ROUTE GENERATION FOR A SYNTHETIC CHARACTER (BOT) USING A PARTIAL OR INCOMPLETE KNOWLEDGE ROUTE GENERATION ALGORITHM IN UT2004

Gregg T. Hanold, Technical Manager
Oracle National Security Group
gregg.hanold@oracle.com
David T. Hanold, Animator
DavidHanoldCG
dhanold@comcast.net

Abstract. This paper presents a new Route Generation Algorithm that accurately and realistically represents human route planning and navigation for Military Operations in Urban Terrain (MOUT). The accuracy of this algorithm in representing human behavior is measured using the Unreal Tournament™ 2004 (UT2004) Game Engine to provide the simulation environment in which the differences between the routes taken by the human player and those of a Synthetic Agent (BOT) executing the A-star algorithm and the new Route Generation Algorithm can be compared. The new Route Generation Algorithm computes the BOT route based on partial or incomplete knowledge received from the UT2004 game engine during game play. To allow BOT navigation to occur continuously throughout the game play with incomplete knowledge of the terrain, a spatial network model of the UT2004 MOUT terrain is captured and stored in an Oracle 11g Spatial Data Object (SDO). The SDO allows a partial data query to be executed to generate continuous route updates based on the terrain knowledge, and stored dynamic BOT, Player and environmental parameters returned by the query. The partial data query permits the dynamic adjustment of the planned routes by the Route Generation Algorithm based on the current state of the environment during a simulation. The dynamic nature of this algorithm more accurately allows the BOT to mimic the routes taken by the human executing under the same conditions thereby improving the realism of the BOT in a MOUT simulation environment.

1. INTRODUCTION

Research on Human Behavior Representation (HBR) in synthetic agents (BOTS) has focused predominantly on Cognitive Modeling. A Cognitive Model attempts to represent human thinking or decision making and translate that to human action. The corollary to Cognitive Modeling is Behavior Generation, which we have defined as the representation of human behavior that mimics or emulates the human. We have found very little research which examines Human Behavior Modeling (HBM) in this context [2], [3].

The research indicates that a system today can be smart enough to give the illusion of life by concentrating on creating consistent believable high level behavior instead of natural looking human actions [1], [5]. The UnrealTournament™ 2004 (UT2004) game engine selected for this research, for example, provides an interface to the physics and AI components that generate the BOT behavior and actions that are used to implement the new route planning and navigation algorithm. The high level behavior elements received from the game engine form the inputs to the algorithm that plans and generates the routes that the BOT executes. BOT execution of the routes includes dynamic behavioral actions based on sensory information to better mimic the human thereby giving this illusion of life or realism as we have defined it. To further improve the realism of BOT route planning and navigation we introduce the concept of using partial or incomplete knowledge of the environment. This concept results in the dynamic calculation of routes based on sensory information and behavioral actions that more closely mimic those available to humans executing similar actions. To be realistic the behavior must mimic that of the human.

The simulation of Human Behavior for the purpose of measuring realism requires a virtual environment that can closely resemble that of the real world. The game industry has successfully achieved this goal with the Massively Multiplayer Online (MMO) games and First Person Shooter (FPS) Games such as Quake III Arena™, Half-Life2™, and Unreal Tournament's Americas Army™ Mod. As previously indicated, in this paper we develop a new route generation algorithm that is executed using Unreal Tournament 2004 (UT2004). The simulation of this new algorithm is accomplished using the Gamebots 2004 (GB2004) UnrealScript package, the Pogamut BOT (agent) and Java Libraries and its Netbeans plug-in, and an Oracle 11g spatially enabled database.

2. Background

Generating realistic human behavior in a virtual environment continues to challenge the simulation community. In recent years the explosion in game technology and advances in multi-agent systems and behavior representation in BOTs, make possible the ability to mimic human actions such as route planning and navigation. A key issue in the virtual environment that remains is how to generate human-like behaviors for BOTS. In recent years, game developers are contributing more and more effort on game artificial intelligence (AI), further supporting the importance of the need for simulating realistic human behaviors.[6] Real world events and the rising training costs in response to them has further shifted emphasis in the simulation community toward realism. This increased emphasis on realism suggests that AI-driven BOTs should be able to act as opponents against human players or as team members to cooperate with human players in the virtual environment. One well studied BOT action in this context is route planning.

