nondiscriminative methods. Given any class, a nondiscriminative method tries to estimate the probability of an instance using the training examples; then, it uses the Bayes rule (or similar, [27]) to estimate the probability of each class given an instance; finally, through these estimates, any new instance can be classified with the most likely class. Note that each class is treated independently and the estimation of the probability of instances given the class can be done by using instances of that class only.

A discriminative method tries to directly estimate the posterior probability of each class given the instance. For this last estimate, all the examples of the training set have to be

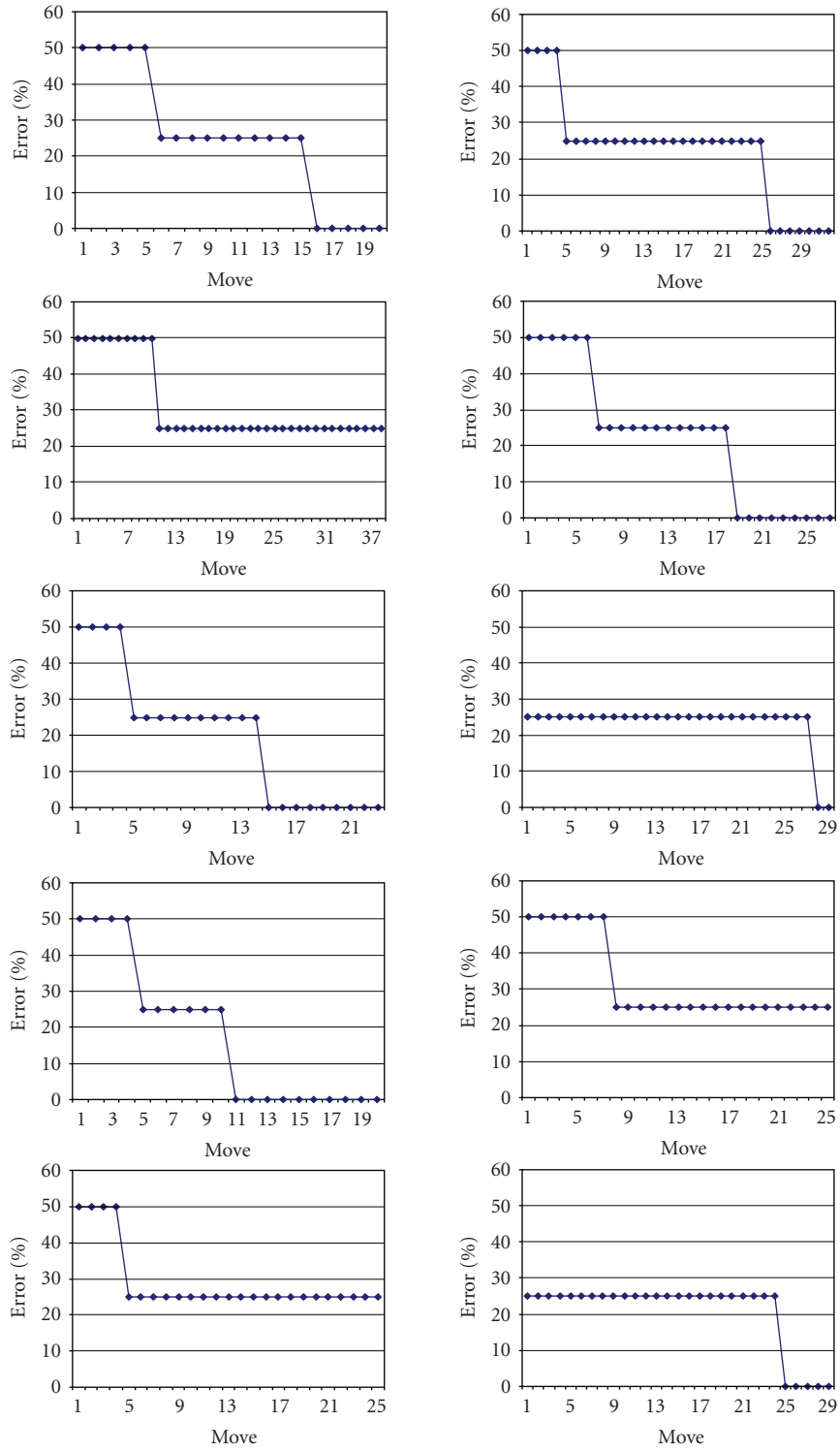


Figure 5: Percentage of error of the prototype-based machine learning algorithm (proportion of times a type of a piece on the board is wrongly predicted) for 10 different game sessions. The x-axis represents the moves of the player during that game session (the number of moves for a game varies on different game sessions).

used together. Generally, a nondiscriminative method tends to be more efficient and computationally less demanding than discriminative methods. However, discriminative methods usually have best performances.

One of the most successful discriminative classification methods is the Support Vector Machine (SVM) [28]. With this method, instances are mapped into a very large feature space where a linear separator (a hyperplane) is found by

the learning algorithm. Advantages of SVM with respect to previously devised methods, like neural networks, for example, are the solid theoretical framework on which they are defined and the small number of hyperparameters that a user has to tune to use them. Furthermore, since the SVM solution can be found by optimizing the dual of a convex constrained quadratic objective function, this solution does not suffer of local minima as neural networks do. Moreover, the solution can be expressed in the very simple form of a linear combination of functions which depend on training instances in the original (lower dimensional) space.

The problem with SVM is that, since the generated model is defined on the basis of a subset of training examples, the evaluation of the decision function can be onerous when the training set size is big. This makes SVM quite unsuitable when a fast reaction of the classifying system is expected (e.g., in a real-time application) or when the utilized device has limited computational capabilities. Furthermore, the storage memory required by the model generated by an SVM is of an order linear with the cardinality of the training set, and this could be an issue when devices with a very limited memory should be used (e.g., mobile devices).

Among nondiscriminative classification methods, prototype-based methods are probably the easiest and less computationally demanding. In this method, a prototype is built for each class representing the common patterns of instances of the associated class. Specifically, once given a distance metric between instances, the basic idea is to build a prototype vector for each class, having the same dimension of the instances, in such a way to minimize the mean value of the distances between the prototype and each of the instances of the class. Once these prototypes are built, a new instance is classified with the class associated with the closest prototype. This method is theoretically founded when some statistical assumptions are made about the distribution of the examples and is empirically demonstrated to obtain good performances.

5.2. Prototype-Based Classification in Ghosts . Since our particular application is thought to work even with very limited computational resources, we have adopted a prototype-based classifier as the machine learning methodology. Specifically, for each player, a prototype of good and a prototype of bad ghost behavior are trained based on 17 features which have been considered informative to determine the nature of a ghost in the game.

In particular, the following features have been chosen: 8 features with binary values representing what was the initial position of the piece on the board among the eight possible ones, 5 features representing the moves of the piece during the game session (if it is the first piece that the player moved, if it is the second piece that the player moved, the number of backward, forward, and lateral moves already performed by that piece), and the remaining 4 features representing the piece's behavior when it has been under threat of being captured (how many times it has reacted by capturing the opponent piece, how many times it has escaped by moving to another board position, how many times it has remained

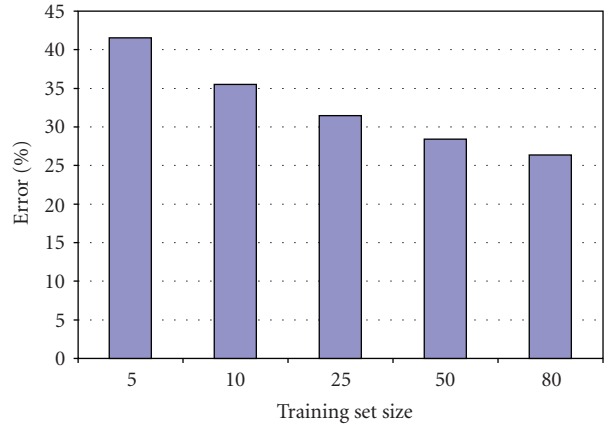


Figure 6: Averaged leave-one-out error obtained by the machine learning algorithm varying the number of game logs used for training.

on its position, and how many times it has moved from its square to threat another opponent's piece).

To build the prototype for a player, our algorithm needs first to collect data, that is, the training set, from previous game sessions with the same human player. For each of these sessions and for each piece a corresponding feature vector is built according to the criteria above which are based on the behavior of the piece in the game. The prototype for good (or bad) pieces is then determined as the average among the feature vectors representing good (or bad) pieces. More formally, given $G = \{g_1, \ldots, g_n\}$ the set of feature vectors for good pieces of a given player, and $B = \{b_1, \ldots, b_n\}$ the set of available feature vectors for bad pieces of the same player, the prototype vectors are computed as:

$$P_G = \frac{1}{n}\sum_{i=1}^{n} g_i, \qquad P_B = \frac{1}{n}\sum_{i=1}^{n} b_i. \tag{1}$$

Now, let be given a new feature vector f representing the profile of a piece of unknown type on the board; a badness score can hence be computed by using the normalized distance with respect to the player prototypes, that is,

$$s(f) = \frac{d(f, P_G) - d(f, P_B)}{d(f, P_G) + d(f, P_B)}, \tag{2}$$

where $d(x, y)$ is the Euclidean distance between vectors x and y. Note that the score is always a number between -1 (definitely good) and $+1$ (definitely bad).

On each move in the middle of a game session, the prediction of the type of the pieces on the board is performed in the following way. First, since the exact number of good and bad pieces (n_g and n_b, resp.) still on the board is a known information, a badness score for each of these pieces can be computed by utilizing (2). Then, the pieces are ranked based on this score and the n_b highest score pieces are predicted to be bad pieces.

The error committed in a prediction is computed as the number of bad pieces which are actually predicted as good ones. Needless to say, with the ranking method we used to discriminate between bad and good pieces, this error

also corresponds to the number of good pieces which are incorrectly predicted as bad.

6. Experimental Results

In this section, experimental results showing the effectiveness of our profiling methodology are reported. The experimental setting consisted in a mobile phone-based version of *Ghosts*. A user played a set of 81 games against other human players. Game logs of these matches, containing the initial state of the board and all the moves of the two players, have been stored during the games.

6.1. Evaluation of the Profile Construction in a Single Game Session. First, we studied the reliability of our profiler during a game session. Clearly, at the beginning of a game, the profile cannot be precise as some features of a piece may still be unavailable or underestimated. Thus, even if the models generated by our prototype-based machine learning algorithm were perfect, the correct classification of a piece may be difficult at the early stage of a game session, given the low informative degree of its profile. We can expect the prediction error to decrease whenever the profile becomes more and more informative and complete. However, this represents a desirable property as, in general, it is not so important to have a very low error when the game is at the very beginning, whereas it becomes crucial as the game session proceeds.

In particular, plots in Figure 5 give the percentage of error that affects our machine learning algorithm during ten single game sessions. Instantaneous values of the error percentage are provided at each move performed during the game session. Needless to say, the total number of moves to conclude a game session is variable; this is why charts in Figure 5 have different ranges of the values on the x-axis. Note that 50% of error means that two good (and two bad) ghosts out of four were wrongly labeled as bad (good), 25% represents the case with only one good (and one bad) ghost wrongly labeled, and 0% error is the case when the system provides a perfect prediction of the type of all the ghosts in the board. As anticipated, the precision of the predictor quickly improves as a game session continues and six out of the ten charts in Figure 5 achieve a percentage of 0% error during the second half of the game session.

6.2. Evaluation of the Machine Learning Algorithm. In a second set of experiments, we evaluated the performance of the machine learning algorithm. This has been done by estimating the probability to make errors in future match games. To this end, a leave-one-out procedure has been used. This measure, very common in the machine learning community for evaluating the expected performance of a classifier, provably gives a statistically unbiased estimate of the expected percentage of mistakes that a classifier will make. To compute these statistics, for each available game session log, a model has been generated by removing that game log from the set and training the prototype-based classifier on the remaining 80 game logs. Then, the mean error in the left out game log is computed as the proportion of piece type guess mistakes out of the total guesses during the game. Finally, all these mean errors have been collected and averaged to obtain a final estimate of the algorithm's performance. In particular, with the considered set of 81 game logs, the leave-one-out estimate resulted in a 26.3% error.

We also wanted to show how the prototype-based classifier improves its accuracy as the number of previously stored game logs increases. To this aim, we have exploited again the leave-one-out procedure but this time considering different training set sizes (consisting of 5, 10, 25, 50, 80 game logs). For each chosen training log set sizes, the leave-one-out procedure has been repeated each time considering a different left out game log; the averaged resulting error percentages are reported in Figure 6. As expected, with a larger size of the training set, the machine learning algorithm is able to build a more reliable prototype of the player's profile, thus improving the performance of the prediction system.

7. Conclusion and Future Work

The mass adoption of mobile phones in our society has transformed them from gadgets into commodities. The technical features of these devices have reached a quality level that makes them able to run multimedia applications. One of the most successful mobile applications is certainly represented by gaming and, in this paper, we discussed technical issues related to this context. In particular, we have proposed an original solution that improves the capability of the AI by allowing it to profile its human opponent and exploit her/his weaknesses. Our approach is particularly useful for imperfect information games and for games running on devices with limited computational capabilities (e.g., mobile phones), where a pure searching algorithm could not be employed; at the same time, it can be easily plugged into any standard AI or temporal difference learning-based algorithm to enhance its performance.

As a real testbed for our solution, we created a mobile phone-based game also considering the state-of-the-art in compatibility and connectivity technology available on today's phones. The game that we have chosen for our experiments is *Ghosts,* an imperfect information game that can be either played against another player or against the AI algorithm. Results gathered during our experimental evaluation demonstrate that our approach achieves the desired goal of an effective profiling of the player.

Intentionally, we have chosen a very simple machine learning technique for our experiments for two main reasons. First, we wanted to be sure that the chosen algorithm could be run, producing the desired goal, even on a device with limited computational capabilities. Second, our experiments were intended to prove the viability of our general method and they achieved this goal; in the future, more complex machine learning algorithms such as, for instance, SVM [28] can be utilized; also, an extended set of features to profile the opponent's behavior could be evaluated.

