Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2011, Article ID 819746, 15 pages
doi:10.1155/2011/819746

Research Article

Enhancing a Commercial Game Engine to Support Research on Route Realism for Synthetic Human Characters

Gregg T. Hanold[1] and Mikel D. Petty[2]

[1] *Oracle National Security Group, Reston, VA 20190, USA*
[2] *Center for Modeling, Simulation, and Analysis, University of Alabama in Huntsville, Huntsville, AL 35899, USA*

Correspondence should be addressed to Mikel D. Petty, pettym@uah.edu

Received 30 July 2011; Revised 19 December 2011; Accepted 22 December 2011

Academic Editor: Kok Wai Wong

Generating routes for entities in virtual environments, such as simulated vehicles or synthetic human characters, is a long-standing problem, and route planning algorithms have been developed and studied for some time. Existing route planning algorithms, including the widely used A* algorithm, are generally intended to achieve optimality in some metric, such as minimum length or minimum time. Comparatively little attention has been given to route realism, defined as the similarity of the algorithm-generated route to the route followed by real humans in the same terrain with the same constraints and goals. Commercial game engines have seen increasing use as a context for research. To study route realism in a game engine, two developments were needed: a quantitative metric for measuring route realism and a game engine able to capture route data needed to compute the realism metric. Enhancements for recording route data for both synthetic characters and human players were implemented within the Unreal Tournament 2004 game engine. A methodology for assessing the realism of routes and other behaviors using a quantitative metric was developed. The enhanced Unreal Tournament 2004 game engine and the realism assessment methodology were tested by capturing data required to calculate a metric of route realism.

1. Introduction

Entities in virtual environments, such as simulated vehicles or synthetic humans, move from place to place in the virtual environment. Algorithms to automatically generate those routes have been developed and studied for some time. The A* graph search algorithm, because of its simplicity and effectiveness, has been applied in a range of simulation environments, including Close Combat Tactical Trainer, Combat XXI, and OneSAF Objective System, and games such as Warcraft and Civilization; some of these applications use variants of the basic A* algorithm [4]. Most route planning algorithms, including A*, are designed to produce or approximate *optimum* routes, where optimality is measured in terms of some application-specific metric; examples include minimum distance for individual humans moving in urban terrain [5], minimum distance for vehicles moving in a road network [6], minimum exposure to threats for combatants moving in a battle area [7], or maximum sensor coverage for search platforms surveying a target area [8]. In contrast, very little attention has been given to producing *realistic* routes, where realism is defined as the similarity of the generated route to a route that would be followed by a real human in the same terrain with the same constraints (e.g., starting and ending locations and movement capabilities) and goals (e.g., minimizing exposure to threats). The route realism research performed to date has been largely focused on very short routes (e.g., within a single room [9]) or highly specific circumstances (e.g., avenues of approach for large vehicle formations [10]).

Commercial game engines have been quite successful at their primary purpose, which is to provide a framework within which to develop engaging and entertaining virtual environments. As their architectures have matured in terms of software design and become more open to external modification, game engines have seen increasing use as a context for research in human behavior modeling (e.g., [5]). Generating realistic behavior in a game engine-generated virtual

Figure 1: Satellite view of the real world McKenna MOUT facility [1].

environment requires, among other things, that the virtual environment replicate those aspects of the real world that affect the behavior to be generated. The game industry has approached this goal with the massively multiplayer online role-playing games such as World of Warcraft and first-person shooter games such as Quake III Arena, Half-Life, and Americas Army. However, validation of the realism of behavior generated for algorithm-controlled characters (hereinafter known as "Bots") in a game engine is still a developing discipline [11]. To date, such validation has largely been limited to face validation by subject matter experts (e.g., see [5, 10]; for more information on face validation, see [12]), due to both limitations in the facilities provided by the game engines to capture validation data and gaps in the quantitative methods available to validate human behavior.

To study route realism for synthetic human characters in the context of a commercial game engine, two developments were needed: a quantitative metric for measuring the realism of a route, and a game engine appropriately enhanced to capture the data about routes executed in the game engine needed to compute that metric [13]. This paper describes those developments. Enhancements for extracting and recording route data (time, location, heading, velocity) for both Bots and human players as they follow routes within the Unreal Tournament game engine are described in Sections 2 and 3. The bot and human data recording through the API during scenario execution required dynamic storage external to the game engine. That data was used as input to a quantitative realism metric for analysis of the routes' realism. A new seven-step methodology for the creation and use of an objective quantitative or statistical metric based on the data captured from a virtual environment and the application of that methodology to develop a metric of route realism is described in Section 4. The methodology was tested and refined through live data collection. The enhancements are specific to the particular game engine used, but suggest what may be required in another game engine. The methodology is applicable to any game engine.

2. Simulation Environment

As previously reported [13, 14], the selection of the simulation or virtual environment to implement the realism metric considered several factors. First, the environment itself must present a realistic representation of the real world with respect to the behavior in question. Second, the API must allow for the collection, measurement, and storage of game and environmental data during run time without impacting game engine performance. Third, the API must support integration with the game engine physics and artificial intelligence engines. Finally, the virtual environment (map or level) must have an interface to allow route data (or other behavior data) collected from humans executing defined scenarios in the physical environment to be input for statistical comparison.

2.1. Game Engines. Three commercial game engines were considered: Quake III Arena, Half-Life 2, and Unreal Tournament 2004. While Quake III Arena and Half-Life 2 met the criteria, Unreal Tournament 2004 (UT2004) ships with synthetic agents or Bots and provides through the custom scripting language, UnrealScript, both an interface to these bots and an interface through which game developers can modify the host game without access to the complex game engine source code. UnrealScript provides a rich object-oriented interface to the UT2004 game engine or modifications to it such as Ravenshield and Infiltration. Other UT2004-based games, such as America's Army and Vegas, lock or limit the ability to make modifications through UnrealScript. With its object-oriented interface and the availability of an Integrated Development Environment (IDE), UT2004 was selected as the base game engine. Since the initial research and selection of UT2004, Epic Games has released Unreal Tournament 3, which offers significant improvements to game and level design. However, at the time of this writing, the port of the API and IDE was not complete.

2.2. Virtual Environment. An important step in the creation of a virtual environment suitable for the measurement of the realism of behaviors is the creation of the virtual world. The virtual environment should model a real-world location to potentially allow the comparison of algorithm-generated behaviors with those of humans executing similar behaviors in the real world. The UT2004 virtual world developed was modeled after Fort Benning's McKenna Military Operations in Urban Terrain (MOUT) training facility; the real-world McKenna facility is shown in Figure 1, and its UT2004 virtual recreation is shown in Figure 2.