Route planning, in general, is a well studied problem with a wide range of application areas, including artificial intelligence in games, robotics, and military simulation. While many algorithms exist for discovering and producing routes or paths, when the terrain can be represented as a graph, A* is arguably the most frequently used graph search technique. First described in 1968 [7], A* has been intensely studied and developed and now has several specialized forms. A* or one of its specialized forms is the basis of route planning in many computer games [8]. The route planning algorithm presented in this paper expands on the principles of A*.

3. SIMULATION ENVIRONMENT

The simulation environment used to develop and implement a new route generation algorithm satisfied several unique factors. First, the application program interface (API) should not introduce bias or confounding variables into the experimental design resulting from the virtual environment. Second, the API must allow for the collection, measurement and storage of game and environmental parameters without impacting game engine performance. Third, the API must support integration of the client application with the game engine physics and artificial intelligence (AI) engines. Finally, the virtual environment (map or level) must have an interface to allow physical parameters collected from humans executing defined scenarios in the physical environment to be input for statistical comparison. The Unreal Tournament 2004™ game engine with the Gamebots 2004 (GB2004) UnrealScript package and Pogamut BOT (agent) and Java Libraries and its Netbeans plug-in addressed these factors.

3.1 Unreal Tournament 2004 ™ (UT2004)

UnrealTournament (UT) was the first game to ship with synthetic agents or BOTS. UT provides a custom scripting language, UnrealScript, through which game developers can modify (MOD) the host game. UnrealScript provides a rich Object Oriented (OO) interface to the UT game engine producing MODS such as Ravenshield™ and Infiltration™. Other UT based games, such as America's Army™ and Vegas, extend the UT2004 game engine and lock or limit the ability to MOD through UnrealScript. With the rich OO interface and the availability of Pogamut Integrated Development Environment (IDE) with its Netbeans plug-in and BOT and Java Libraries, UT2004 was selected as the base game engine.

3.2 Gamebots 2004(GB2004)

Gamebots, an UnrealScript package, was jointly developed by USC and Carnegie Mellon University (CMU) as an interface between the server and client. The interface provides access to sensory information such as the location and direction of a player in the game world or a message received from a teammate through synchronous and asynchronous messages communicated between server and client. BOT action commands from client to server are also accessed through this interface. Andrew Marshall at USC-ISI created a higher-level interface based on the Gamebots protocol, called JavaBot API [Marshall, 2002] to handle the specific Gamebots protocol, network socket programming, message passing, and other related issues, which makes the development of BOT AI neater and simpler.

3.3 Pogamut

Expanding on the JavaBot API and extending the Gamebots' UnrealScript, Jakub Gemrot and Rudolf Kadlec developed the Pogamut plug-in to the Netbeans™ IDE (http://artemis.ms.mff.cuni.cz). The base Pogamut Architecture, shown in Figure 1, integrates the UT2004 Server through the GameBots 2004 (GB2004) API with the Client and Netbeans IDE.

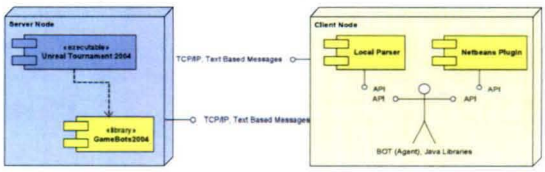

Figure 1 - Pogamut Architecture [4]

3.4 Oracle 11g Database with Spatial Data Objects

The basic Pogamut architecture integrated with UT2004 was extended with database functionality to permit advanced analytical processing of the environmental information available through the GB2004 and Pogamut interface. The Oracle 11g database provides two important functions. First, it provides for the parameter storage and subsequent analysis and retrieval based on BOT and Player sensory and action logic. Second, with the Spatial Data Objects (SDO), network and spatial analytics could be applied to the UT2004 map environment and collected during BOT initiation. Player/BOT monitoring functions added to the base GB2004 and Pogamut Core library provide near real time sensory and game parameters and spatially aware updates to the database. Figure 2 represents the simulation environment implemented.