Another attractive future direction for this work would be that of applying recent machine learning methods which can deal with problems different from the classification one. In fact, it can be noted that the reduction of the *Ghosts* game to a classification problem is not the only one possible. Actually, the real need in the *Ghosts* game is to decide which the good and bad pieces are once we have a rank of the pieces based on their "badness;" then, ad hoc approaches to learn rankings and preferences (e.g., [29]) can be exploited to improve the performance of the system.

Finally, we also plan to apply our solution to more complex imperfect information games such as *Invisible Chess* and *Kriegspiel* which are heterodox chess variation in which players are not informed of their opponents position and moves [30, 31].

Acknowledgments

The authors feel indebted toward the anonymous referees, whose insightful comments and suggestions greatly helped them in improving the final version of this manuscript. Furthermore, their deep gratitude goes to Federico Giardina and Ivan Mazzarelli for their technical contribution in developing the digital version of the game and the experimental setup.

References

[1] "Homo Mobilis," The Economist, 2008, http://www.econ-omist.com/specialreports/displaystory.cfm?story_id=10950487/.

[2] "Cellular Mobile Gaming Grows to EUR 6 Billion in 2006," 2003, http://www.cellular.co.za/news_2003/101503-mobile_gaming_grows_to_eur_6_bil.htm.

[3] B. Gibson, "Casual Gamers and Female Gamers to Drive Mobile Games Revenues over the $10 Billion Mark by 2009," http://www.juniperresearch.com/shop/viewpressrelease.php?id=19&pr=16.

[4] ZDNet Personal Tech, "Mobile-game market to boom," 2004, http://news.zdnet.com/2100-1040_22-5355648.html.

[5] J. O. B. Soh and B. C. Y. Tan, "Mobile gaming," *Communications of the ACM*, vol. 51, no. 3, pp. 35–39, 2008.

[6] E. Koivisto, "Mobile games 2010," in *Proceedings of the International Conference on Game Research and Development (CyberGames '06)*, pp. 1–2, Murdoch University, Perth, Australia, December 2006.

[7] Symbian OS: the Open Mobile Operating System, http://www.symbian.com/.

[8] Qualcomm Brew, http://brew.qualcomm.com/brew/en/.

[9] "The Java ME Platform—the Most Ubiquitous Application Platform for Mobile Devices," http://java.sun.com/javame/index.jsp.

[10] Bluetooth.com | The Official Bluetooth® Technology Info Site, http://www.bluetooth.com/bluetooth.

[11] IEEE 802.11-2007, "IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2007.

[12] E. Gustafsson and A. Jonsson, "Always best connected," *IEEE Wireless Communications*, vol. 10, no. 1, pp. 49–55, 2003.

[13] V. Gazis, N. Houssos, N. Alonistioti, and L. Merakos, "On the complexity of "always best connected" in 4G mobile networks," in *Proceedings of the 58th IEEE Vehicular Technology Conference (VTC '03)*, vol. 4, pp. 2312–2316, Orlando, Fla, USA, October 2003.

[14] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 211–229, 1959.

[15] M. Campbell, "Knowledge discovery in deep blue," *Communications of the ACM*, vol. 42, no. 11, pp. 65–67, 1999.

[16] V. Allis, *A knowledge-based approach of connect-four the game is solved: white wins*, M.S. thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, 1998.

[17] R. Gasser, *Efficiently harnessing computational resources for exhaustive search*, Ph.D. thesis, Eidgenössische Technische Hochschule, Zurich, Switzerland, 1995.

[18] M. Buro, "The Othello match of the year: Takeshi Murakami vs. Logistello," *International Computer Chess Association Journal*, vol. 20, no. 3, pp. 189–193, 1997.

[19] D. Billings, "The first international RoShamBo programming competition," *International Computer Games Association Journal*, vol. 23, no. 1, pp. 42–50, 2000.

[20] D. Egnor, "Iocaine powder," *International Computer Games Association Journal*, vol. 23, no. 1, pp. 33–35, 2000.

[21] T. Mitchell, *Machine Learning*, McGraw Hill, New York, NY, USA, 1997.

[22] J. Schaeffer, "The games computers (and people) play," *Advances in Computers*, vol. 50, pp. 189–266, 2000.

[23] D. Billings, L. Pena, J. Schaeffer, and D. Szafron, "Using probabilistic knowledge and simulation to play poker," in *Proceedings of the 16th National Conference on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conference (AAAI '99)*, pp. 697–703, Orlando, Fla, USA, July 1999.

[24] M. L. Ginsberg, "GIB: steps toward an expert-level bridge-playing program," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI '99)*, pp. 584–589, Stockholm, Sweden, July-August 1999.

[25] B. Sheppard, "Mastering Scrabble," *IEEE Intelligent Systems*, vol. 14, no. 6, pp. 15–16, 1999.

[26] F. Aiolli and C. E. Palazzi, "Enhancing artificial intelligence in games by learning the opponent's playing style," in *Proceedings of the 1st IFIP Entertainment Computing Symposium on New Frontiers for Entertainment Computing, in Conjunction with the 20th IFIP World Computer Congress (ECS '08)*, pp. 1–10, Milan, Italy, September 2008.

[27] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence, Prentice Hall, Englewood Cliffs, NJ, USA, 1995.

[28] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, Berlin, Germany, 1995.

[29] F. Aiolli, "A preference model for structured supervised learning tasks," in *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM '05)*, pp. 557–560, Houston, Tex, USA, November 2005.

[30] A. Bud, D. Albrecht, A. Nicholson, and I. Zukerman, "Playing invisible chess with information-theoretic advisors," in *Proceedings of the AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, Stanford, Calif, USA, March 2001.

[31] P. Ciancarini, F. Dalla Libera, and F. Maran, "Decision making under uncertainty: a rational approach to Kriegspiel," *Advances in Computer Chess*, vol. 8, pp. 277–298, 1997.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 125714, 13 pages
doi:10.1155/2009/125714

Research Article

Breeding Terrains with Genetic Terrain Programming: The Evolution of Terrain Generators

Miguel Frade,[1] F. Fernandez de Vega,[2] and Carlos Cotta[3]

[1] *Escola Superior de Tecnologia e Gestão (ESTG), Instituto Politécnico de Leiria, Alto do Vieiro, 2411-901 Leiria, Portugal*
[2] *Centro Universitario de Mérida, Universidad de Extremadura, C/Sta Teresa de Jornet 38, 06800 Mérida, Spain*
[3] *ETSI Informática (3.2.49), Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain*

Correspondence should be addressed to Miguel Frade, mfrade@estg.ipleiria.pt

Received 14 June 2008; Accepted 24 November 2008

Recommended by Abdennour El Rhalibi

Although a number of terrain generation techniques have been proposed during the last few years, all of them have some key constraints. Modelling techniques depend highly upon designer's skills, time, and effort to obtain acceptable results, and cannot be used to automatically generate terrains. The simpler methods allow only a narrow variety of terrain types and offer little control on the outcome terrain. The Genetic Terrain Programming technique, based on evolutionary design with Genetic Programming, allows designers to evolve terrains according to their aesthetic feelings or desired features. This technique evolves Terrain Programmes (TPs) that are capable of generating a family of terrains—different terrains that consistently present the same morphological characteristics. This paper presents a study about the persistence of morphological characteristics of terrains generated with different resolutions by a given TP. Results show that it is possible to use low resolutions during the evolutionary phase without compromising the outcome, and that terrain macrofeatures are scale invariant.

1. Introduction

Artificial terrain generation techniques are an important facet of graphical applications that attempt to represent a real or an imaginary world. Among those applications are computer animation, architecture, virtual reality, and video games (for a more extensive list of examples check the Virtual Terrain Project, http://www.vterrain.org/Misc/Why.html). On the virtual terrain field, much of the research has been focused on how to accelerate the visualisation of large terrains to achieve interactive frame rates. As a result, many level of detail (LOD) algorithms have been developed [1–3], which increase rendering speeds by using simpler versions of the geometry for objects that have lesser visual importance such as those far away from the viewer [4]. This approach have been successfully applied on many video games, a field where artificial terrain generation techniques are probably more prominent.

On the other hand, terrain generation has received less attention in the literature. Fractal-based techniques are still the most prevalent, specially on video games, in spite of the several generation techniques existing today (see Section 2). This happens because of their speed, ease of implementation, and ability to create irregular shapes across an entire range of LODs. Nevertheless, these techniques allow only a confined variety of terrain types [2, 5], little control on the outcome, and are only focused on the generation of realistic terrains. Although this is important, it might prevent designers from achieving their goals when they attempt to represent an alien or an exotic looking terrain. Fractal-based techniques do not allow designers to express their full creativity or to evolve a terrain accordingly to their aesthetic feelings rather than realism. The terrain novelty might have a positive impact on a product's target audience and increase their interest. The Genetic Terrain Programming (GTP) technique [6] allows the evolution of Terrain Programmes (TPs) based on aesthetic evolutionary design with Genetic Programming (GP). For a specific resolution, it is known the ability of those TPs to generate a family of terrains—different terrains, but with coherent morphological features. This paper presents a set of experiments to study the perseverance of terrain morphological features across different resolutions. This is

a desired characteristic by video games' designers, as it will enable them to adapt the terrain to the required processing power, without recurring to additional algorithms. This property will also help to improve performance during the TPs' evolutionary phase.

Section 2 introduces some background about the traditional terrain generation techniques and their main constrains. It also presents an overview of evolutionary systems applied to terrain generation. Section 3 describes the GTP technique, the developed tool, and the achieved results. Finally, the conclusions and future work are presented on Section 4.

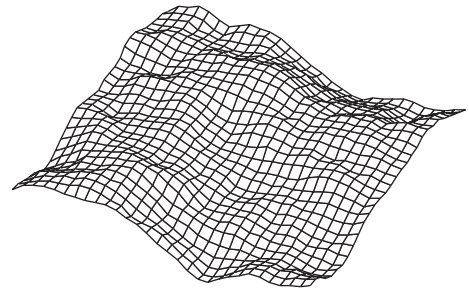


Figure 1: A discrete height map example.

2. Background

Artificial terrain generation has been addressed by several researchers for a long time, and therefore many techniques and algorithms have been developed. To establish a base line of comparison with real algorithms, Saunders [7] proposed the following list of traits that an ideal terrain generation algorithm should have

(i) low requirements of human input,

(ii) allow a high degree of human control,

(iii) to be completely intuitive to control,

(iv) produce models at arbitrary levels of detail,

(v) fast enough for real-time applications,

(vi) to be able to generate a wide variety of recognisable terrain types and features,

(vii) to be extensible to support new types of terrain.

Some of the listed characteristics are in tension with one another, such as low requirements of human input and high degree of human control. This means that all terrain generation techniques had to choose priorities and made some compromises regarding their traits. Another important attribute of a terrain generation technique is how it represents a terrain. The chosen data structure will influence the way the terrain is built, the available tools to manipulate it, and might affect also the terrains features that can be represented. Height maps are probably the most common method used to represent terrains, although other data structures exist. Formally, a height map is a scalar function of two variables, such that for every coordinate pair (x, y) corresponds an elevation value h, as shown in (1). In practice, a height map is a two-dimensional rectangular grid of height values, where the axis values are spaced with regular intervals valid over a finite domain (see Figure 1). The most common data structure to represent them is 2D arrays filled with the elevations values:

$$h = f(x, y). \tag{1}$$

The regular structure of height maps is their main advantage, since it allows the optimisation of operations such as rendering, collision detection, and path finding. The render of huge height maps in real time is now possible due to the creation of several continuous level of detail (CLOD) algorithms [1–3], which render highly visible areas of the terrain with detailed geometry, using progressively simpler geometry for more distant parts of the terrain. Collision detection is greatly simplified if one of the objects is a height map, because only a few surrounding triangles need to be checked for collision. A second advantage is the fact that height maps are compatible with grey-scale images (if the heights values are normalised). This means that image processing and computer vision techniques may be used to construct, modify, and analyse terrain models represented as height maps. For example, a height map can be stored, imported, or exported using an image file format, or a filter can be applied to smooth a rough terrain. Finally, Geographic Information Systems (GIS) use height maps to represent real-world terrain, which are commonly built using remote sensing techniques such as satellite imagery and land surveys. This is another advantage due to the significant amount of real-world terrain models available to work with.

The main limitation of height maps is the inability to represent structures where multiple heights exist for the same pair of coordinates. So, height maps are inherently unable to represent caves, overhangs, vertical surfaces, and other terrain structures in which multiple surfaces have the same horizontal coordinates. Fortunately, only a small percentage of natural terrain fall into this category, and this limitation can be overcome by using separate objects placed on top of the terrain model. A second disadvantage of height maps is that it has a finite uniform resolution, which means there is no simple way to handle a terrain with different local levels of details. If the resolution is chosen to match the average scale of the features in the terrain, then any finer-scale features will be simplified or eliminated. Conversely, if the resolution is chosen to be high enough to capture the fine-scale features, areas containing only coarse features will also be captured at this same high resolution, an undesirable waste of space and processing time. Ideally, a terrain representation for terrain generation would either be infinite in resolution, or else would adaptively increase its resolution to accommodate the addition of fine-scale details, rather than requiring a prior decision about resolution. A third disadvantage of height maps is its inadequacy to represent terrain on a planetary scale. Rectangular height maps do not map directly to spheroid objects; usually a two-pole spherical projection is used, and in those cases the density of height field points will be substantially greater in areas near the poles than at

those near the equator. For the purpose of our technique, the advantages of height maps outcome their shortcomings, though.

2.1. Traditional Terrain Generation Techniques. Traditional techniques for terrain generation can be categorised into three main groups: (1) measuring, (2) modelling, and (3) procedural. Next, we briefly review each of these techniques.

(1) Measuring techniques gather elevation data through real-world measurements, producing digital elevation models (http://rockyweb.cr.usgs.gov/nmpstds/demstds.html). These models are commonly built using remote sensing techniques such as satellite imagery and land surveys. One key advantage of measuring techniques lies in the fact that they produce highly realistic terrains with minimal human effort, although this comes at the expenses of the designer control. In fact, if the designer wants to express specific goals for the terrain's design and features, this approach may be very time-consuming since the designer may have to search extensively for real-world data that meet her targeted criteria.