The game engine must provide an algorithm to generate bot behavior of the type to be assessed for realism. For this research, the behavior was route planning and execution; route planning for the bot used the A* algorithm and route execution used the game engine's standard route following process. In UT2004, A* requires only one custom object, to mark the destination. Native to the UT2004 is the UnrealScript language through which custom map objects can be added. The BotDestinationPathNode was added to the available map objects to provide a destination for the A* route calculation; the script to do so is in (Algorithm 1.) The A* algorithm required nodes to be added to the virtual terrain and their edges to be computed. The "Build AI Paths" function in the UnrealEd creates an internal search graph of nodes (pathnodes) and edges (reachspecs), as shown in Figure 3.

Figure 2: "Satellite" view of the virtual McKenna MOUT facility [2].

```
// BotDestinationPathNode. Goal Node
class BotDestinationPathNode extends PathNode
        placeable;
```

Algorithm 1: Unrealscript pathnode listing.

2.3. Application Programming Interface (API). The first key element to the development of a virtual environment suitable for measuring realism was the existence of an API to both the game engine hosted on a server and the human player and bot clients. As previously noted, UT2004 was one of the first game engines to ship with algorithm-controlled characters, that is, with bots. UT provides a custom scripting language, UnrealScript, through which game developers can modify the host game. In addition to an interface with the UT2004 game Engine, UnrealScript also supports integration with the map editor, UnrealEd, and several integrated development environments. Through the API, we developed an architecture that implements the instrumentation interface required to monitor and record bot and human player execution data from which the realism metric can be calculated.

2.4. Integrated Development Environment. A second key element of virtual environment suitable for measuring realism was the Integrated Development Environment (IDE). Several IDEs can be used with the UT2004 Game Engine. These include Visual Studio (.NET framework with C# and C++), Netbeans (Java), and Eclipse (IDE). For this work, we used the Netbeans IDE. Netbeans was configured with the Pogmut2 plug-in to UT2004 [3] for bot development and data recording, jdbc drivers for database connectivity, and JChart for near-real-time charting. The architecture required to implement the UT2004 simulation environment includes the following software packages: (1) UT2004 and (2) Netbeans (JChart and Pogamut 2 plug-in) and (3) Gamebots 2004 (GB2004) UnrealScript Library and (4) Database with Spatial Libraries.

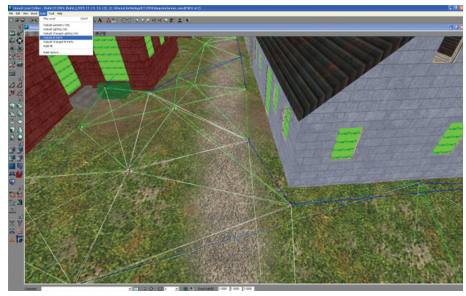

Figure 3: Close up view of UT2004 pathnodes and reachspecs in the virtual McKenna.

3. Architecture and Implementation

Implementing an architecture that supports realistic behavior representation revealed several challenges. The first was to develop an interface that would allow player route data to be collected and recorded without triggering the UT2004 cheat protection code. The second was to develop a mechanism through which route data for humans executing the defined scenarios in the real world could have route data imported into the UT2004 virtual environment for statistical comparison to route data from routes executed in the virtual environment. To solve these challenges, GB2004 and the Pogamut Libraries were modified. By using GB2004 and Pogamut, all data collection was independent of UT2004, thereby avoiding the anticheat triggers.

3.1. Component Overview. The components required to implement an environment suitable for the development of a metric that would measure the realism of bot actions consist of the server node which hosts the UT2004 game engine server, the client node hosting the Unreal client, the IDE node hosting the bot and experiments, and the database node hosting a database with the spatial libraries for data storage and route planning.

GB2004 is an UnrealScript package jointly developed by University of Southern California and Carnegie Mellon University as an interface between the server and the clients that provide an interface to the UT2004 engine. The interface with UT2004 can access sensory information, such as the location and rotation of a player or bot in the game world, messages generated through game play, and UT2004 action commands that control bot behavior.

The interface with the client provides a synchronous and asynchronous messaging interface that allows bot action commands to be issued from client to server and data and game information to be requested and received by the client. Marshall implemented this capability through a higher-level interface, called JavaBot API [15], which handled the specific GB2004 protocol, network socket interface, and client-server messaging. That work demonstrated an environment that supports the development of bot logic using the UT2004 engine.

Expanding on the JavaBot API and extending the GB2004 UnrealScript, Gemrot developed the Pogamut plug-in to the Netbeans IDE. The base Pogamut Architecture, shown in

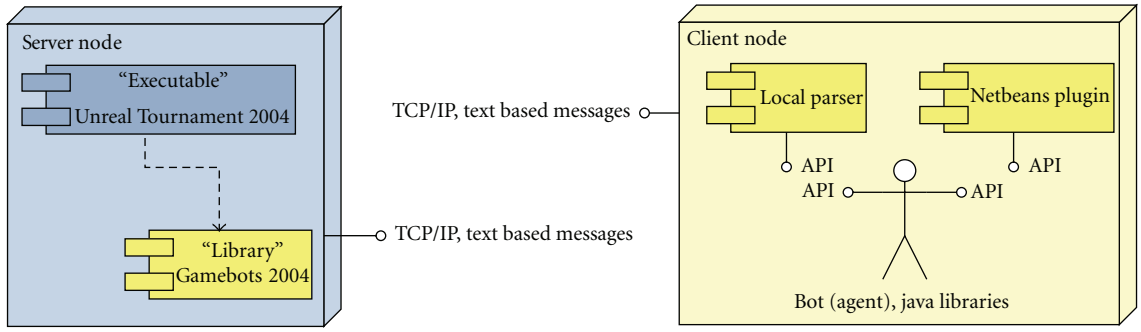

Figure 4: Pogamut architecture overview [3].