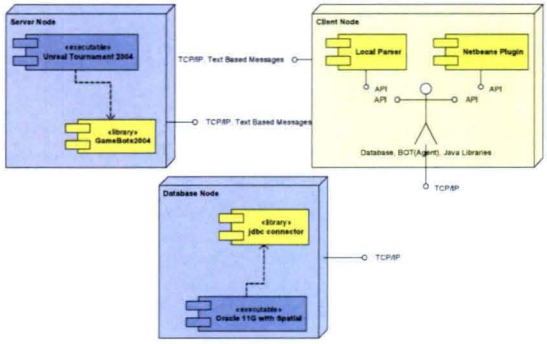

Figure 2 – Simulation Environment Architecture

4. ROUTE PLANNING ALGORITHM

Executing realistic route planning and navigation in the UT2004 game environment introduced several challenges. First, most game virtual environments (maps or levels) model the terrain as a graph by inserting pathnodes along navigable features such as urban streets, intersecting at distinct intersections, hall ways and stairs within buildings, and paths or trails in rural environments. A typical pattern is shown in Figure 3 for the McKenna MOUT virtual terrain used in this research. The UT2004 function to build AI Paths then computes the navigable edges between the nodes taking into account non-trafficable buildings, barriers and terrain. BOT logic programmed in the Pogamut Client instructs the BOT navigation AI to either execute a computed A* route to destination or to proceed to a specified pathnode. Because these pathnodes are predetermined during map design, do not change (i.e. are always reachable) and do not represent all paths the human can travel within the map, realism is not achieved. Second, UT2004 stores the pathnodes and the calculated edges in the map. BOT logic then retrieves this data during initialization and builds the navigation paths using A*, nearest neighbor or pathnode lookup. Because all the information is available, the routes a BOT plans are always perfect (i.e. shortest distance, shortest time, or least cost). Leveraging the graph available through the Pogamut interface to UT2004 and GB2004 and the Oracle 11g database with SDO, we developed a route planning algorithm which has at its core A*, but uses incomplete or imperfect knowledge in its execution. In addition to the SDO, the database makes available near real time dynamic information about the environment upon which the BOT logic can react. Thereby creating a more realistic BOT and addressing the challenges of the UT2004 environment.

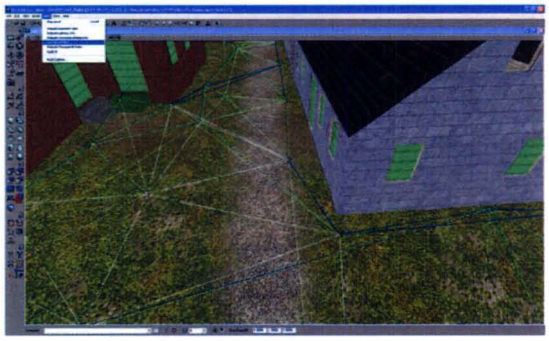

Figure 3 – Pathnodes with UT2004 edges

4.1 A* (A-star) Algorithm

A* is a best-first search algorithm that finds the least cost path from a given start pathnode to an end pathnode. Cost in A* is an attribute of the graph edges and pathnodes included in the path. The cost values are defined to correspond to a desired property of the real-world terrain the graph represents. In UT2004, the value of the cost is determined by the map designer. Later, in our realistic design we will show how to change this to allow dynamic calculation of this cost based on the environment.

As previously noted A*, is a well researched and documented algorithm. A good description and representation of the algorithm is found at: http://en.wikipedia.org/wiki/A*_algorithm. The pseudo-code is shown in Figure 4. Execution of A* begins with the identification of the start and the goal pathnode. Working partial paths are assembled from the start pathnode along the edges connecting surrounding pathnodes towards the goal pathnode. This process of following the edges from pathnode to pathnode is repeated, adding pathnodes to the partial paths until one reaches the goal pathnode. A heuristic function, denoted f(x), where x is the end pathnode of a partial path currently being considered, is used to

calculate the next pathnode of the graph to extend the path to.