(2) Modelling is by far the most flexible technique for terrain generation. A human artist models or sculpts the terrain morphology manually using a 3D modelling programme (e.g., Maya, 3D Studio (http://www.autodesk.com/fo-products), or Blender (http://www.blender.org)), or a specialised terrain editor programme (e.g., the editors that ship with video games like Unreal Tournament 2004 (http://www.mobygames.com/game/unreal-tournament-2004), SimCity 4 (http://simcity.ea.com/about/simcity4/overview.php), or SimEarth (http://www.mobygames.com/game/simearth-the-living-planet)). The way the terrain is built is different depending on the features provided by the chosen editor, but the general principle is the same. With this approach, the designer has unlimited control over the terrain design and features, but this might be also a disadvantage. By delegating most or all of the details up to the designer, this technique imposes high requirements on the designer in terms of time and effort. Also the realism of the resulting terrain is fully dependent on the designer's skills.

Finally, (3) procedural techniques are those in which the terrains are generated programmatically. This category can further be divided into physical, spectral synthesis, and fractal techniques.

Physically-based techniques simulate the real phenomena of terrain evolution trough effects of physical processes such as erosion by wind (http://www.weru.ksu.edu), water [8], thermal [9], or plate tectonics. These techniques generate highly realistic terrains, but require an in-depth knowledge of the physical laws to implement and use them effectively. Physically-based techniques are also very demanding in terms of processing power.

Another procedural approach is the spectral synthesis. This technique is based on the observation that fractional Brownian motion (fBm) noise has a well-defined power spectrum. So random frequency components can be easily calculated and then the inverse fast Fourier transform (FFT) can be computed to convert the frequency components into altitudes. The problem of using this technique for simulating real-world terrain is that it is statistically homogeneous and isotropic, two properties that real terrain does not share [9]. Furthermore, it does not allow much control on the outcome of terrains' features.

Self-similarity is the key concept behind any fractal technique. An object is said to be self-similar when magnified subsets of the object look like (or identical to) the whole and to each other [10]. This allows the use of fractals to generate terrain which still looks like terrain, regardless of the scale in which it is displayed [11]. Every time these algorithms are executed they generate a different terrain due to the incorporated randomness. This class of algorithms is the favourite one by game's designers, mainly due to their speed and simplicity of implementation. There are several tools available that are predominantly based on fractal algorithms, such as Terragen (which is a hybrid fractal/modelling tool) (http://www.planetside.co.uk/terragen) and GenSurf (a mapping tool for Quake 3 Arena video game) (http://tarot.telefragged.com/gensurf). However, generated terrains by this techniques are easily recognised because of the self-similarity characteristic of fractal algorithms. Besides, not all terrain types exhibit the self-similar property across all scales. For example, both photos from Figure 2 are from Death Valley (Calif, USA), but seen at very different scales. On Figure 2(a) is a close-up of cracked dried mud in a creek from the Death Valley and on Figure 2(b) is a satellite image of the same region. As is easily verified, in this case there is no self-similarity between the two scales of these terrain photos. Although these algorithms present some parameters that can be tweaked to control, for example, the roughness, the designer does not have control on the resulting terrain features.

2.2. Evolutionary Terrain Generation Techniques. Evolutionary algorithms (EAs) are a kind of bioinspired algorithms that apply Darwin's theory [12] of natural evolution of the species, where living organisms are rewarded through their continued survival and the propagation of its own genes to its successors. There are four main classes of EAs: genetic algorithms (GAs) [13], evolutionary strategies [14], GP [15], and evolutionary programming [16]. Evolutionary algorithms can be seen as search techniques [17]. They are able to achieve good solutions to many types of problems, thanks to their flexibility and adaptability to different search scenarios. This characteristic is the key factor of success in such diverse fields as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, and chemistry. Apart from their use as optimisers, evolutionary algorithms have also been used as an experimental framework to validate theories about biological evolution and natural selection, particularly through work in the field of artificial life [18].

Evolutionary design is a branch of evolutionary computation which has its roots in three different disciplines: computer science, evolutionary biology, and design. Evolutionary design has taken place in many different areas over the last decade. Designers have optimised selected parts of their designs using evolution; artists have used evolution to generate aesthetically pleasing forms; architects have evolved

(a) (b)

Figure 2: Images of Death Valley: (a) "cracked mud on the way to the borax haystacks," by *redteam*, Creative Commons license, (b) a satellite image from NASA (public domain). On this example, there is no self-similarity between the two scales of this region.

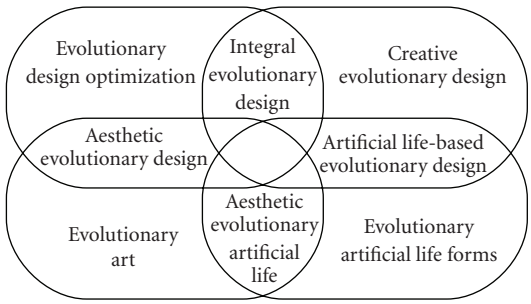


Figure 3: Evolutionary design categories.

new building plans from scratch; computer scientists have evolved morphologies and control systems of artificial life. Evolutionary design can be divided into four main categories [19]: evolutionary design optimisation, creative evolutionary design, evolutionary art, and evolutionary artificial life forms. However, some author's work may be included in two or more categories creating four overlapping subcategories shown in Figure 3.

Evolutionary art systems are similar in many ways. They all generate new forms or images from the ground up (random initial populations); they rely upon a human evaluator to set the fitness value of an individual based on subjective evaluation, such as aesthetic appeal; population sizes are very small to avoid user's fatigue and allow a quick evaluation, and user interfaces usually present a grid on the screen with the current population individuals, allowing the user to rank them. However, they differ on their phenotype representations [20].

GP has been the most fruitful evolutionary algorithm applied to evolve images interactively. Karl Sims used GP to create and evolve computer graphics by mathematical equations. The equations are used to calculate each pixel [21]. He created several graphic art pieces including *Panspermia* and *Primordial Dance,* and also allowed visitors to interact with his interactive art system at art shows and exhibitions. His *Galapagos* (http://www.karlsims.com/galapagos/index.html) is an L-system-based interactive evolutionary computation (IEC) system that allows visitors to create their own graphic art through their interaction.

Tatsuo Unemi developed Simulated Breeding ART (SBART) [22, 23], an IEC graphics system open to public. *SBART* uses GP to create mathematical equations for calculating each pixel value and its (x, y) coordinates. As GP nodes, *SBART* assigns the four arithmetic fundamental operators $(+, -, \times,$ and $\div)$, *power*, *sqrt*, sin, cos, log, exp, min, and max. The terminal nodes are constants and variables. Three values at each pixel are calculated using one generated mathematical equation by assuming that the constants are 3D vectors consisting of three real numbers, and the variables are a 3D tuple consisting of $(x, y, 0)$. The three calculated values are regarded as members of a vector (hue, lightness, and saturation), and are transformed to RGB values for each pixel. These three values are normalised to values in $[-1, 1]$ using a saw-like function. The *SBART*'s functions were expanded to create a collage [24]. A human user selects preferred 2D images from 20 displayed images at each generation, and the system creates the next 20 offspring. Sometimes exporting/importing parents among multiple *SBART* instances is allowed. This operation is iterated until the user obtains a satisfactory image.

In Neuro Evolutionary Art (NEvAr) [25], of Machado and Cardoso, the function set is composed mainly of simple functions such as arithmetic, trigonometric, and logic operations. The terminal set is composed of a set of variables x, y, and random constants. The phenotype (image) is generated by evaluating the genotype for each (x, y) pair belonging to the image. In order to produce colour images, NEvAr resorts to a special kind of terminal that returns a different value depending on the colour channel—red, green, or blue—that is being processed. This tool focuses on the reuse of useful individuals, which are stored in an image database and led to the development of automatic seeding procedures.

To the best of our knowledge, Ong et al. [26] were the first authors to propose an evolutionary approach to generate terrains. They proposed an evolutionary design optimisation technique to generate terrains by applying genetic algorithms to transform height maps in order to conform them to the required features. Their approach breaks down the terrain generation process into two stages: the terrain silhouette generation phase, and the terrain height map generation phase. The input to the first phase is a rough 2D map

laying out the geography of the desired terrain that can be randomly generated or specified by the designer. This map is processed by the first phase to remove any unnaturally straight edges and then fed to the second phase, along with a database of preselected height map samples representative of the different terrain types. The second phase searches for an optimal arrangement of elevation data from the database that approximates the map generated in the first phase. Since the height map generation algorithm is inherently random, the terrains generated from two separate runs of the algorithm will not be the same, even if they use the same map.

We proposed a new technique, based on aesthetic evolutionary design, designated GTP [6]. Our approach consists on the combination of interactive evolutionary art systems with GP to evolve mathematical expressions, designated TPs, to generate artificial terrains as height maps. GTP relies on GP as evolutionary algorithm, which creates mathematical expressions as solutions (further details are presented on Section 3).

2.3. Genetic Programming. Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. More precisely, GP is a systematic domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done. In GP, a population of computer programmes is evolved. Generation by generation, a population of programmes is stochastically transformed into new, hopefully better, populations of programmes [27]. Due to its heuristic nature, GP can never guarantee results. However, it has been used successfully in many areas, such as [17] artificial life, robots and autonomous agents, financial trading, neural networks, art, image and signal processing, prediction and classification, and optimisation.

Algorithm 1 shows the basic steps of GP. The generated programmes are run for evaluation (Line 3) and compared with some ideal. This comparison is quantified to give a numeric value called fitness. The best programmes are chosen to breed (Line 4) and produce new programmes for the next generation (Line 5). The primary genetic operators used to create new programmes from existing ones are the following:

(i) *crossover*: the creation of a child programme by combining randomly chosen parts from two selected parent programmes,

(ii) *mutation*: the creation of a new child programme by randomly altering a randomly chosen part of a selected parent programme.

In GP, programmes are usually expressed as trees rather than as lines of code. For example, Figure 4 shows the tree representation of the programme $\max(x + x, x + 3 * y)$. The variables and constants in the programme (x, y, and 3) are leaves of the tree, or terminals in GP terminology. The arithmetic operations ($+$, $*$, and max) are internal nodes called functions. The sets of allowed functions and terminals together form the primitive set of a GP system.

For those who wish to learn more about GP, the book *A Field Guide to Genetic Programming* from Poli et al. [27] has a very good introduction to GP. A thoroughly analysis on this topic is provided on the book *Genetic Programming—On the Programming of Computers by Means of Natural Selection* by Koza [15], the main proponent of GP who has pioneered the application of Genetic Programming in various complex optimisation and search problems.

3. Genetic Terrain Programming

Current terrain generation techniques have their own advantages and disadvantages, as detailed in Section 2. Notwithstanding the importance of real-looking terrains, none of the existing methods focused on generating terrains accordingly to designers' aesthetic appeal. The main goals of GTP are to address the weaknesses of existing methods, allowing also the generation of aesthetic terrains. Thus, providing a *better* way of generating virtual terrains for a broad range of applications, with a special emphasis on video games.

In light of the idealised terrain generator, the goals of GTP are (in order of decreasing importance) as follows:

(1) capable of generating diverse features and terrain types, both aesthetic and realistic,

(2) extensibility,

(3) intuitive to control,

(4) automated generation with arbitrary resolution,

(5) low requirements of human input.

To achieve these goals, we use aesthetic evolutionary design with GP, where the phenotypes are terrains represented as height maps. This approach consists of a guided evolution, through interactive evolution, according to a specific desired terrain feature or aesthetic appeal. The extensibility and ability to generate diverse features and terrain types are assured by the GP. The diversity of solutions is directly dependent on the GP terminal and function sets. So, the extensibility feature can be easily achieved by adding new functions and terminals. The designer will guide the terrains evolution, performing this way the control of the outcome, by selecting which ones he prefers for his specific goals. Consequently, the software tool will be easy and intuitive to use with low input requirements. The outcome of the interactive evolution will be TPs, which are mathematical expressions with incorporated randomness. Those TPs can be used, like a procedural technique, to automatically generate different terrains with different resolutions and the same consistent features.

3.1. Method. The initial population is created randomly, with trees depth size limited initially to 6 and a fixed population size of 12. The number of generations is decided by the designer, who can stop the algorithm at any time. The designer can select one or two individuals to create the next population, and the genetic operators used depend

1: Randomly create an initial population of programs from the available primitives
2: **repeat**
3: Execute each program and ascertain its fitness
4: Select one or two program(s) from the population with a probability based on fitness to participate in genetic operations
5: Create new individual program(s) by applying genetic operations with specified probabilities
6: **until** an acceptable solution is found or some other stopping condition is met (e.g., a maximum number of generations is reached)
7: **return** the best-so-far individual

Algorithm 1: Genetic programming basic algorithm [27].

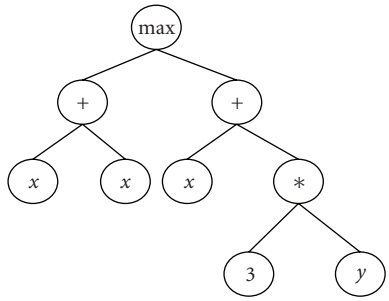


Figure 4: GP tree representation of $\max(x + x, x + 3 * y)$.

Table 1: GP functions.

Name	Description
plus(h_1, h_2) minus(h_1, h_2) multiply(h_1, h_2)	Arithmetical functions
sin(h) cos(h) tan(h) atan(h)	Trigonometric functions
myLog(h)	Returns 0 if $h = 0$ and $\log(abs(h))$ otherwise
myPower(h_1, h_2)	Returns 0 if $h_1^{h_2}$ is NaN or Inf, or has imaginary part, otherwise returns $h_1^{h_2}$
myDivide(h_1, h_2)	Returns h_1 if $h_2 = 0$ and $h_1 \div h_2$ otherwise
myMod(h_1, h_2)	Returns 0 if $h_2 = 0$ and $\text{mod}(h_1, h_2)$ otherwise
mySqrt(h)	Returns sqrt ($abs(h)$)
negative(h)	Returns $-h$
FFT(h)	2D discrete Fourier transform
smooth(h)	Circular averaging filter with $r = 5$
gradient $X(h)$ gradient $Y(h)$	Returns the gradient (dh/dx or dh/dy) of a height map h. Spacing between points is assumed to be 1

upon the number of selected individuals. If one individual is selected, only the mutation operator will be used. In case the designer chooses to select two individuals, both the standard crossover and mutation operators [15] will be applied (see Table 3). Like in others IEC systems, the fitness function relies exclusively on designers' decision, either based on his aesthetic appeal or on desired features.