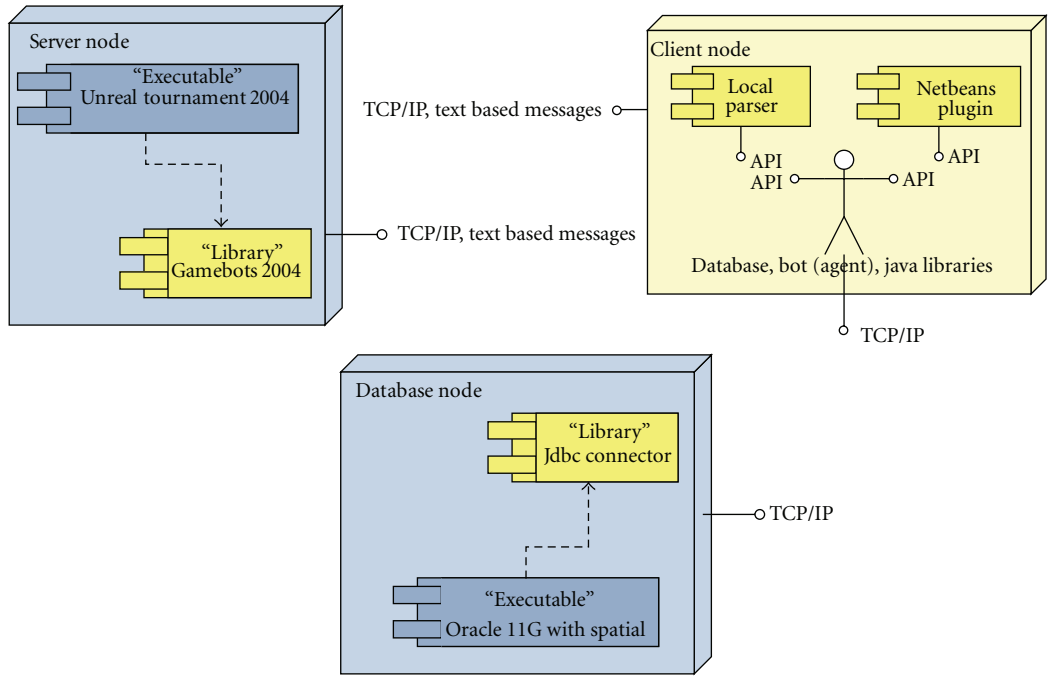

Figure 5: Pogamut database component overview.

Figure 4, integrates the UT2004 Server through the GB2004 API with the Client and Netbeans IDE.

The Pogamut architecture consists of four components: (1) GB2004 and (2) Parser and (3) Client and (4) IDE. To support the realism metric, the basic Pogamut architecture was extended with a database component as shown in Figure 5. This added component provided storage and access to recorded data for both bot and human behaviors for use in calculating the realism metric. The database allows for the additional asynchronous and synchronous processing of game information required to control bot actions and offload processing from the UT2004 game engine and Client. In addition to the database, the Pogamut Experiment Class in the IDE was modified to connect with the database and provide for human player monitoring on the UT2004 server.

3.2. GameBots 2004. GB2004 is programmed in UnrealScript and makes the UT2004 environment available to the Client/IDE which runs the bot through libraries and the TCP/IP-based API. It defines a text protocol that the client must implement to successfully run a bot in the UT2004 environment. This protocol consists of commands (sent from Client to GB2004) and messages (sent from GB2004 to Client). Commands are used to control bot actions. Messages serve to acknowledge the command and transmit information about events (asynchronous messages) or about state of the game (synchronous messages).

Communication between GB2004 and Client/IDE is based on TCP/IP. The bot is run through the Netbeans IDE and its Pogamut plug-in. This off-loads the bot's use of system resources to the IDE; thereby ensuring game engine performance is not confounded by bot execution.

Leveraging the concepts introduced in work on behavior believability in video games [16], GB2004 was modified to extract from the UT2004 game engine the same run time behavior data as are extracted for bots. Java classes were also added to extend the Pogamut agent class API libraries to allow a human player to communicate with GB2004 and

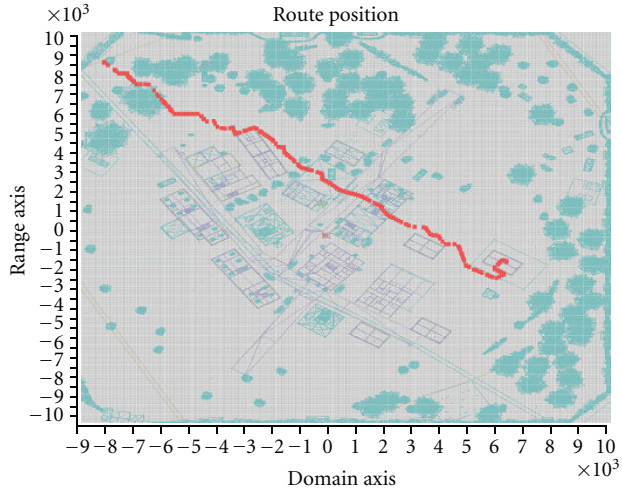

Figure 6: Bot A* route in virtual McKenna.

record the human player behavior data. Access to human player behavior data in the simulation environment is a critical prerequisite to the validation of the realism of bot actions.

3.3. Parser and Mediator. The Pogamut Parser translates text messages to Java objects and implements compression for transmission between GB2004 and the Client/IDE. The Pogamut parser class generates two threads for each bot. The first thread implements the communication from GB2004 to the bot (messages for the bot) and the second one implements the communication from the bot to the GB2004 (commands to the bot in UT2004). The Pogamut Mediator class creates the listener between the parser and client for detecting and delivering messages from parser to client and vice versa. The mediator recognizes the GB2004 protocol allowing communication with the UT2004 and GB2004.

To provide for player monitoring and human action simulation, the Pogamut Parser class was modified to generate two additional threads. The first thread implements the equivalent communication from the modified GB2004 to the human player objects within UT2004 (messages for the human player) and the second implements the communication from the UT2004 human player objects to the GB2004 (commands to the human player Objects in UT2004). This modification does not interact with the players themselves, only the human player Objects within the UT2004 game engine through the GB2004 human player modifications. To ensure the integrity of the human player actions within the game, these threads only serve to collect and provide to the IDE human player data.

A minor modification to the Pogamut Mediator class in the Java API used in the IDE was required for communication with the human player. A supporting modification was made to the GB2004 messaging interface.

3.4. Bot Client/IDE. The Bot Client/IDE consists of the Pogamut libraries and APIs which are used in the development of the bot. The libraries and APIs manage the communication between the bot's logic and the UT2004 server through the Parser and Mediator as described above. The Bot Client/IDE also uses the information received from the Parser through the Mediator to build a world model for each bot executing in the game and updates the model with information received from the bot. The parameters that describe this world model are stored in a database whose access process executes in its own thread to avoid impacting the bot logic thread during execution. The database also provides for asynchronous data analysis to update the game model near real time. This allows the bot logic access to a game model that more closely mimics what is available to the human player so as to more closely mimic human player actions.

The Bot Client/IDE provides these services to bots' behavior control logic:

(i) communication with Parser via Mediator,

(ii) map representation,

(iii) bot memory,

(iv) inventory,

(v) access for logic,

(vi) database access.

3.5. Experiment Client. The development of a realism metric required a new client capable of interfacing with both bot and human player data in UT2004 as well as the world model. Using the modifications made to the Pogamut libraries and GB2004, we developed the Experiment Client. The Experiment Client runs in a Netbeans IDE to provide an IDE for development of metric experiments to record route data. We used the bots' execution of the A* algorithm for route planning and execution as the test case.

As with the Bot Client/IDE, the Experiment Client consists of the modified Pogamut libraries and APIs described above. In addition to the modifications made to support human player behavior data recording, the Pogamut Experiment Class was extended to support a threaded database and JChart interface. The database provides storage for data analysis and bot action input. This enhancement allows real-time animation of bot or human player routes during scenario execution, as illustrated in Figure 6. An example of off-line data analysis for realism measurement is shown in Figure 7.