$$f(x) = g(x) + h(x) \quad (4.1)$$

where: g(x) is the total cost of the partial path

h(x) is the distance from x to the goal

For A* to be optimal, the distance function h(x) must underestimate the distance to the goal. The Pogamut implementation of A* used in this research computes h(x) as the straight-line (Euclidean) distance to the goal pathnode, guaranteeing the underestimate (admissibility) criteria is met.

```
function A*(start,goal)
    closedset := the empty set    % The set of nodes already evaluated.
    openset := set containing the initial node % The set of tentative nodes to be
                                                evaluated.
    g_score[start] := 0    % Distance from start along optimal path.
    h_score[start] := heuristic_estimate_of_distance(start, goal)
    f_score[start] := h_score[start]  % Estimated total distance from start to goal
                                        through y.
    while openset is not empty
        x := the node in openset having the lowest f_score[] value
        if x = goal
            return reconstruct_path(came_from,goal)
        remove x from openset
        add x to closedset
        foreach y in neighbor_nodes(x)
            if y in closedset
                continue
            tentative_g_score := g_score[x] + dist_between(x,y)
            tentative_is_better := false
            if y not in openset
                add y to openset
                h_score[y] := heuristic_estimate_of_distance(y, goal)
                tentative_is_better := true
            elseif tentative_g_score < g_score[y]
                tentative_is_better := true
            if tentative_is_better = true
                came_from[y] := x
                g_score[y] := tentative_g_score
                f_score[y] := g_score[y] + h_score[y]
    return failure

function reconstruct_path(came_from,current_node)
    if came_from[current_node] is set
        p = reconstruct_path(came_from,came_from[current_node])
        return (p + current_node)
    else
        return the empty path
```

Figure 4 – Wikipedia's or the A* algorithm.

The Pogamut A* multiplies an edge cost value determined by the level or map designer with the edge length to calculate cost variable (g(x)). The value of the edge cost is always greater than or equal to 1, to ensure admissibility or the heuristic function, h(x).

4.2 Imperfect Knowledge Algorithm (IKA)

From the preceding discussion of A* two limitations to the generation of realistic routes stand out. First, A* guarantees an optimal path based on the heuristic, which in this case is distance. To mimic the route planning and subsequent navigation of humans, A* must be modified to account for additional variables, such as cover, terrain type, slope, doors, and windows. Second, A* calculations compute a-priori, a single optimal (based on the heuristic) path to the goal during BOT initialization. Updates to this path are computationally impractical during game play unless the number of pathnodes is minimized. Minimizing the number of pathnodes would unrealistically limit the possible BOT paths.

To solve these limitations, we first modify the A* heuristic function's cost variable g(x) to permit dynamic calculation of cost based on current game environment.

$$g'(x) = w_1 * g(x) + w_2 + w_3 + \ldots w_n \quad 4.2$$

where: w_1 is the length cost factor

$w_2 - w_n$ are cost factors determined from BOT, Player and environment

The modified heuristic function:

$$f'(x) = g'(x) + h(x) \quad 4.3$$

where $g'(x) > g(x)$

Second, we develop a mechanism for dynamic collection environmental parameters and limiting the available pathnodes for planning [9]. The dynamic collection of environmental parameters that include BOT, Player, and game parameters use the modified Pogamut architecture, which includes the database component and extension to the GB2004 UnrealScript Library and Pogamut Core Java Library for Player monitoring. The initial parameters ($w_2 - w_n$) used include Player Visibility, Door, Window, terrain type and the standard edge length. To limit the available pathnodes, a network model of the UT2004 pathnodes and calculated edges is constructed at BOT initialization using the Oracle 11g database with Spatial Data Objects. The network model and SDO permit expanding the pathnode and edge density to more closely approximate the possible paths a human might execute. To ensure timely IKA route calculation, retrieval of pathnodes and edges is accomplished using the SDO geometry function:

SDO_FILTER(p.geom, boundingbox) 4.4

where p.geom is the SDO geometry column of the NavPoints Table, and boundingbox is the SDO geometry of the area visible to the BOT

The listing in figure 5 returns the navigable nodes and edges contained within the boundingbox (SDO_BB(player)) modified pathnode pattern, shown in figure 6, for application in the IKA. The modified pathnode pattern ensures the possible paths available to the BOT are consistent with those available to the human player.

```
SELECT n.unrealid                  //returns unrealid of navigable points
  FROM navpoints n                 //navpoints table constructed at BOT init
  WHERE SDO_FILTER(n.geom,
    SDO_BB(player) ) = 'TRUE';     //SDO_BB(player returns Geometry of player
                                   // location
```

Figure 5 – SQL listing for bounded pathnodes.