According to Bentley [20], the designer is likely to score individuals highly inconsistently as he might adapt his requirements along with the evolved results. So, the continuous generation of new forms based on the fittest from the previous generation is essential. Consequently, nonconvergence of the EA is a requirement. Evolutionary art systems do not usually use crossover operators on their algorithms, because EAs are used as a continuous novelty generators, not as optimisers. Therefore, in our algorithm, the use of two individuals for breeding the next generation should be limited. The extensive use of crossover operator will converge the population to a single solution, leading to the loss of diversity and limiting the designer to explore further forms.

Each GP individual is a tree composed by functions, listed in Table 1, and height maps as terminals (see Table 2). Most terminals depend upon a random ephemeral constant (REC) to define some characteristics, such as the spectrum value of *fftGen*. All terminals have some form of randomness, which means that consecutive calls of the same terminal will always generate a slightly different height map. This is a desired characteristic because we want to be able to create different terrains by each TP, but we want them to share the same features. All terminals generate surfaces that are proportional to the side size of the height map. This ensures that the terrain features of a TP are scale invariant. Figure 5 shows height maps of size 30×30 generated by terminals *fftGen*, *gauss*, *step*, and *sphere*. Except *rand*, all terminals depend upon a random ephemeral constant (REC) to define some characteristics. REC is a special terminal that creates values randomly which remain constant until it disappears from the GP tree due to the use of a genetic operators. Figure 6 presents an example of a TP in tree form with two REC values represented in grey ellipses within the terminals.

While in [23, 24], the mathematical equations are used to calculate both the pixel value and its coordinates, in *GenTP* only the height will be calculated. The (x, y) coordinates will

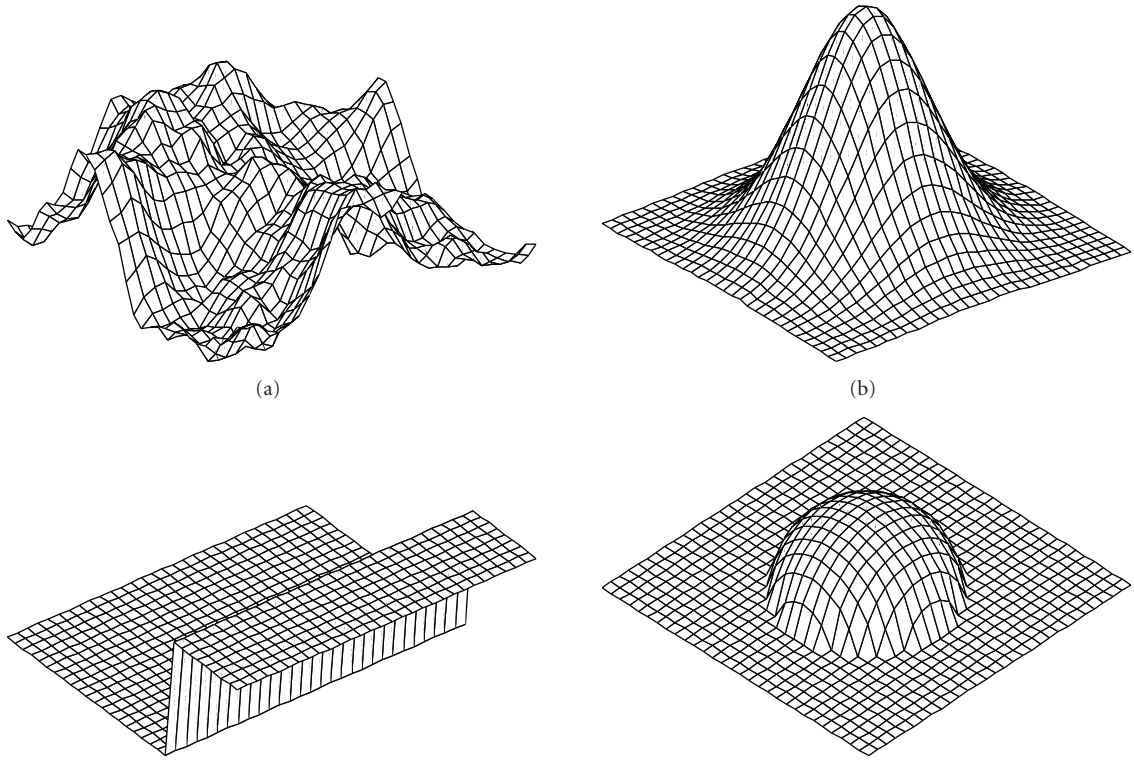


Figure 5: Examples of height maps terminals *fftGen*, *gauss*, *step*, and *sphere*.

Table 2: GP terminals.

Name	Description
rand	Map with random heights between 0 and 1
fftGen	Spectral synthesis-based height map, whose spectrum depends on an REC: $1/(f^{\mathrm{REC}})$
gauss	Gaussian bell shape height map, whose wideness depends on an REC
plane	Flat inclined plane height map whose orientation depends on an REC within 8 values
step	Step shape height map whose orientation depends on an REC within 4 values
sphere	Semisphere height map whose centre location is random and the radius depends on an REC

be dictated by the matrix position occupied by the height value.

In GTP, the 12 individuals of the population must be executed during the interactive evolutionary phase to be evaluated by a designer, which will choose the TPs for the next generation. This means that using high resolution on this phase will consume more time, and the application will be less responsive. An additional variable (that will be denoted as *s*) is introduced in all terminals to control the resolution during the TP execution. The axis values in the terminals' functions are discrete with regular intervals, and the variable *s* controls the spacing between axis values by specifying the height map grid size, which covers a predefined area. The greater is the *s* value, the lesser is the distance between each grid point and greater is the resolution.

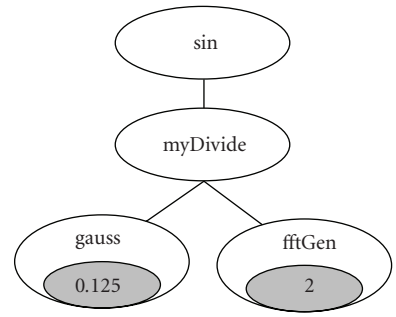


Figure 6: Example of a GP tree individual with two RECs (in grey ellipses).

3.2. GenTP Tool. To implement this new technique, we developed Generator of Terrain Programs (GenTP) [28], an application developed with GPLAB (http://gplab.sourceforge.net/), an open source GP toolbox for MATLAB

Table 3: Parameters for a GTP run.

Objective	Generate realistic or aesthetic terrains
Function set	Functions from Table 1, all operating on matrices with float numbers
Terminal set	Terminals from Table 2 chosen randomly
Selection and fitness	Decided by the designer accordingly to desired terrain features or aesthetic appeal
Population	Fixed size with 12 individuals; initial depth limit 6; no tree size limits; random initialisation
Parameters	If 2 individuals are selected: 90% subtree crossover and 10% mutation; if just one individual is selected: 50% mutation (without crossover)
Operators	Three mutation operators are used with equal probability: (1) *Replace mutation* where a random node is replaced with a new random tree generated by the grow method; (2) *Shrink mutation* where a random subtree (S) is chosen from the parent tree and replaced by a random subtree of S; (3) *Swap mutation* where two random subtrees are chosen from the parent tree and swapped, whenever possible the two subtrees do not intersect. One crossover operator is used: *subtree crossover* where random nodes are chosen from both parent trees, and the respective branches are swapped creating two offsprings
Termination	Can be stopped at any time by the designer, the "best" individual is chosen by the designer

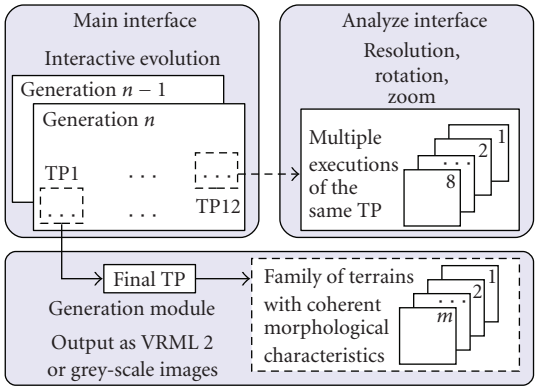


Figure 7: *GenTP*'s functional modules.

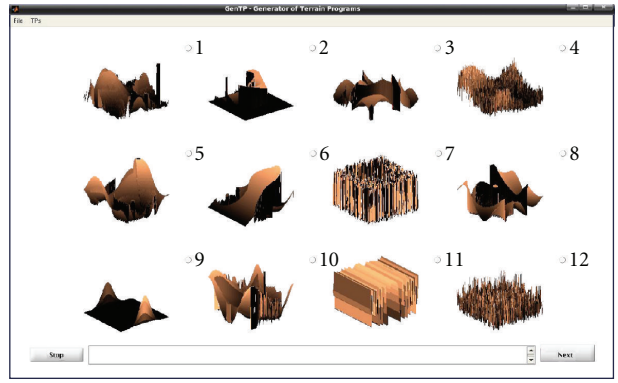


Figure 8: *GenTP* main user's interface.

(http://www.mathworks.com/). *GenTP* has three functional modules (depicted in Figure 7):

(i) interactive evolution,

(ii) analyse,

(iii) generation.

The interactive evolution module is where the GP is implemented, and the designer chooses the desired terrains for the next generation, for the analyse or generation modules. Figure 8 shows the graphical user interface (GUI) of *GenTP*'s main interface, which is the visible part of the interactive evolution module. The 12 individuals of current population are represented as 3D surfaces and displayed in a 3×4 grid. Each TP is evaluated to produce a height map of size 100×100 to be displayed to the designer. The height map size can be changed, but should be kept small otherwise it might have a negative impact in the tool responsiveness. We will return to this point later on Section 3.3.

The *GenTP* main GUI allows a designer to select one or two individuals to create the next population generation. The number of selected TPs will influence their evolution. If just one TP is selected—only the mutation operator will be applied—the next generation will present few variations from the selected individual, and the TP will evolve slowly. On the other hand, if the designer opts to select two individuals, the next generation will present more diversity and the evolved TPs can change their look more dramatically.

On the bottom of the main GUI, the designer can see the TP mathematical expression that generated the selected terrain and save it on a text file or database. This option will allow the integration of TPs, as a procedural technique, to produce terrains for example on a video game.

Although the main interface serves its purpose, some times it is difficult to see all TP features due the display angle used to show the generated terrain. It may be also difficult to inspect small details of a generated terrain, and it is not possible to test the TP's features perseverance across multiple executions. For these reasons, it might be difficult for the designer to chose the TPs for the next generation. To solve these limitations, the analyse module was added to our application. This new functionality opens a new window, see

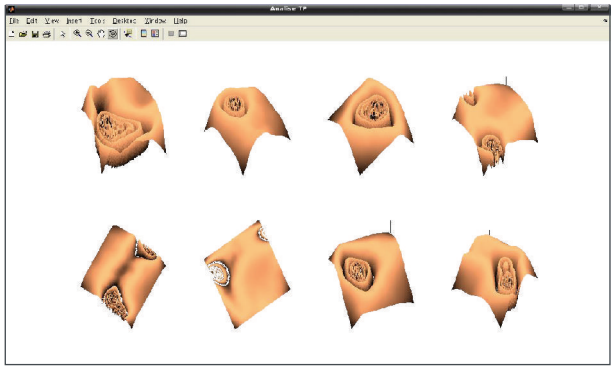
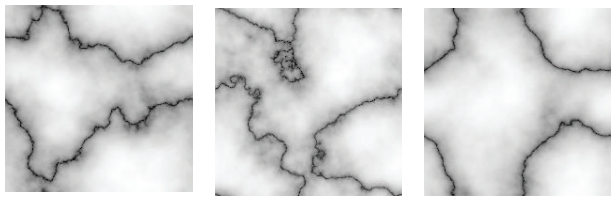
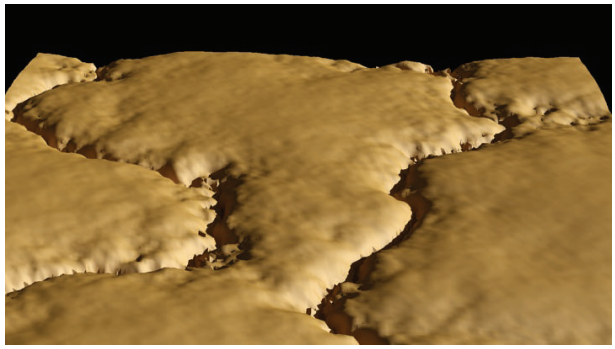


Figure 9: *GenTP* analyse user's interface.



(a)



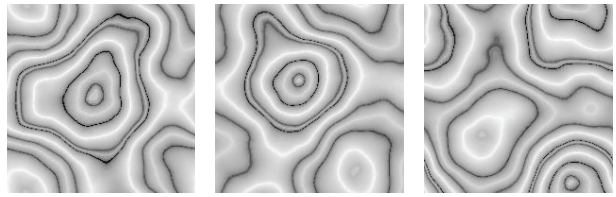
(b)

Figure 10: Family of terrains from TP in (2): (a) represented as grey-scale images, (b) rendered with 3D studio.

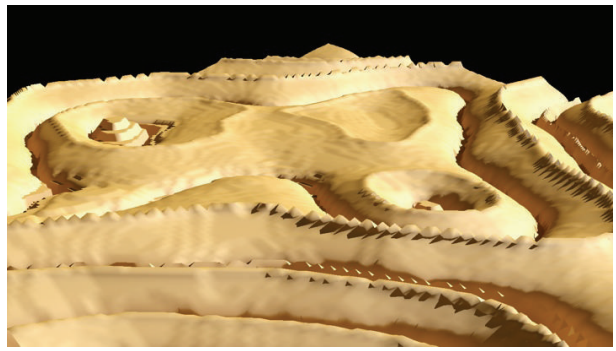
Figure 9, and performs 8 consecutive executions of the TP selected from the main interface. To allow a more detailed analysis of the TP characteristics, this interface allows the designer to rotate, zoom, and change the terrains resolution. This way the designer has more information about a TP to decide if it will be selected, or not, for the next generation.