Unlike the Bot Client/IDE, the Experiment Client does not execute any bot or game logic. This allows all bot and game logic to be executed on distributed clients, which prevents the data recording from impacting bot game performance. The Experiment Client generates bot and human player Agents running in threads during execution.

These agents inherit all the bot and human player attributes with the exception of bot logic and human player commands. The attribute inheritance exposes the bot and human player behavior data without violating the game engine's anticheat code or exposing the data to the bot or human player during execution. The Experiment Client also uses the information received from the Parser through the

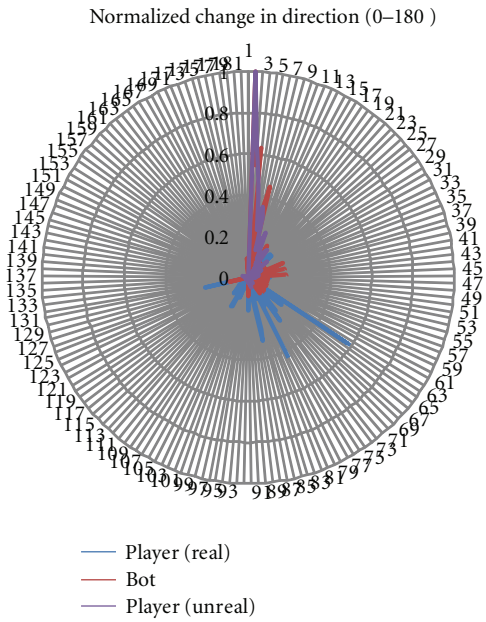

Figure 7: Radar chart of normalized heading change metric.

Mediator for each bot and human player, and the world model generated by the Bot Client or in the absence of a bot, by the Experiment Client. The Experiment Client interface to the database also provides for asynchronous data analysis to update the game model near real time in the absence of a bot. The interface also allows implementation of metrics in the database for real-time and off-line analysis and display.

The Experiment Client provides the following services:

(i) communication with the Parser through the Mediator,

(ii) map representation,

(iii) access to bot and human player memory,

(iv) game inventory (weapons for both human player and bots),

(v) access for experiment and metric logic,

(vi) database access.

3.6. Database with Spatial Libraries. The database node forms the final component of the simulation environment. For this simulation environment, we used an Oracle 11 g database with its spatial option. The Oracle 11 g database provides two important functions. First, it provides for the behavior data storage and subsequent quantitative analysis of the realism metrics collected through a database thread implemented in the Pogamut bot (agent) interface with GB2004 and UT2004 server. Second, a dynamic world model is maintained within the database to provide near real time access by the bot's logic to more closely mimic human actions based on the changing game world environment. In addition, we used the spatial module to develop a network model of the UT2004 world model collected during Bot Client or Experiment Client initiation. The network model allows for real-time modification to the edges for the UT2004 generated search graph during game play and subsequent updates to the A* routes for the bots (Figure 8).

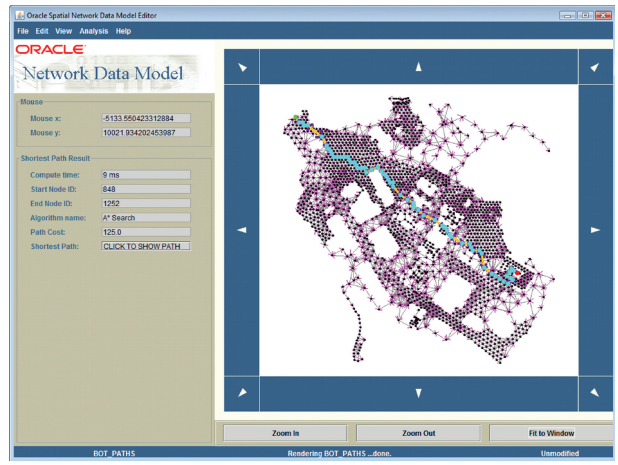

Figure 8: UT2004 network model A* route.

4. Measuring Realism

The previous section defined the game engine modifications required to support the collection of environmental (map or level) world model and player and bot behavior data within the virtual environment. This section describes the development of the virtual environment necessary to support bot and player behavior data collection and a methodology for using those data to calculate the realism metric.

4.1. Physical and Virtual Environments. In addition to the UnrealScript GB2004 and Java API modifications that permit access to the physics and artificial intelligence components of the game engine, Bot Client, and Experiment Client described above, the UT2004 editor provided the interface to the virtual world through which both the bot and human player actions could be programmed and added to the environment. In addition, the virtual space had to physically exist and have the capacity to record human actions and behavior. The McKenna facility was found to meet these requirements. McKenna is fully instrumented for recording training scenarios and has been modeled in the UT2004 MODS America's Army and Ravenshield. While not trivial, the McKenna Ravenshield model was ported to UT2004 and the GB2004 UnrealScript components required for player and bot monitoring added. The accuracy and realism of the bots' and players' actions could then be measured for a quantitative analysis of their realism. Finally, the modeling of a physical location in the virtual world will allow future with truth (recorded human execution of defined scenarios and actions) for validation of realism metrics.

There are several characteristics and properties that the physical and virtual environment must possess for realism to be accurately measured and quantitatively compared.

```
/**
* New Pogamut Class for Storing Player Parameters during game play
* Class to Create MonitoredPlayerBody for storing GB2004 Messages
* @author fabien tence initial code
* @author gregg hanold extended for player and A-star
*/
public class MonitoredPlayerBody extends AgentBody {
    public MonitoredPlayerBody(Logger logger, String playerUnrealID) {
        helloMessage = MessageType.HELLO_MONITORING;
        knownObjects = new KnownObjects();
        platformLog = logger;
        for (MessageType type: MessageType.values()) {
            this.typeMessageListeners.put(type,
                new ArrayList<RcvMsgListener>()); }
        initializer = new MonitoringInitializer(playerUnrealID); }}
```

Algorithm 2: MonitoredPlayerBody class listing.

(i) The virtual world must provide the same type of activity using the same methods as the physical world.

(ii) The virtual world must contain the same number and types of objects (actors) as the physical world.

(iii) The virtual world must convey the same level of detail that would be encountered in the physical world and change in a manner that is believable, realistic, and faithfully mimics the physical world.

The virtual world must mimic the experience and milieu of the physical world and present the user with an experience that is unpredictable and nonscripted in addition to being believable and realistic [17].

4.2. Data Recording and Measurement. A significant challenge in producing quantitative measures was developing an interface to the UT2004 server that could record the same data from both the bot and human players executing a scenario in the virtual world and humans in the real world. To accomplish this, GB2004 UnrealScript and the Pogamut Core Java Libraries were modified to provide for advanced player messaging and a new database thread integrated with Experiment Client. The listings for the Player Monitoring classes are shown in (Algorithms 2 and 3). In addition, an input mechanism to display and compare in the virtual world route data recorded in the physical world was developed.

Bot and human player monitoring functions occur in separate executables. Bot monitoring is executed as part of the bot spawning. To initiate player monitoring, a Pogamut Experiment must be executed. The Main listing for executing player monitoring is shown in (Algorithm 4). The message listener was added to an extension of the AgentBody class to interface with GB2004. Both bot and player monitoring implement a database thread through a Java dbConn class that constructs a jdbc connector with select, insert, update, and delete methods for an Oracle 11 g database. The dbConn class was constructed during the bot and player monitoring initialization and is populated with the map representation, bot and player memory, and object inventory. It is updated with each doLogic() logic cycle. The doLogic() override for the player monitoring logic cycle is given in (Algorithm 5).

The simulation architecture called for an Experiment Client to execute the monitoring and recording functions of both bot and human players. This client runs on its own dedicated computer which offloads the processing from the bot and UT2004 clients. Distribution of this processing makes possible the simultaneous collection of both bot and human player behavior data without impacting the UT2004 game engine, player, or Bot Client performance. The Experiment Client makes possible near real time recording and analysis of both bot and human player route data.

4.3. Methodology. The proposed general methodology for measuring the realism of algorithm-generated bot behavior, such as route planning and following, has seven steps.

(1) *Create scenarios.* Create a scenario (or set of scenarios) that includes the behavior for which a realism measurement is desired and that can be executed by both bots and human players in the virtual environment.

(2) *Identify data.* Determine the virtual environment data variables that describe the behavior as it is executed at run time.

(3) *Execute with humans in virtual environment.* Execute the scenario in the virtual environment with human players, recording the behavior data during execution.

(4) *Execute with bot in virtual environment.* Execute the scenario in the virtual environment with a bot, recording the behavior data during execution.

(5) *Execute with humans in real world.* If possible, execute the scenario in the real world with humans, recording the behavior data during execution.

(6) *Measure realism.* Calculate the realism metric values for the behaviors of the bots and human players in the

```
/**
* New Pogamut Class for Monitoring Player Parameters during game play
* Class to Create MonitoredPlayer Instance and tie to GB2004 Messages
* @author fabien tence initial code
* @author gregg Hanold extended for player and A-star
*/
public class MonitoredPlayer extends Agent {
    private String unrealID;
    public MonitoredPlayer(String unrealID) {
        this.unrealID = unrealID; //needed in initBody can't call super
        Agent.instancesAlive + =  1;
        // init of compounds
        initLogging();
        initBody(); // preceeds memory, memory needs to register listener in body
        initMemory();
        initGameMap(); // must be called as the last item !
        // uses reference to the memory, body ( For A* search must know
        //where the agent is)
        this.body.addTypedRcvMsgListener(this,
            MessageType.MAP_FINISHED); }
    @Override
    protected GameBotConnection initGBConnection(URI gameBots)
        throws UnknownHostException {
        return new MonitoringConnection(gameBots.getHost(),
        gameBots.getPort() ==  −1 ? 3003: gameBots.getPort()); }
    @Override
    protected void initBody() {
        this.body = new MonitoredPlayerBody(platformLog, unrealID);
        this.body.addRcvMsgListener(this); //GB20004 Messaging
        this.body.exceptionOccured.addListener(
            new FlagListener<Boolean>() {
                public void flagChanged(Boolean changedValue,
                    int listenerParam) {
                if (changedValue) {
                    platformLog.severe("Exception occured, stopping agent.");
                    exceptionOccured = true; }}} );
}}
```

Algorithm 3: MonitoedPlayer class listing.

virtual environment, and if available, for the humans in the real world.

(7) *Assess realism.* Compare the values of the realism metrics. Using a suitable statistical hypothesis test, determine if the realism metric values for the human players in the virtual environment and the humans in the real world are distinguishable from those of the bot in the virtual environment. The closer the bot's realism metric values are to the human's values, the more realistic the algorithm-generated behavior is.

4.4. Testing. To test the enhanced UT2004 simulation environment, it was used to collect route data from a group of volunteer human players who were persons attending the 2010 Huntsville Simulation Conference [18]. The data collection process exercised both the enhanced simulation environment and five of the seven steps of the methodology; the two unexercised steps remain for future work, as detailed in Section 5. The testing process is described within the framework of the methodology's steps.

(1) *Create scenarios.* Because possible differences in routes resulting from differing amounts of terrain information available were of interest, three scenarios, denoted A (no terrain knowledge), B (partial terrain knowledge), and C (complete terrain knowledge), were developed. All three scenarios required route planning and following, the realism of which was the subject of the test.

(2) *Identify data.* The data variables needed to measure the realism of routes were determined to be location, heading, and velocity. Sources for these data were found in the game engine.

(3) *Execute with humans in the virtual environment.* The volunteer human players were told that their objective was to navigate within the virtual environment as they would in the physical world. Because the objective of this test was measuring route realism and not actual game play, each human players was given an introduction and demonstration of the

```
public class Main extends Experiment {
    Vector<MyMonitoring> monitoredPlayers;
    private boolean isMonitoring;
    public Main(ExperimentDescriptor descriptor, UTServer server) {
        super(descriptor, server); }
    @Override
    protected void stageOneInit() throws Exception {
        monitorPlayingPlayers(); }
    @Override
    protected void stageOneInit() throws Exception {
        getLogger().info("Experiment started.");
        if (isMonitoring) {
            createControlPanel(); //Small panel to stop the experiment
            //Wait for all monitored player to stop playing (or an error occurs
            //in the monitoring)
            Iterator<MyMonitoring> iter = monitoredPlayers.iterator();
            while (iter.hasNext()) {
                iter.next().getSemaEndMonitor().acquire(); }}
        setResultAndTerminate(new ExperimentResult()); }
    private void monitorPlayingPlayers() throws PogamutException {
        Set<Player> players = getServer().getInfo().getPlayers();
        if (players != null && !players.isEmpty()) {
            Iterator<Player> iter = players.iterator();
            monitoredPlayers = new Vector<MyMonitoring>();
            while (iter.hasNext()) {
                Player player = iter.next();
                MyMonitoring theMonitoredPlayer =
                        new MyMonitoring(player.UnrealID);
                getServer().monitorPlayer(theMonitoredPlayer);
                monitoredPlayers.add(theMonitoredPlayer);
                isMonitoring = true; }}}
    @Override
    protected void stageThreeFinish() {
        getLogger().info("Experiment finished.");
        if (isMonitoring) {
            Iterator<MyMonitoring> iter = monitoredPlayers.iterator();
            while (iter.hasNext()) {
                MyMonitoring plr = iter.next();
                plr.disconnect(); }}}
    private void createControlPanel() {
        final JFrame frame = new JFrame();
        JButton stopButton = new JButton("Stop");
        stopButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                Iterator<MyMonitoring> iter = monitoredPlayers.iterator();
                while (iter.hasNext()) {
                    MyMonitoring plr = iter.next();
                    plr.saveData();
                    plr.getSemaEndMonitor().release();}
                    frame.dispose(); }});
        frame.add(stopButton);
        frame.setPreferredSize(new Dimension(200, 75));
        frame.setVisible(true);
        frame.pack(); }
    public static void main(String[] args) {}}
```