When the designer achieves the desired TP, then he can save it in a file, or can pass it to the generator module. This module is responsible for the generation of height maps, as many as desired, from the selected TP. Those height maps can be saved as VRML 2.0 permitting its import from other applications, such as 3D modelling and render tools.

3.3. Experimental Results. Our previous work [6] has shown the ability of our technique to evolve TPs capable of generating a family of height maps (different terrains that share the same morphological characteristics). Figure 10 shows terrains from a TP evolved with river beds in mind,



(a)



(b)

Figure 11: Family of terrains from TP in (3): (a) represented as grey-scale images, (b) rendered with 3D studio.

and Figure 11 shows terrains from a TP evolved to obtain a family of aesthetic terrains. All these results were obtained with a fixed resolution of 200×200.

An experiment was conducted to test the perseverance of terrain features across several resolutions and the consequent impact in generation time on our evolutionary tool [29]. A set of TPs was chosen to generate terrains with grid sizes from 50 to 450 with increments of 50. To perform these tests, it was necessary to modify the terminals in order to include the variable s to specify the resulting height map size.

Figures 12, 13, 14, and 15 present the results of TP's shown in (4), (5), (6), and (7) at three different resolutions with grid sizes of 50×50, 150×150, and 450×450.

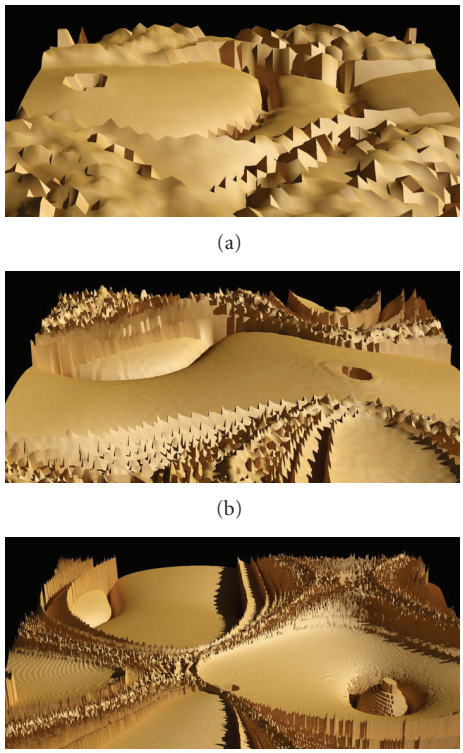
$$\begin{aligned} \mathrm{TP} = {} & \mathrm{myLog}\,(\mathrm{myLog}\,(\cos(\mathrm{minus}\,(\mathit{fftGen}\,(2.00), \\ & \mathit{fftGen}\,(3.75))))) \end{aligned} \tag{2}$$

$$\begin{aligned} \mathrm{TP} = {} & \mathrm{myLog}\,(\mathrm{myLog}\,(\mathrm{myLog}(\mathrm{myLog}\,(\mathrm{myLog}\,(\\ & \mathrm{myLog}(\mathit{fftGen}\,(3.00))))))) \end{aligned} \tag{3}$$

$$\begin{aligned} \mathrm{TP} = {} & \mathrm{myLog}\,(\mathrm{myLog}\,(\mathrm{myMod}\,(\mathrm{myLog}\,(\mathit{fftGen}\,(s, 3.75)), \\ & \mathrm{myLog}\,(\mathrm{myLog}\,(\mathit{fftGen}\,(s, 4.25)))))) \end{aligned} \tag{4}$$

$$\begin{aligned} \mathrm{TP} = {} & \mathrm{myPower}\,(\cos(\mathrm{myDivide}\,(\mathrm{myLog}\,(\mathrm{smooth}(\\ & \mathit{fftGen}(s, 2.75))),\ \mathrm{myMod}\ (\sin(\mathit{fftGen}\,(s, 0.50)), \\ & \mathrm{myDivide}\,(\mathrm{myLog}\,(\mathrm{smooth}\,(\mathit{fftGen}\,(s, 2.75))), \\ & \mathrm{myMod}\,((\sin(\mathit{fftGen}\,(s, 0.50))), \mathit{fftGen}\,(s, 2.25)))))) \end{aligned} \tag{5}$$

$$\begin{aligned} \mathrm{TP} = {} & \mathrm{multiply}(\ \sin(\mathit{fftGen}(s, 3.00)), \mathrm{smooth}(\\ & \mathrm{multiply}(\sin(\cos(\sin(\cos(\mathrm{multiply}(\mathit{fftGen}\,(s, 1.75), \\ & \mathit{fftGen}\,(s, 0.75)))))), \mathit{fftGen}\,(s, 0.50)))) \end{aligned} \tag{6}$$

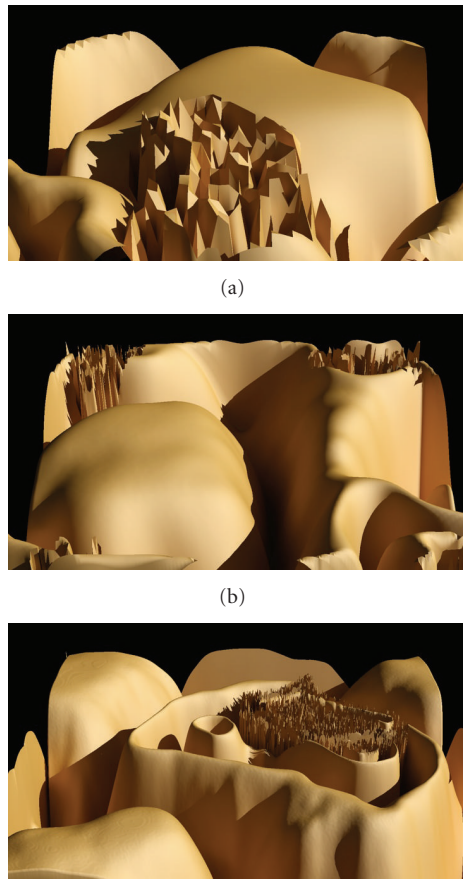


(a)

(b)

(c)

Figure 12: Exotic terrain generated by TP in (4), with resolutions 50×50, 150×150, and 450×450.



(a)

(b)

(c)

Figure 13: Exotic terrains generated by TP in (5), with resolutions 50×50, 150×150, and 450×450.

$$\begin{aligned} \text{TP} = \text{plus}\,(&\textit{fftGen}\,(s, 2.00),\\ &\text{smooth}\,(\text{myMod}\,(\text{gauss}\,(s, 0.75),\\ &\cos(\textit{fftGen}\,(s, 1.00))))) \end{aligned} \tag{7}$$

In these experiments, all TPs have preserved their main features independently of the chosen grid size. Due to the inherent randomness embedded in terminals, consecutive calls of the same TP will always generate a slightly different height map. This is a desired characteristic, that can be controlled by fixating the random number seed. Note that when generating terrains at different resolutions, the amount of necessary random numbers will vary accordingly with the chosen resolution. This explains the differences from terrains at different resolutions generated by the same TP.

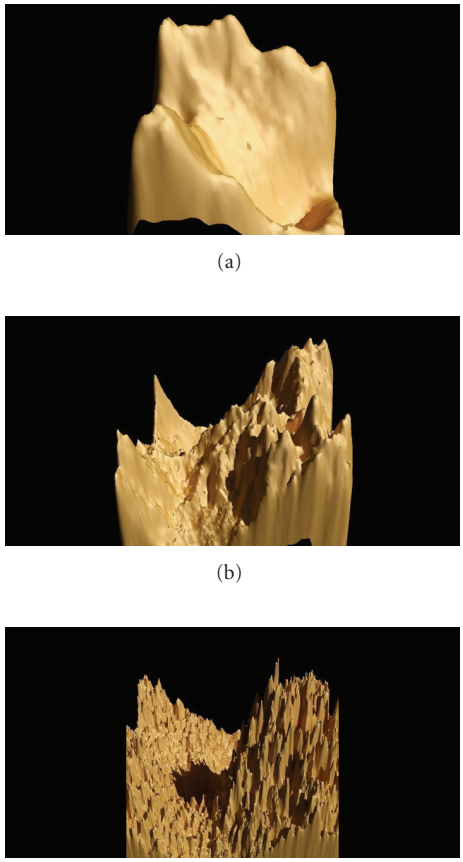
Figure 16 shows the average time of 10 executions of each TP at each grid size on a Pentium Core 2 Duo at 1.66 GHz with 2 GB of RAM. As expected, the generation time increases at a quadratic pace with the increase of the number of grid points, for example, for TP 7 from 18.4 millisecond at 50×50 to 1066.0 millisecond at 450×450. The generation time also increased, as anticipated, with the number of TP's nodes.

The values presented on Figure 16 are the times for generating each individual. The time to generate the entire population must be multiplied by the population size. For TP 5 (with 17 nodes) with a resolution of 450×450, each individual takes 3.122 seconds. So, to generate an entire population of 12 indviduals, 37.464 seconds will be needed. A delay of this magnitude is not negligible and will have a negative impact on the response time of interactive application such as our tool. According to Card et al. [30] and Testa and Dearie [31],

(i) 0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result;

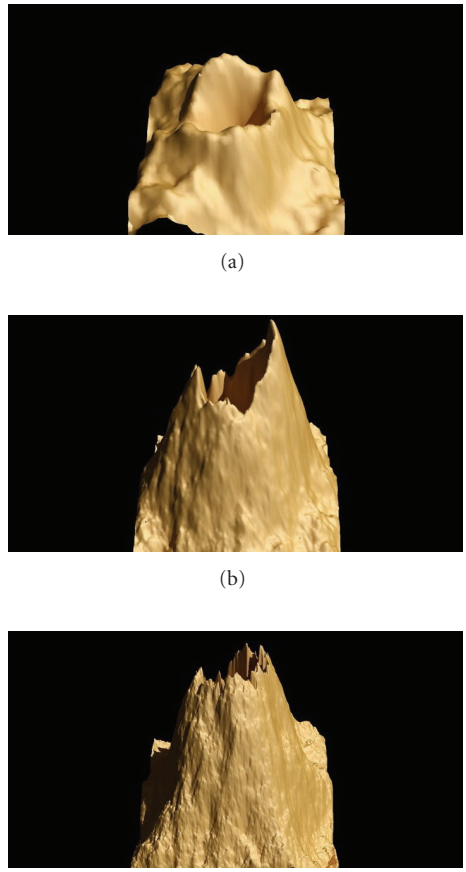
(ii) 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data;

(iii) 10 seconds are about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.



(a) (b) (c)

Figure 14: Mountains generated by TP in (6), with resolutions 50 × 50, 150 × 150, and 450 × 450.



(a) (b) (c)

Figure 15: Volcanoes generated by TP in (7), with resolutions 50 × 50, 150 × 150, and 450 × 450.

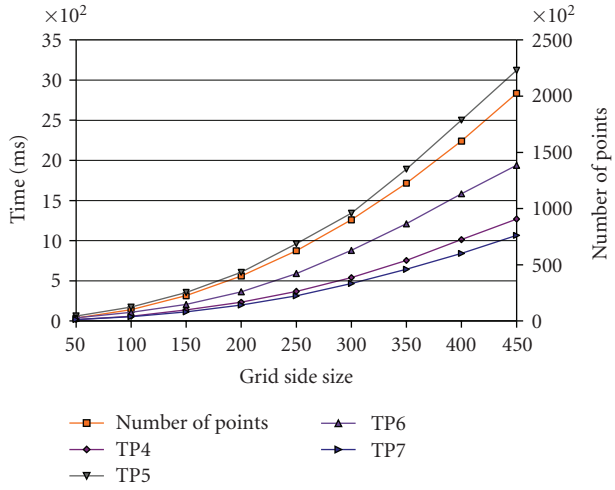
The response times should be as fast as possible to keep designer's attention focused on the application. Our goal is to keep generation times for the entire population around 1 second and never exceeding 10 seconds. The generation time depends on the chosen resolution and on each individual number of nodes, which tends to increase with the number of GP generations, a phenomenon known as bloat [27]. So, from the responsiveness point of view, the use of the lowest resolutions for the evolutionary phase is better. However, if the used resolution is too low the output might not represent all the terrain features, specially small details, and force the designer to use the analyse window more often. This will increase the time needed by the designer to choose the best terrain at each generation and consequently the overall time to achieve the desired terrain. A compromise must be made between the terrain resolution for the evolutionary phase and the application responsiveness. From our set of experiences we found the grid size of 100 × 100 to be the best settlement. Short generation times will be also advantageous for the future implementation of automated terrain evolution.

Once the TP is designed it can be used on a video game like any other procedural technique. On this case, the time for generating a new terrain is negligible, given that it will be generated before the game begins, during the "load" period.

4. Conclusions and Future Work

This paper presented the GTP technique which allows the evolution of TPs to produce terrains accordingly to designers' aesthetic feelings or desired features. Through a series of experiments we have shown that the feature persistence is independent of the chosen resolution. This means that during the evolutionary phase low resolutions can be used without compromising the result. Consequently less time will be required for our evolutionary tool, enabling it to be more responsive, which is an important characteristic on interactive tools. Additionally, the resulting TPs can be incorporated in video games, like any other procedural technique, to generate terrains, with the same features, independently of the chosen resolution.

Some game publishers require that all players have the same game "experience" if they make the same choices. They want to measure or obtain quality control both on the user experience side as well as on the development and testing end. This requirement seems to contradict our goal of achieving different terrains with the same TP. However, if a TP is incorporated on a video game as a procedural technique, our technique can deliver two levels of control regarding randomness. First, a specific TP will always



Grid size	Time (ms)			
	1 (TP 4)	2 (TP 5)	3 (TP 6)	4 (TP 7)
50	18.5	61.4	37.7	18.4
100	59.6	171.9	106.9	53.9
150	136.4	355.1	205.3	113.1
200	233.2	605.7	362.9	198.6
250	368.3	962.5	590.1	312.6
300	539.8	1342.3	878.7	466.8
350	753.5	1889.8	1210.3	643.4
400	1012.7	2501.1	1583.6	841.3
450	1269.8	3122.1	1939.1	1066.0
GP nodes	8	17	13	7

Figure 16: Terrain generation times versus grid sizes.

generate terrains with the same features, this means that in spite of the present randomness those terrains are similar and not completely random. Second, if full control over the final terrain is required the seed for the random number generator can be kept the same across separate runs of the TP, allowing the same terrain to be regenerated as many times as desired.