Algorithm 4: Player monitoring Main class listing.

```
// Player doLogic() parameter recording segment
//Overrides Experiment doLogic in Pogamut Library to add db recording
   @Override //doLogic() to capture player movement and position
   protected void doLogic() throws PogamutException {
      long time = System.currentTimeMillis(); //Time of current execution
      if (time − lastSaveTime > 60000) {
      dbConn.Insert ("tbl_PParams",time, vel, dir, loc); //Update Player Table
         lastSaveTime=time; } //Set data recording interval
      totalTS++;
      Triple vel = getMemory().getAgentVelocity(); //Player velocity
      Triple dir = getMemory().getAgentRotation(); //Player movement direction
      Triple loc = getMemory().getAgentLocation(); //Player postion
      if (vel.x != 0 || vel.y != 0) { //the player is moving
         wasImmo = false;
      } else { //the player is not moving
         if (wasImmo) {
            consecImmoTS++; }
         wasImmo = true;
         immoTS++; }
      . . .
   }
//BOT doLogic() parameter recording segment
   protected void doLogic() throws PogamutException {
      long time = System.currentTimeMillis(); //Time of current execution
      if (time − lastSaveTime > 60000) {
      dbConn.Insert ("tbl_BParams", time, vel, dir, loc); //Update BOT Table
         lastSaveTime=time; } //Set data recording interval
      totalTS++;
      Triple vel = memory.getAgentVelocity(); //BOT velocity
      Triple dir = memory.getAgentRotation(); //BOT movement direction
      Triple loc = memory.getAgentLocation(); //BOT position
      if (vel.x != 0 || vel.y != 0) { //the BOT is moving
         wasImmo = false;
      } else { //the BOT is not moving
         if (wasImmo) {
            consecImmoTS++; }
         wasImmo = true;
         immoTS++; }
      . . .
         action_run_roamAround(); //BOT Path following Navigation command
}
```

Algorithm 5: Player monitoring dologic () listing.

UT2004 navigation controls available: stop, walk, run, jump, crouch, and turn. All other UT2004 game-related controls were turned off. The human players were provided a period of time to familiarize themselves with the game controls and to ask the experimenter any questions related to the scenario. The familiarization period and the disabling of nonmovement controls were intended to eliminate the effects of player familiarity with the game engine on the recorded route data.

All of the human players from whom data was collected were unfamiliar with the specific terrain area before the experiment. The players were randomly assigned to one of the three scenarios and were given terrain information consistent with the scenario.

For scenario A (no terrain knowledge), the human players were given the following briefing: "You are to locate and enter a building with a chain link fence surrounding it. The fence will have a single gate. The building is green with a black roof, a single door, and windows on all sides. Once in the building you are to enter the room with two windows, one above the other. The lower window will have its view obstructed by a jersey barrier. You will be given a few minutes to familiarize yourself with the UT2004 Client controls, while the database is initialized for collection." No map was provided to the human player.

For scenario B (partial terrain knowledge), the human players were given the same briefing as in Scenario A. The map in Figure 6 was provided to them, showing the starting and destination points, but no routes.

For scenario C (complete terrain knowledge), the human players were given the same briefing as in Scenario A. As with

Table 1: Sample route data recording.

Time stamp	X-Coordinate	Y-Coordinate	Z-Coordinate	Velocity-X	Velocity-Y	Velocity-Z
1288187592014	−8144.53	8358.18	−605.41	100.79	−68.13	0.00
1288187592264	−8047.99	8292.91	−603.25	364.52	−246.42	0.00
1288187592514	−7957.77	8231.93	−601.60	364.53	−246.41	0.00
1288187592796	−7852.70	8160.07	−600.03	357.64	−256.31	0.00
1288187592999	−7758.46	8078.95	−597.98	324.40	−297.26	0.00
1288187593249	−7670.81	7989.45	−595.45	288.14	−332.54	0.00
1288187593499	−7610.44	7912.54	−592.89	269.10	−348.13	0.00
1288187593749	−7536.66	7814.22	−589.56	258.49	−356.07	0.00

scenario B, the map in Figure 6 was provided, showing the starting and destination points, but no routes. During the scenario execution, a display of the Figure 6 map was updated near-real-time with the player's current position.

The mechanism for recording the route data within the game engine has been described. Table 1 shows a small sample of the recorded route data for a typical route. Figure 9 illustrates typical routes for each of the three scenarios; in the figure they are identified as "None" (scenario A), "Partial" (scenario B), and "Complete" (scenario C).

(4) *Execute with bot in the virtual environment.* A bot generated and followed a route using the A* algorithm. Figure 9 also shows the bot route.

(5) *Execute with humans in the real world.* This step of the methodology remains for future work; see Section 5. Because the terrain used for the test models the real-world McKenna facility, which is regularly used for training, data may be available at some point in the future.

(6) *Measure realism.* As of this writing, the route realism metrics are currently under development. Several candidate metrics are under development, including (1) the ratio of route length to straight line distance from the starting point to the ending point, and (2) the normalized heading change for each route. Calculating the former is straightforward; calculating the latter proceeds as follows.

(i) Filter out all route segments in which the bot or human player was not moving.

(ii) Convert the individual route headings for each of the route segments, both bot and human player, to normalized values as $H_n = H - H_{sl}$, where H_{sl} is the straight line heading from start to destination; all negative values are converted to positive by adding 360.

(iii) Calculated the heading change for each route segment as $H_c(x) = |H_n(x-1) - H_n(x)|$, converting to a 0 to 180 degree scale to include both left and right heading changes under the assumption that a heading change of 1 to 179 degree was a right heading change and 181 to 359 degrees was a left heading change.

(iv) Compute the mean, standard error of the mean, and standard deviation of the heading changes for each of the record routes, both bot and human players.

The results of these calculations are summarized in Table 2 and Figure 10.

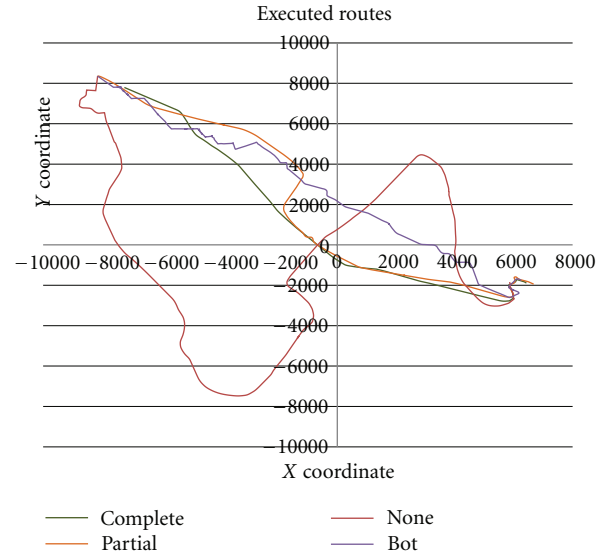

Figure 9: Typical routes collected during testing.