The TPs' scale invariance showed in our results preludes the implementation of a zoom feature. Fixating the random number generator seed is not enough to implement this feature due to the variation of the amount of necessary random numbers accordingly with the zoom. Besides, some terminals, like *rand* and *fftGen*, are not based on continuous functions. Other improvement to our technique will be the composition of a terrain through the use of several TPs, previously stored on a database, where the generated terrains will be joined on a credibly and smooth way. This will allow the control over localised terrain features. We also want to implement the GTP technique as a Blender plug-in to increase both the flexibility and the target audience for our technique.

The search for a terrain with a specific feature might be a tiresome endeavour on interactive evolutionary applications [20]. Therefore, it is desirable to automate, as much as possible, the task of evaluating TPs to avoid designers fatigue. We plan to develop fitness functions to perform the automatic evaluation of TPs accordingly to a feature by means of statistic measures. Another future work will be the inclusion of more features in our technique in order to generate full landscapes including textures, vegetation, and buildings.

Acknowledgment

The third author acknowledges the support of MICINN under project TIN2008-05941.

References

[1] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes," in *Proceedings of the 8th IEEE Visualization Conference (VIS '97)*, pp. 81–88, IEEE Computer Society Press, Phoenix, Ariz, USA, October 1997.

[2] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 769–776, 2004.

[3] S. Li, X. Liu, and E. Wu, "Feature-based visibility-driven CLOD for terrain," in *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pp. 313–322, IEEE Computer Society Press, Canmore, Canada, October 2003.

[4] J. H. Clark, "Hierarchical geometric models for visible-surface algorithms," in *Proceedings of the 3rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '76)*, p. 267, Philadelphia, Pa, USA, July 1976.

[5] B. Pelton and D. Atkinson, "Flexible generation and lightweight view-dependent rendering of terrain," Tech. Rep. COEN-2003-01-22, Department of Computer Engineering, Santa Clara University, Santa Clara, Calif, USA, 2003.

[6] M. Frade, F. F. de Vega, and C. Cotta, "Modelling video games' landscapes by means of genetic terrain programming—a new approach for improving users' experience," in *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, M. Giacobini, A. Brabazon, S. Cagnoni, et al., Eds., vol. 4974 of *Lecture Notes in Computer Science*, pp. 485–490, Springer, Naples, Italy, 2008.

[7] R. L. Saunders, *Terrainosaurus: realistic terrain synthesis using genetic algorithms*, M.S. thesis, Texas A&M University, College Station, Tex, USA, 2006.

[8] A. D. Kelley, M. C. Malin, and G. M. Nielson, "Terrain simulation using a model of stream erosion," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*, pp. 263–268, ACM, Atlanta, Ga, USA, August 1988.

[9] J. Olsen, "Realtime procedural terrain generation—realtime synthesis of eroded fractal terrain for use in computer games," Department of Mathematics and Computer Science (IMADA), University of Southern Denmark, 2004.

[10] H. O. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer, New York, NY, USA, 2nd edition, 2004.

[11] R. Voss, "Fractals in nature: characterization, measurement, and simulation," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*, Anaheim, Calif, USA, July 1987.

[12] C. Darwin, *On the Origin of Species by Means of Natural Selection*, John Murray, London, UK, 1859.

[13] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.
[14] T. Bäck, F. Hoffmeister, and H. P. Schwefel, "A survey of evolution strategies," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 2–9, San Diego, Calif, USA, July 1991.
[15] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, USA, 1992.
[16] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley & Sons, New York, NY, USA, 1966.
[17] W. B. Langdon and A. Qureshi, "Genetic programming—computers using "natural selection" to generate programs," Tech. Rep. RN/95/76, University College London, London, UK, 1995.
[18] T. S. Ray, "Evolution, ecology and optimization of digital organisms," 1992.
[19] P. Bentley, "Aspects of evolutionary design by computers," in *Advances in Soft Computing: Engineering Design and Manufacturing*, Springer, New York, NY, USA, 1998.
[20] P. Bentley, *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, Calif, USA, 1999.
[21] K. Sims, "Artificial evolution for computer graphics," in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*, pp. 319–328, ACM, Las Vegas, Nev, USA, August 1991.
[22] T. Unemi, "A design of multi-field user interface for simulated breeding," in *Proceedings of the 3rd Asian Fuzzy and Intelligent System Symposium (AFSS '98)*, pp. 489–494, Masan, Korea, June 1998.
[23] T. Unemi, "SBART 2.4: breeding 2D CG images and movies and creating a type of collage," in *Proceedings of the 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems (KES '99)*, pp. 288–291, Adelaide, Australia, August-September 1999.
[24] T. Unemi, "SBART 2.4: an IEC tool for creating 2D images, movies, and collage," in *Proceedings of the Workshop on Genetic and Evolutionary Computational Conference*, p. 153, Las Vegas, Nev, USA, July 2000.
[25] P. Machado and A. Cardoso, "NEvAr—the assessment of an evolutionary art tool," in *Proceedings of the Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science (AISB '00)*, G. Wiggins, Ed., Birmingham, UK, April 2000.
[26] T. J. Ong, R. Saunders, J. Keyser, and J. J. Leggett, "Terrain generation using genetic algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 1463–1470, ACM, Washington, DC, USA, June 2005.
[27] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*, Lulu, Morrisville, NC, USA, 2008.
[28] M. Frade, F. F. de Vega, and C. Cotta, "Gentp—uma ferramenta interactiva para a geração artificial de terrenos," in *Proceedings of the 3rd Iberian Conference in Systems and Information Technologies (CISTI '08)*, M. P. Cota, Ed., vol. 2, pp. 655–666, Ourense, Spain, April 2008.
[29] M. Frade, F. F. de Vega, and C. Cotta, "Genetic terrain programming—an aesthetic approach to terrain generation," in *The Annual International Conference & Symposium on Computer Games and Allied Technology (CGAT '08)*, pp. 1–8, Singapore, April 2008.
[30] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The information visualizer, an information workspace," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pp. 181–186, ACM, New Orleans, La, USA, April-May 1991.
[31] C. J. Testa and D. B. Dearie, "Human factors design criteria in man-computer interaction," in *Proceedings of the Annual Conference*, vol. 1, pp. 61–65, ACM, 1974.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 436732, 9 pages
doi:10.1155/2009/436732

Research Article

Fine-Tuning Parameters for Emergent Environments in Games Using Artificial Intelligence

Vishnu Kotrajaras and Tanawat Kumnoonsate

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Payathai Road, Patumwan Bangkok 10330, Thailand

Correspondence should be addressed to Vishnu Kotrajaras, ajarntoe@gmail.com

Received 30 May 2008; Accepted 8 September 2008

Recommended by Kok Wai Wong

This paper presents the design, development, and test results of a tool for adjusting properties of emergent environment maps automatically according to a given scenario. Adjusting properties for a scenario allows a specific scene to take place while still enables players to meddle with emergent maps. The tool uses genetic algorithm and steepest ascent hill-climbing to learn and adjust map properties.Using the proposed tool, the need for time-consuming and labor-intensive parameter adjustments when setting up scenarios in emergent environment maps is greatly reduced. The tool works by converting the paths of events created by users (i.e., the spreading of fire and the flow of water) for a map to the properties of the map that plays out the scenario set by the given paths of events. Vital event points are preserved while event points outside the given scenario are minimized. Test results show that the tool preserves more than 70 percent of vital event points and reduces event points outside given scenarios to less than 3 percent.

1. Introduction

For computer games that rely on players interacting with the game maps, emergent and realistic environments become crucial to their degree of realism. Emergent environments can be created on a map by dividing the map into cells, augmenting the cells with various physical properties, and building rules for influencing properties between each cell [1]. Natural phenomena, such as the spreading of fire or the flow of water, look more dynamic and realistic than similar phenomena created using scripts [2].

Emergent maps have been widely used in ecological modeling [3], but for computer games, emergent maps have not seen much use. One reason is because when a given scenario is required to take place on a map, every property of every cell needs to be set in order to produce such scenario. Developers need to put in a lot of time and effort. Moreover, scenarios that have not been well set may behave in ways that the developers had not anticipated. One way to produce a scenario is to make properties inactive and play out the scenario using a script, and then switch the properties back on later. This solution has a limitation because players cannot interfere with the scenario. Players must wait until the scenario is completely played out before being able to do any change to parts of the map that are changed by the scenario. In ecological modeling, an emergent map is used by setting all its properties first (usually imitating a real world map) then allowing the scenario to play itself out. The study of what causes a given scenario is through trial and error. Therefore, it will be very useful if, given a scenario, natural properties that cause it can be found automatically.

We present in this paper a hybrid of genetic algorithm and steepest ascent hill-climbing for adjusting properties of every cell on a cellular automata map in order to produce a given scenario. With a tool based on our technique, game developers are able to focus on designing their scenario and spend much less time setting cell properties. Controllable games scenarios lead to the following features, which are usually unavailable in emergent games.

(i) Editable scenario—scenario are very important for games. Crucial moments in a game story can be revealed using well-set scenarios. Furthermore, a well-set scenario can

provide challenge for players. Normally, a scenario is played out using scripts, sacrificing any possible interactions from players. Our editable scenario is different. Developers can specify how a scene is played out just like writing scripts to dictate what happens during the game, but the environment remains emergent throughout the entire play.

(ii) In-game cut scene—although cut scenes produced as movies can be used, in-game cut scenes can tell stories while players are still in the middle of scenarios, without disrupting game flow.

2. Related Works

Regarding game environment, Sweetser and Wiles [1] have developed and tested a cellular automata map for real-time strategy games. Sweetser's experimental system was called EmerGEnT system. Although EmerGEnT system did not model an environment in great detail, it was good enough for using in games, with fire, water, and explosions being integrated into its cellular automata properties. EmerGEnT system can be divided into 3 levels. The first layer is the behavior layer, which shows the effects that players see. The second layer is the rule layer, which controls behavior both between cells and within each cell. The final layer is the property layer, which determines how cells interact according to the rules. Sweetser's work, although consists of many physical properties, does not provide any feature for setting cell properties following given effects that users see.

A probabilistic model can also be used to simulate fire in broad scale over long time periods [3, 4]. Hargrove's model was designed for simulating real forest fires. It included humidity, fuel types, wind, and firebrand. These factors influenced the probability of fire spreading from one cell to another, and the probability of isolated cells getting ignited. Probabilistic models do not perform well for small cell sizes compared to thermodynamic models. To be able to work with cells in a game map, which generally represents small areas, thermodynamic models are more suitable. Probabilistic models are also more difficult to control compared to thermodynamic models, which have precise rules for events.

Hill-climbing can be used for tuning system parameters. Merz et al. [5] developed Opi-MAX for tuning MAX's numeric parameters. MAX is an expert system for high-level diagnosis of customer-reported telephone troubles. It can be customized by changing a set of numeric parameters. Opi-MAX uses a searching algorithm called greedy hill-climbing to optimize parameters. It works by randomly changing parameters one by one. A parameter is repeatedly changed from its initial value and each of its change effect can be observed from overall output. A change that results in a better output will be carried out, and a change that does not contribute to a better output will be ignored. Each parameter is changed for a constant number of times. The system stops when all parameters are dealt with. This technique, however, is not suitable for our work, since we have no good initial values of any property. It can be used to help improving some partially tuned parameters, nevertheless. Therefore, we use it for fine-tuning environment properties adjusted during our genetic algorithm runs.

Neural networks can also be used for parameter tuning. Legenstein et al. [6] developed a movement prediction method for objects moving on 8×8 grids. Each cell on a grid provides an input to the recurrent unsupervised neural networks. It can predict object movement by predicting sensor inputs, which are numeric values like cell properties. Given previous events, Legenstein et al. work predicts the next event, while our work predicts the initial condition given a sequence of events in time. We have experimented with neural networks but found that results were unsatisfactory. Fire only followed given waypoints for about 50% and spread out of the waypoints more than 100%. We believe it was because cells that had different cell properties were allowed to produce similar events (even though the intensity of fire might be different). Therefore, fires with different intensity were trained with similar events, causing the neural network to fail to learn effectively. Therefore, we switched to genetic algorithm.

Breukelaar and Bäck [7] used genetic algorithm to generate cellular automata's transition rules that display a desired behavior. The work was demonstrated by finding rules that evolve all cells to the same state by majority, evolve cells to form a checkerboard, and evolve cells to form desired bitmaps. The main focus was to find transition rules, given a single parameter. In our work, the rules are given, but we need to find several parameter values of all cells at the start of each scenario. Therefore, the genetic encoding is totally different from [7]. Karafyllidis [8] used genetic algorithm to convert continuous-state cellular automata that predicts forest fire spreading to discrete-state cellular automata that outputs nearest results. The number of cells and states for the discrete-state version were also minimized. The outcome cellular automata were used to build a dedicated parallel processor for real-time processing. Only one parameter was used in each cell. It was the rate of fire spreading. Using only one parameter allows faster execution. However, such approximation is not applicable for our work since we want players to still be able to physically change all properties of each cell.

3. Tool for Adjusting Properties of Emergent Environment Maps

3.1. Overview. We base our cellular automata maps and rules on EmerGEnT system [1]. Our tool can adjust properties in EmerGEnT system-like map automatically to create a scenario of fire spreading and water flowing that matches the scenario given by a user. Properties that our tool adjusts include material, temperature, mass, damage, and wetness for the spreading of fire, and height for the flow of water. Paths of both kinds of events are controlled by waypoints (currently, there are two kinds of events, fire and water flow). Each waypoint is defined by its position, the time an event takes place at that position, and the radius of the event, as shown in Figure 1. The tool then creates a timetable containing the beginning and the end of each event on every

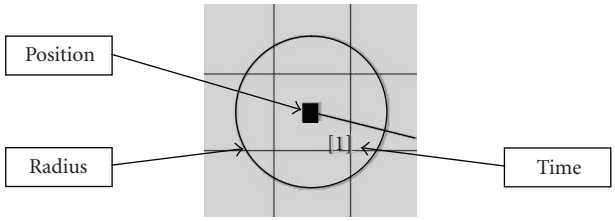


Figure 1: Waypoint.

cell. Impossible paths, such as fire crossing water ways or fire spreading in the opposite direction to the wind, are not produced. When a fire is spreading, its burning area moves along the given waypoints. For water, flooding areas never dry out. At the beginning of the spreading of fire, an object that changes temperature to 3000 units is put into the first waypoint at the time given by a user. For water, an object that adds 30 units of water per unit time is put into the first waypoint at the time given by the user. There can be more than one starting waypoint for both fire and water, depending on each designed scenario. The map properties are then adjusted by genetic algorithm, with steepest ascent hill-climbing being applied at selected generations. Finally, the best chromosomes are kept to be included in the initial population of genetic algorithm when adjusting other scenarios. These chromosomes will be chosen during genetic algorithm if they fit any newly given scenario.

3.2. Genetic Algorithm. Each individual in our population contains 2 strands of chromosomes, chromosomes that store properties related to fire (see Figure 2) and chromosomes that store properties related to water (see Figure 3). Separating fire and water properties results in a reduced search space for each type of events because for each type of events, there are some properties unrelated to it. Properties related to fire include material, temperature, mass, damage, and wetness. There is, according to EmerGEnT model, only 1 property related to water. It is the water's height. All property values are stored in real number, except the material value, which is stored as an integer representing the material identifier of a cell. Each cell does not need a special firing condition because its properties (including how much it burns and how much water is in it) constantly change according to properties of neighbor cells and related properties within that cell, according to the terrain physical rules given for EmerGEnT model. In order for a scenario to take place, we only need a starting condition, which the system gives by putting in fire sources or water sources at starting waypoints.

Our genetic algorithm has population of 1000 chromosomes and evolves for 100 generations. The initial population is generated randomly, with some limited range defined for each value in order to speedup convergence. A new generation is selected from the following.

(i) The highest fitness chromosome.

(ii) The first 1% elitist chromosomes. These will be subjected to mutation.

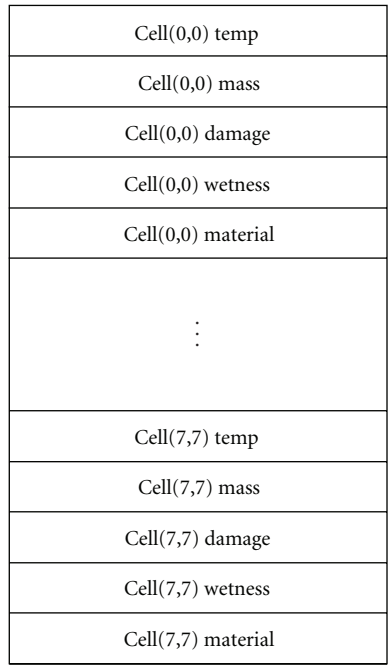


Figure 2: Chromosome for fire event.

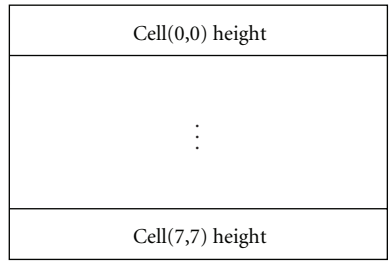


Figure 3: Chromosome for water event.

(iii) The crossover of chromosomes chosen by (1). selectPopulation is an integer used to identify a chromosome. The lower the value, the higher the fitness of the selected chromosome. For example, if selectPopulation is 0, we know we have selected the highest fitness chromosome. Function random(a, b) returns a random number between a and b, but not including b. Equation (1) guarantees that the bottom half of the chromosomes will never be chosen for crossover. After the crossover finishes, half of the resulting chromosomes are mutated.

Our genetic algorithm uses elite strategy because it tries to select elite chromosomes first. The first 1% of the highest fitness chromosomes is selected for certain and the rest are in the top half:

$$\text{selectPopulation} = \text{random}\left(0, \frac{\text{populationSize}}{\text{random}(2,6)}\right). \quad (1)$$

When we perform a crossover between two chromosomes, each property value on the first chromosome can be

combined with its counterpart from the second chromosome. We use uniform crossover with blending defined by (2):

$$p_{\text{new1}} = \beta p_1 + (1-\beta)p_2,$$
$$p_{\text{new2}} = \beta p_2 + (1-\beta)p_1. \quad (2)$$

From (2), p_1 is a property value (in real number) selected from the first chromosome, for example, it can be the temperature of a cell at coordinate (2, 3) in our map, as defined by the first chromosome. p_2 is a corresponding property value from the second chromosome. For example, if p_1 is the temperature of a cell at coordinate (2, 3), as defined by the first chromosome, then p_2 must be the temperature of a cell at coordinate (2, 3), as defined by the second chromosome. p_{new1} is the new value of that property in the first resulting chromosome. p_{new2} is the new value of that property in the second resulting chromosome. β is used to combine the values from p_1 and p_2. The value of β varies in each crossover. β is selected randomly from 0, 1 and a random value between 0 and 1. The chance of selecting each choice from these 3 choices is equal. We experimented with 2 other ratios for these 3 choices on 3 sample maps (the first map has more than 1 fire path, the second map has a very long fire path, and the third map contains a very long water flow). With all other settings equal, on average, equal ratio gave the best result. Therefore, we decided to use it in our genetic algorithm.

Mutation rate of 10, 20, and 30% were tested on the 3 sample maps above. It was found that the mutation rate of 20% gave the best result on average. Therefore, we choose the mutation rate of 20% for our genetic algorithm. The mutation range is constrained to be within 50 units away from the old value in order to prevent very odd chromosomes with low fitness from being produced.

The fitness value of each chromosome is calculated from the average of the fitness of each time unit that events occur, with weight defined by (3). Any time frame that contains event(s) at waypoint(s) is given a higher fitness value than a time frame with no event point in order to make the scores at event points stand out. Events occurring later in a scenario are also considered more important than events occurring early in the scenario. This allows different starting scenes for a single scenario

$$\text{fitness}_t^{\text{waypoint}} = \left(1.8 + \frac{0.2 \times t}{\text{time}_{\max}}\right) \times \text{fitness}_t,$$
$$\text{fitness}_t^{\text{non-waypoint}} = \left(0.8 + \frac{0.2 \times t}{\text{time}_{\max}}\right) \times \text{fitness}_t. \quad (3)$$

Our fitness functions were defined after several experiments. $\text{fitness}_t^{\text{waypoint}}$ is the fitness value at time t, if that time contains event(s) at waypoint(s). The importance of that time frame has also been weighed into its value. $\text{time}_{\max}$ is the maximum time frame that the current scenario takes place. fitness_t is the fitness value at time t before being given any weight. $\text{fitness}_t^{\text{non-waypoint}}$ is defined similar to $\text{fitness}_t^{\text{waypoint}}$, except that it is given less weight due to its lack of event points.

The fitness of each time frame before weighing (fitness_t) is defined in (4) as follows:

$$\text{fitness}_t = \frac{\sum_{i=1}^{n} \text{fitness}\,(\text{relevantCell}_i)}{m \times 10}, \quad (4)$$

relevantCell_i is a cell that burns or floods at time t or was designed to burn or flood at time t (we do not count a cell twice, however). The number n is the sum of the number of cells that actually burn or flood at time t and the number of cells designed to burn or flood at time t (again, we do not count a single cell twice). fitness(c) is a fitness value of cell c. It can have a negative value depending on whether the cell being in the designed path or not (see below). m is the number of cells designed to burn or flood at time t. The maximum fitness score for each cell is 10. If our parameter tuning is perfect, cells designed to burn or flood at time t will actually burn or flood at that time frame with fitness value equal to 10. In addition, no other cells will burn or flood. Therefore, n will equal to m and the value of fitness_t will be 1.

The value of fitness(c) is calculated from each of the following steps.

(i) If a cell outside specified paths produces events—lose 1 point if an adjacent cell is inside any event path. If the cell does not have any adjacent path, 5 points are deducted instead. This discourages events outside the specified path.

(ii) If a cell does not produce an event when the event is set to occur—gain points equal to two times the cell temperature divided by maximum temperature if the event is a spreading of fire. The maximum score obtainable from this portion of the function is 2. Water events get no score here. The reason the fire situation gets some score even though the cell does not produce the event is because high temperature gives the cell a probability of burning in later time frames, which can result in similar fire events later on.

(iii) If a cell produces an event at its specified time—gain 8 points. Get additional points according to (5) and (6) for fire and water, respectively,

$$\text{addScore}_{\text{fire}} = \frac{2 * \text{Burn}}{0.5 * \text{MaxBurn}}. \quad (5)$$

For (5), Burn is the intensity of fire in the calculated cell and MaxBurn is the maximum possible value of Burn. The maximum score from this equation is limited to 2. Any burn that spreads with at least half the intensity of the maximum intensity will get full mark. The reason we need $\text{addScore}_{\text{fire}}$ is to encourage all fires to burn fast. From our experiment, this can prevent fires from dying out unexpectedly in the middle of scenarios:

$$\text{addScore}_{\text{water}} = \frac{2 * \text{fluid}}{0.5 * \text{MaxFluid}}. \quad (6)$$

For (6), fluid is the amount of water currently in the cell. MaxFluid is the maximum amount of water that cell can contain. Similar to (5), (6) has its maximum value being 2 and it is needed in order to prevent water from flowing not as far as designed. The difference from (5) is that we use the amount of water instead of speed. This is because in our model, based on Sweetser's, there is no water speed parameter.

Table 1: Result of hill-climbing test.

Setting	20th–100th	60th–100th	100th	Not use
Average Map 1	0.516818	0.542901	0.527788	0.476567
Average Map 2	0.448464	0.457916	0.447966	0.406109
Average Map 3	0.669976	0.66909	0.669541	0.669655
Average All	0.545086	0.556636	0.548431	0.517444

```
int  tuneValueFire = 27;
real bestFitness;
for each cell{
      //tune for fire event
      real currentFitness = · · · · //calculate the cell's fitness value
      While (tuneValueFire >0){
         Find the fitness when add the temperature value by tuneValueFire
         Find the fitness when subtract the temperature value by tuneValueFire
         Find the fitness when add the mass value by tuneValueFire
         Find the fitness when subtract the mass value by tuneValueFire
         Find the fitness when add the damage value by tuneValueFire
         . . .//try both add and subtract for all parameters of fire chromosomes
         . . .

         if (the best fitness value (compared to currentFitness) is found from the tuning
         trial results above){
               select a modification that causes such fitness
               commit changes according to the selected modification
         } else {
               tuneValueFire = tuneValueFire;
         }
      } // end while
} // end for each cell
```

Algorithm 1: Pseudocode for steepest ascent hill-climbing on fire related values.

(iv) If a cell produces a specified event before its intended starting time, but not more than 1 time unit—gain 4 points. This is to allow a slightly different scenario to still gain points.

(v) If a cell still produces a given event after its intended end time, but not more than 4 time unit—in case of fire, gain 5 points minus the difference between current time and end time. If the difference in time is just 1, the score will be 4, similar to the score when an event takes place before its intended starting time. But we give points for other nearby time frames in order to allow for fire trails. From our experiments, fire trails are very important for an overall fitness of fire events. There is no score for water remaining in a cell, however, since our system follows Sweetser's model that lets water stay in a cell indefinitely.

3.3. Steepest Ascent Hill-Climbing. We use steepest ascent hill-climbing to the best chromosome, with the same fitness function as our genetic algorithm, in order to improve map's properties. Table 1 shows the effect of steepest ascent hill-climbing used in our tool, in term of fitness values compared among four settings. Three settings employ steepest ascent hill-climbing at every 20 generations of the genetic algorithm.

In the first setting, we start applying it at the 20th generation. In the second setting, we start applying it at the 60th generation. In the third setting, we apply the algorithm only to the last generation (100th generation). In the fourth setting, we do not use steepest ascent hill-climbing algorithm. Each setting is tested on three different maps of 8×8 cells. For each map, each setting is tested 3 times. The test maps are chosen to represent 3 scenarios that could be set by developers. Map 1 contains a fire that breaks into 2 paths. Map 2 contains a long path of fire. Map 3 contains water flow. Each property is tuned in sequence until it cannot be tuned further. The values used in tuning come from observations during experiments. The algorithm for tuning fire-related parameters is shown in pseudocode in Algorithm 1. For water events, the algorithm is similar, except it works only on water-related parameter.

The result shows that, on average, tests that steepest ascent hill-climbing are applied to have noticeably better fitness values than the tests without steepest ascent hill-climbing. From Table 1, it seems that starting to apply steepest ascent hill-climbing at the 60th generation gives the best fitness value (except in Map 3, where its fitness value is slightly lower than the fitness values from other

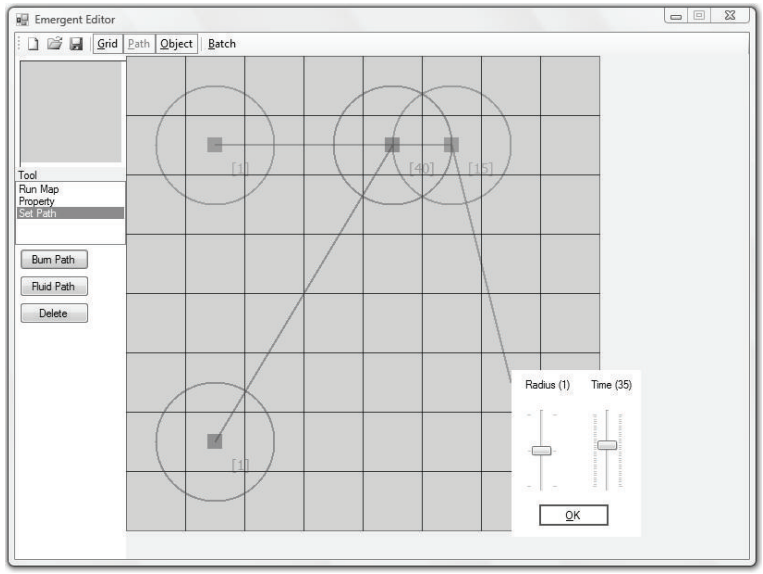


Figure 4: Path editor.

tests). More experiments are needed in order to determine whether steepest ascent hill-climbing produces a significantly better result than ordinary genetic algorithm in our problem. We, therefore, run our parameter tuning with genetic algorithm alone and with our steepest ascent hill-climbing as the genetic algorithm enhancement (we apply the second setting from Table 1). The testing setup and its outcome are discussed in Section 4.