(7) *Assess realism.* This step of the methodology remains for future work; see Section 5.

5. Results and Future Work

The primary objective of this paper was to describe the enhancements made to a commercial game engine in order to collect data that might be used to measure and assess the realism of algorithm-generated behaviors, such as route planning. This section summarizes the results and future work.

5.1. Results. An implementation of an instrumented commercial game engine in which route realism can be quantitatively measured and a methodology for collecting route data needed for those measurements was presented. Analysis of data collected for bots and human players executing routes within the virtual environment will provide quantitative data required to develop route realism metrics, which will in turn be used to validate or assess the realism of algorithm-generated routes. UT2004 with the Pogamut Java Libraries and Netbeans IDE and GB2004 provided a suitable platform for the acquisition of the needed route data.

Table 2: Sample bot and human realism metric calculations.

Type	Variable	N	N*	Mean	SE_Mean	StDev
A-BOT	BOTHdg(D)	255	1	13.17	1.02	16.24
B-NK	P5Hdg(D)	484	1	6.06	0.84	18.52
B-NK	P6Hdg(D)	387	1	2.83	0.25	4.87
B-NK	P13Hdg(D)	397	1	1.07	0.21	4.23
B-NK	P16Hdg(D)	412	1	5.93	0.79	16.03
B-NK	P17Hdg(D)	603	1	4.26	0.39	9.51
B-NK	P18Hdg(D)	564	1	4.29	0.37	8.68
C-Part	P7Hdg(D)	235	1	5.99	1.32	20.26
C-Part	P8Hdg(D)	451	1	1.60	0.34	7.25
C-Part	P10Hdg(D)	400	1	1.85	0.28	5.61
C-Part	P14Hdg(D)	515	1	1.96	0.28	6.24
C-Part	P15Hdg(D)	489	1	1.94	0.44	9.73
D-Comp	P3Hdg(D)	118	1	4.57	0.87	9.45
D-Comp	P4Hdg(D)	190	1	2.53	0.59	8.09
SL	SLhdg(D)	218	1	0.41	0.18	2.66

Where type is defined by: A-BOT: BOT, B-NK: Human Player No Knowledge, C-Part: Human Player Partial Knowledge, D-Comp: Human Player Complete Knowledge. Variable represents the test run for BOT-BOT Heading, Pxx-Player xx Heading. SL-Straight Line Heading. N: Number of sample points. N*: Number of missing sample points. Mean: Mean of the Variable Heading in Degrees. SE: Mean-Standard Error of the Mean. StDev: Standard Deviation.

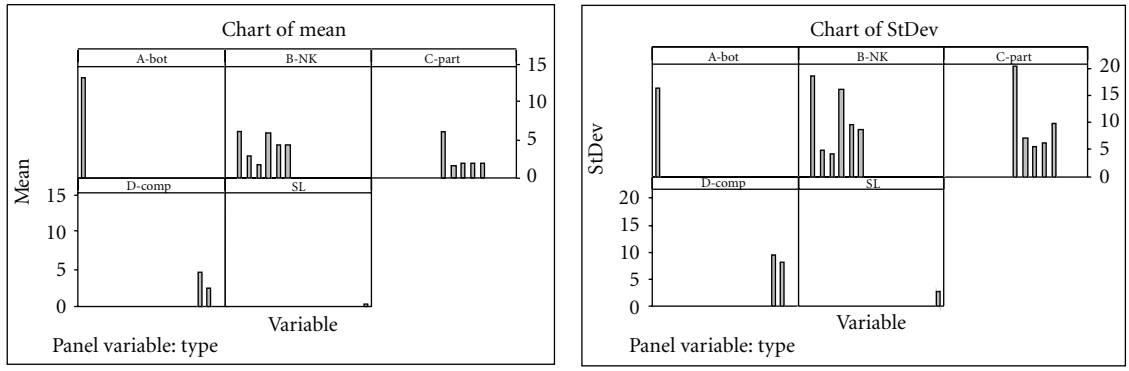

Figure 10: Plot of mean and standard deviation of normalized heading change metric.

The enhancements to UT2004 are specific to that game engine, but they suggest what may be required in another game engine to collect similar data. The overall methodology is applicable to any game engine with a suitable API.

5.2. Future Work. The next phase of this research will focus on the development of a quantitative realism metric based on route data that can be applied to any given route to measure its realism. Validation of the realism metric will be performed via comparisons of realism metric values for routes followed by algorithms and humans executing the same scenarios in the virtual environment. Statistical hypothesis tests will be used to demonstrate that the realism metrics reliably distinguish between algorithm-generated and human-generated routes.

Based on lessons learned from the data collection testing, an improved set of scenarios will be used in the next phase of the research. Four human player scenarios will be used.

(1) *No terrain knowledge.* The player is given no information before route planning and execution.

(2) *Incomplete terrain knowledge.* The player is shown a map of the terrain with starting and destination points marked before planning and execution.

(3) *Incomplete terrain knowledge with preplanned route.* The player is shown a hardcopy map of the terrain with starting and destination points marked before route planning and execution. He/she is given the opportunity to plan a route on the map before starting route execution; the player retains the map during route execution.

(4) *Complete knowledge of the terrain with preplanned route.* The player is shown a hardcopy map of the terrain with starting and destination points marked before route planning and execution. He/she is given the opportunity to plan a route on the map before starting route execution; the player retains the map during route execution. During route execution, the player is shown his/her current position plotted in near real time on an animated terrain map.

Table 3: Experimental design for human player data collection.

Players	Map	Start	Destination	Category	Player	Map	Start	Destination	Category
1	1	1	1	1	64	2	1	1	1
2	1	1	2	1	63	2	1	2	1
3	1	1	3	1	62	2	1	3	1
4	1	1	3	1	61	2	1	3	1
5	1	1	1	1	60	2	1	1	1
6	1	1	2	1	59	2	1	2	1
7	1	1	3	1	58	2	1	3	1
8	1	1	4	1	57	2	1	4	1
9	1	1	1	1	56	2	1	1	1
10	1	1	2	1	55	2	1	2	1
11	1	1	3	1	54	2	1	3	1
12	1	1	4	1	53	2	1	4	1
13	1	1	1	1	52	2	1	1	1
14	1	1	2	1	51	2	1	2	1
15	1	1	3	1	50	2	1	3	1
16	1	1	4	1	49	2	1	4	1
17	1	2	1	2	48	2	2	1	2
18	1	2	2	2	47	2	2	2	2
19	1	2	3	2	46	2	2	3	2
20	1	2	3	2	45	2	2	3	2
21	1	2	1	2	44	2	2	1	2
22	1	2	2	2	43	2	2	2	2
23	1	2	3	2	42	2	2	3	2
24	1	2	4	2	41	2	2	4	2
25	1	2	1	2	40	2	2	1	2
26	1	2	2	2	39	2	2	2	2
27	1	2	3	2	38	2	2	3	2
28	1	2	4	2	37	2	2	4	2
29	1	2	1	2	36	2	2	1	2
30	1	2	2	2	35	2	2	2	2
31	1	2	3	2	34	2	2	3	2
32	1	2	4	2	33	2	2	4	2
33	1	3	1	3	32	2	3	1	3
34	1	3	2	3	31	2	3	2	3
35	1	3	3	3	30	2	3	3	3
36	1	3	3	3	29	2	3	3	3
37	1	3	1	3	28	2	3	1	3
38	1	3	2	3	27	2	3	2	3
39	1	3	3	3	26	2	3	3	3
40	1	3	4	3	25	2	3	4	3
41	1	3	1	3	24	2	3	1	3
42	1	3	2	3	23	2	3	2	3
43	1	3	3	3	22	2	3	3	3
44	1	3	4	3	21	2	3	4	3
45	1	3	1	3	20	2	3	1	3
46	1	3	2	3	19	2	3	2	3
47	1	3	3	3	18	2	3	3	3
48	1	3	4	3	17	2	3	4	3
49	1	4	1	4	16	2	4	1	4
50	1	4	2	4	15	2	4	2	4

Table 3: Continued.

Players	Map	Start	Destination	Category	Player	Map	Start	Destination	Category
51	1	4	3	4	14	2	4	3	4
52	1	4	3	4	13	2	4	3	4
53	1	4	1	4	12	2	4	1	4
54	1	4	2	4	11	2	4	2	4
55	1	4	3	4	10	2	4	3	4
56	1	4	4	4	9	2	4	4	4
57	1	4	1	4	8	2	4	1	4
58	1	4	2	4	7	2	4	2	4
59	1	4	3	4	6	2	4	3	4
60	1	4	4	4	5	2	4	4	4
61	1	4	1	4	4	2	4	1	4
62	1	4	2	4	3	2	4	2	4
63	1	4	3	4	2	2	4	3	4
64	1	4	4	4	1	2	4	4	4

Two maps will be used, each with four starting and ending points, giving 16 possible combinations per map. Each human player will execute one scenario on each of the two maps. Table 3 summarizes the resulting experimental design. The minimum number of human required is 64.

If the necessary data are available, the routes followed in the virtual environment will also be compared with routes followed by humans moving in the real world. To allow for this possibility, the virtual environment used for data collection was a recreation of a real location, the Fort Benning McKenna MOUT facility. The routes of real soldiers in the McKenna facility would be compared to algorithm-generated and human-generated routes in the virtual environment.

Since the development of the UT2004 simulation environment, Epic Games has released the UT3 Engine and UnReal Developers Kit. The UT2004 environment should be ported to UT3 to take advantage of the additional engine access and behavior control available through this update. Finally, further development of the Pogamut Experiment classes to include parameter identification and statistical analysis should be explored.

Appendix

Selected Source Code

Selected source code excerpts for key parts of the implementation are provided. For more details, see Algorithms 1, 2, 3, 4, and 5.

Disclosure

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] P. Fua, "McKenna MOUT Site, Fort Benning, Georgia," 2009, http://www.ai.sri.com/~fua/rcvw/Benning.html.

[2] D. R. Scribner and P. H. Wiley, "The development of a virtual McKenna Military Operations in Urban Terrain (MOUT) site for command, control, communication, computing, intelligence, surveillance, and reconnaissance (C4ISR) studies," Tech. Rep. ARL-TR-4139, June 2007.

[3] M. Dorfler, "Pogamut 2 IDE for Agents in UT2004," 2009, https://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Architecture/.

[4] E. Beeker, "Potential error in the reuse of Nilsson's a algorithm for path-finding in military simulations," *Journal of Defense Modeling and Simulation*, vol. 1, no. 2, pp. 91–97, 2004.

[5] Z. Shen and S. Zhou, "Behavior representation and simulation for military operations on urbanized terrain," *Simulation*, vol. 82, no. 9, pp. 593–607, 2006.

[6] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Advanced Engineering Informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[7] D. A. Reece, M. Kraus, and P. Dumanoir, "Tactical movement planning for individual combatants," in *Proceedings of the 9th Conference on Behavior Representation in Modeling and Simulation*, pp. 301–308, Orlando, Fla, USA, May 2000.

[8] D. R. Van Brackle, M. D. Petty, C. D. Gouge, and R. D. Hull, "Terrain reasoning for reconnaissance planning in polygonal terrain," in *Proceedings of the 3rd Conference on Computer Generated Forces and Behavioral Representation*, pp. 285–306, Orlando, Fla, USA, March 1993.

[9] D. Brogan and N. Johnson, "Realistic human walking paths," in *Proceedings of the Computer Animation and Social Agents*, pp. 94–101, May 2003, New Brunswick, NJ, USA.

[10] R. G. Burgess and C. J. Darken, "Realistic human path planning using fluid simulation," in *Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation*, pp. 3–12, Arlington, Va, USA, May 2004.

[11] D. E. Diller, W. Ferguson, A. M. Leung, B. Benyo, and D. Foley, "Behavior modeling in commercial games," in *Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation*, pp. 257–268, Arlington, Va, USA, May 2004.

[12] M. D. Petty, "Verification, validation, and accreditation," in *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, J. A. Sokolowski and C. M. Banks, Eds., pp. 325–372, John Wiley & Sons, Hoboken, NJ, USA, 2010.

[13] G. Hanold and D. Hanold, "Route generation for a synthetic character (BOT) using a partial or incomplete knowledge route generation algorithm in UT2004," in *MODSIM World*, Virginia Beach, Va, USA, October 2009.

[14] G. Hanold and D. Hanold, "A method for quantitative measurement of the realism of synthetic character (BOT) actions within the UT2004 virtual environment," in *MODSIM World*, Virginia Beach, Va, USA, October 2009.

[15] A. N. Marshall, J. Vaglia, J. M. Sims, and R. Rozich, "Unreal Tournament Java Bot," 2006, http://sourceforge.net/projects/utbot.

[16] F. Tence and C. Buche, "Automatable evaluation method oriented toward behaviour believability for video games," in *Proceedings of the 9th International Conference on Intelligent Games and Simulation*, Valencia, Spain, November 2008.

[17] M. R. Stytz and S. B. Banks, "Considerations for human behavior modeling interoperability within simulation environments," in *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, Fla, USA, September 2006.

[18] G. T. Hanold and M. D. Petty, "Developing a modeling and simulation environment and methodology for the measurement and validation of route realism," in *Proceedings of the Huntsville Simulation Conference*, Huntsville, Ala, USA, October 2010.