3.4. Emergent Editor. Our tool for adjusting properties of emergent environment maps is in the form of a map editor. We name the editor emergent editor. Its features can be divided into 4 parts.

(i) Path editor (Figure 4)—a user can define, edit, or delete waypoints of fire events and water events.

(ii) Automatic properties adjustment (Figure 5)—this feature adjusts map properties automatically according to a given scenario defined by paths of events.

(iii) Event player (Figure 6)—it can show the original scenario defined by paths of events, and the scenario created after the map properties are set.

(iv) Property editor (Figure 7)—a user can also view or edit map properties directly from this view.

4. Testing and Results

In this section, we first discuss results from experiments using genetic algorithm enhanced by steepest ascent hill-climbing (which gives better results than using genetic algorithm alone). Then we discuss whether using steepest ascent hill-climbing really gives significantly better results statistically. Finally, an experiment showing how our parameter-tuning tool can help map designers save time is presented.

Testing is initiated by creating paths of events randomly on a map of 8×8 cells. Paths of events are limited to total of 6 waypoints. Events with the same number of waypoints are tested between 2–5 times. Each waypoint has its radius of one or two cells. The total running time of each event is limited to 50 time units. The wind direction is chosen randomly between no wind and random direction. From 100 tests, using genetic algorithm and steepest ascent hill-climbing, our result shows that the tool preserves 75.09% of event points at waypoints on average and produces event points outside given scenarios by only 2.3% on average.

Figure 8 shows our test result grouped by the number of waypoints. The horizontal axis represents various fire and water scenarios. $\mathrm{Fm}_1\mathrm{m}_2 \ldots \mathrm{m}_n$ represents n fire paths in the map, where the first path has m_1 waypoints, the second path has m_2 waypoints, and so on. $\mathrm{Wm}_1\mathrm{m}_2 \ldots \mathrm{m}_n$ represents water paths in the same way. The vertical axis represents the percentage of event points for each scenario.

Figure 9 illustrates one of the test scenarios with 3 waypoints of fire spreading. Figure 9(a) shows the events designed to take place at 3 points in time, while Figure 9(b) shows actual events that take place after the actual parameter adjustment at the same points in time. Darker cells are forest cells, while lighter cells are grass cells. There is also a water cell at the middle-bottom of the map (white cell). Cells with white circles are cells that catch fire.

There are 11 tests that our tool gives less than half of events correctly at waypoints. When we look into their causes, we discover that their scenarios are impossible to take place. Fire was set to burn longer than the fuel in the map could support. Water was set to flow too quick or too far from its source. In our system, the further away from its source, the slower the water can flow. This is because there are more cells to absorb water.

In order to find out whether steepest ascent hill-climbing significantly enhances the accuracy, we run the experiment again, with and without steepest ascent hill-climbing, and compare their results using a paired t-test.

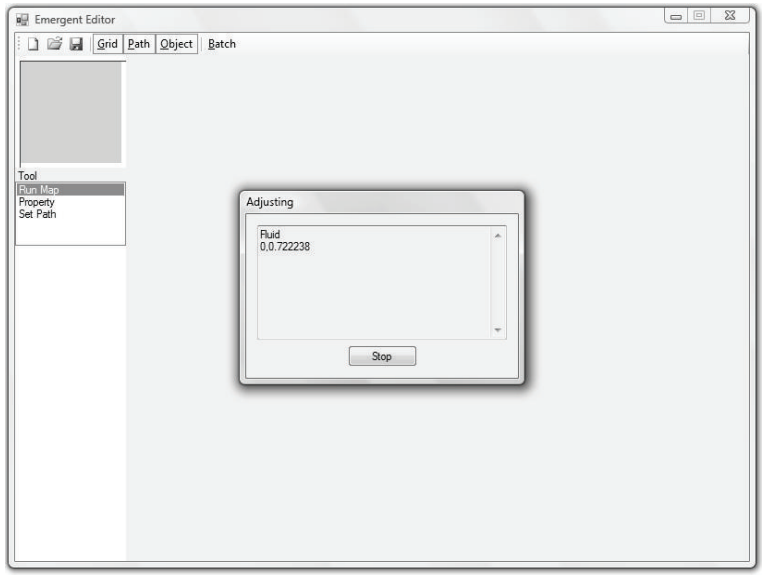


Figure 5: Automatic properties adjustment.

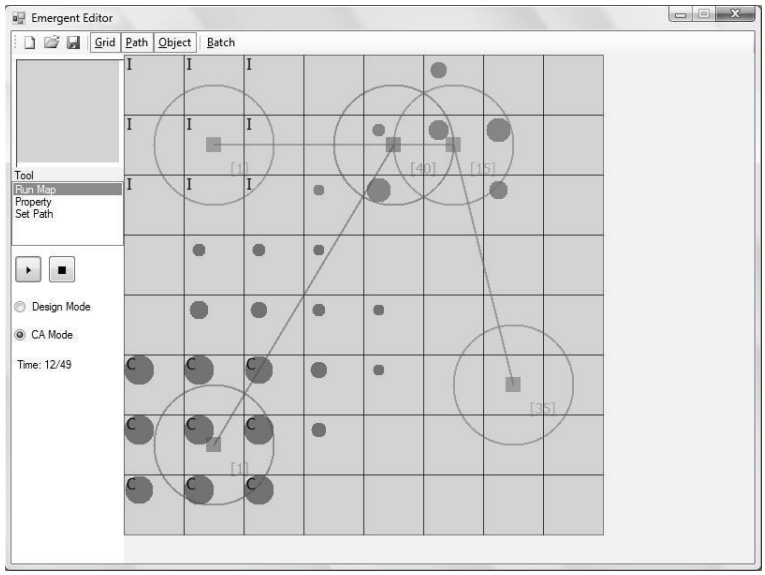


Figure 6: Event player.

The paired t-test result for the number of preserved waypoints informs us that the two-tailed P value is less than .0001. By conventional criteria, this difference is considered to be extremely statistically significant. The difference between the mean of the experiment using steepest ascent hill-climbing and the experiment that does not use steepest ascent hill-climbing equals to 2.2953264. The 95% confidence interval of this difference is from 1.3483512 to 3.2423016. The intermediate values used in calculations include $t = 4.8094$, df $= 99$, and standard error of difference $= 0.477$. For the percentage of outside fire path, the difference is found to be not quite statistically significant. The 95% confidence interval of this difference is from -3.10873116 to 0.26262415. For the percentage of outside water path, the difference is found not to be statistically significant. The 95% confidence interval of this difference is from -0.29874686 to 0.40685569.

From the t-test results, we can conclude that using steepest ascent hill-climbing gives a significant boost to the number of preserved waypoints. Therefore, it should be used in conjunction with genetic algorithm for tuning map parameters.

In order to test whether the tool that uses our parameter-tuning technique actually benefits scenario designers, we perform an experiment by having 6 testers manually tune the maps from Section 3.3 for 30 minutes per map

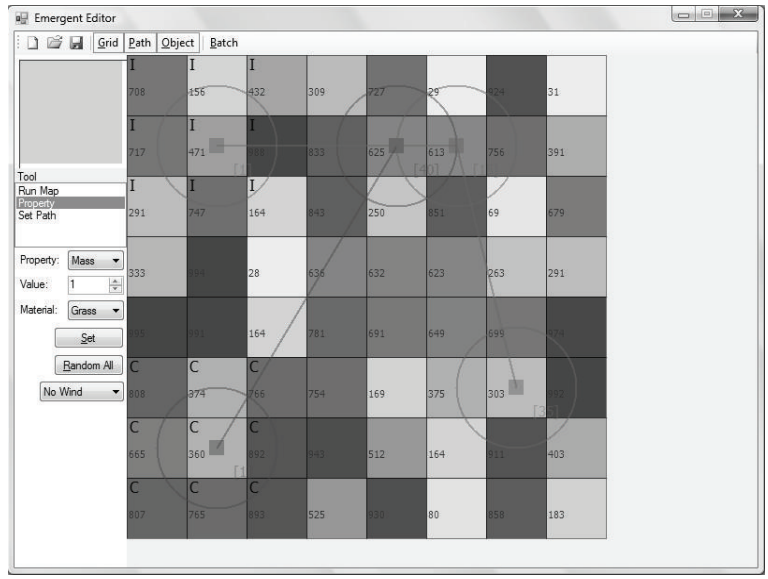


Figure 7: Property editor.

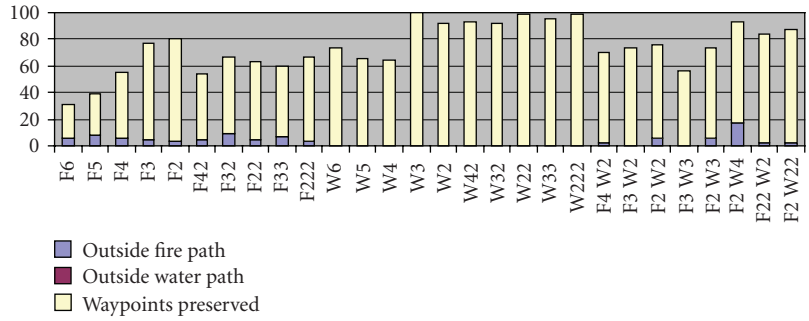
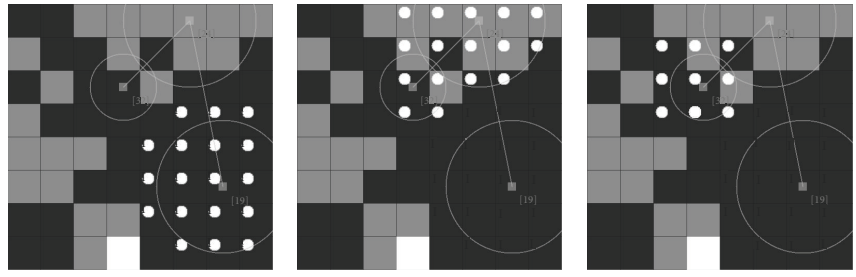
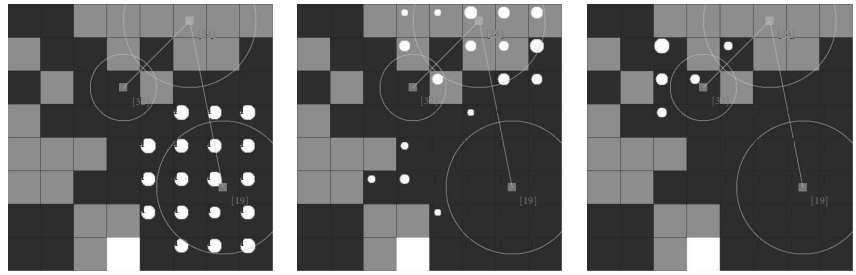


Figure 8: Test result.



(a)



(b)

Figure 9: Test scenario.

Table 2: Tuning accuracy results from testers and the tool.

Tester ID	Waypoint preserved (%)	Outside path (%)
1	55.68	1.23
2	61.11	0.00
3	62.47	0.00
4	42.22	45.47
5	51.11	0.00
6	35.56	0.00
Average tester result	51.36	7.78
Our tool result	76.05	10.06

(the 30-minute period is the time our testers are willing to spend). We compare the testers' results with the results obtained by our tool in Table 2.

From Table 2, it can be seen that our tool is much more precise in preserving waypoints, given an equal period of operation time. Therefore, our tool is capable of producing accurate scenarios faster than human. For outside fire and water paths, our tool performs worse than human. This turns out to be because most of the testers cheat by removing fuel from all outside paths. This cheat cannot be done in actual nature simulations or games because it will produce very unnatural maps. For tester 4 who does not cheat, the amount of outside path is 45.47%. This is another good indication that using our tool can save valuable development time.

5. Conclusion

From our experiment, we conclude that genetic algorithm, with help from steepest ascent hill-climbing technique, can be used effectively for adjusting parameters in emergent maps which leads to simple-to-control scenarios without the need to manually edit any property.

Some problems, such as fire burning out before the expected ending time and water running too slowly or too short in distance, still need to be solved. This can be solved by having the algorithm also adjust the initial temperature of fire and the amount of water at the first waypoint. There are also other possible improvements. It may be useful to allow users to control scenarios with other means besides creating paths of events. Increasing the speed of the tool will allow a better use with bigger maps and more complex environments. Other kinds of emergent environments, such as environments used for actual ecological modeling, are good candidates for expanding the value of our tool.

Acknowledgment

This research is sponsored by Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand.

References

[1] P. Sweetser and J. Wiles, "Using cellular automata to facilitate emergence in game environments," in *Proceedings of the 4th International Conference on Entertainment Computing (ICEC '05)*, Sanda, Japan, September 2005.

[2] P. Sweetser and J. Wiles, "Scripting versus emergence: issues for game developers and players in game environment design," *International Journal of Intelligent Games and Simulations*, vol. 4, no. 1, pp. 1–9, 2005.

[3] W. W. Hargrove, R. H. Gardner, M. G. Turner, W. H. Romme, and D. G. Despain, "Simulating fire patterns in heterogeneous landscapes," *Ecological Modelling*, vol. 135, no. 2-3, pp. 243–263, 2000.

[4] W. Song, F. Weicheng, W. Binghong, and Z. Jianjun, "Self-organized criticality of forest fire in China," *Ecological Modelling*, vol. 145, no. 1, pp. 61–68, 2001.

[5] C. J. Merz, M. Pazzani, and A. P. Danyluk, "Tuning numeric parameters of a knowledge-based system for troubleshooting the local loop of the telephone network," *IEEE Expert*, vol. 11, no. 1, pp. 44–49, 1996.

[6] R. Legenstein, H. Markram, and W. Maass, "Input prediction and autonomous movement analysis in recurrent circuits of spiking neurons," *Reviews in the Neurosciences*, vol. 14, no. 1-2, pp. 5–19, 2003.

[7] R. Breukelaar and Th. Bäck, "Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior," in *Proceedings of Conference on Genetic and Evolutionary Computation (GECCO '05)*, pp. 107–114, Washington, DC, USA, June 2005.

[8] I. Karafyllidis, "Design of a dedicated parallel processor for the prediction of forest fire spreading using cellular automata and genetic algorithms," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 1, pp. 19–36, 2004.