In addition to the SDO_Geometry objects stored in the NavPoints table, we added the tables tbl_PParams and tbl_BParams to provide the structure for storing the BOT, Player and game parameters that are used in computing the dynamic cost function of the Imperfect Knowledge Algorithm (IKA). This data structure also permits

dynamic near real time updates to the BOT, Player, and game parameters used in the heuristic, through the monitored player class added to the Pogamut Core library. The monitored player class performs sense functions on human players and BOTS running in the game and updates tbl_PParams and tbl_BParams with the sense results.

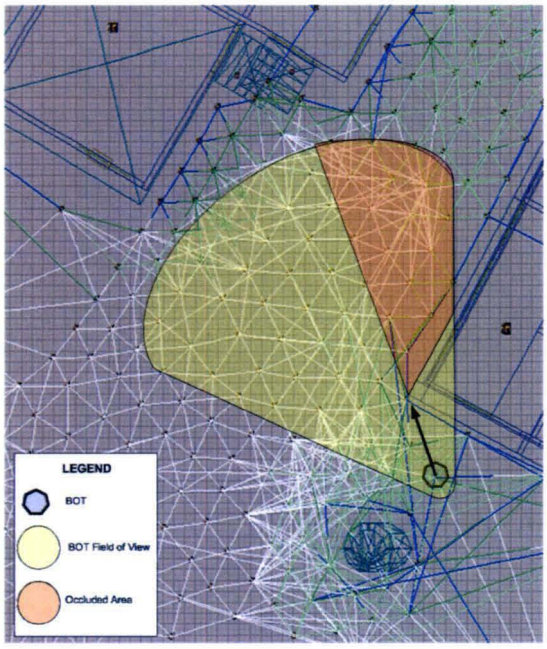

Figure 6 – UT2004 Network Model

The IKA requires additional modifications to the map or level design to identify specific terrain features such as doors, windows, and terrain type. UnrealScript is used to create additional pathnode actors that can be placed in the map or level to identify these features as well as the general planning pathnode that provides intermediate pathnodes from which the IKA will calculate the overview routes used to produce the IKA goal pathnodes. The concept of two pass route planning using UnrealTournament was introduced by Zhuoqian Shen to provide navigation through a multi-level (room) map [6].

```
// BotGPPathNode. General Purpose Node
class BotGPPathNode extends PathNode
        placeable;
// BotDestinationPathNode. Goal Node
class BotDestinationPathNode extends PathNode
        placeable;
// BotDoorPathNode. Door Node
class BotDoorPathNode extends PathNode
        placeable;
// BotWindowPathNode. Window Node
class BotWindowPathNode extends PathNode
        placeable;
```

Figure 7 – UnrealScript Listing for Pathnode extensions

As noted above, the IKA is implemented in two passes. The first pass is executed at BOT initialization to generate and store all goal pathnodes that will be used by the IKA. During initialization, all pathnodes and their neighbors, as computed by the UT2004, are stored in a SDO within the NavPoints table. The IKA then uses A* to compute the optimum path from the UT2004 pathnode, PlayerStartNode, to the pathnode, BOTDestinationPathNode, using only the BotGPPathNodes (Figure 8). Once the initial path has been generated as part of initialization, these points are used to determine the goal pathnodes for subsequent IKA path calculations.

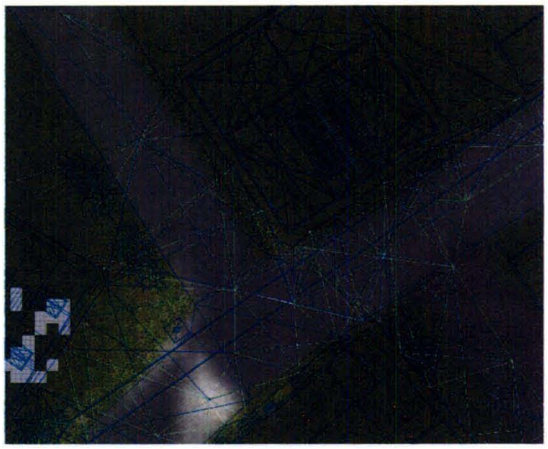

Figure 8 – BOTGPPathNodes Pathnodes

Subsequent passes execute the IKA using the dynamically generated cost variable g'(x) (4.2) and the modified heuristic function f'(x) (4.3). The IKA has built into the BOT's doLogic() function the sensory function that triggers route calculation, and when appropriate re-calculation of the IKA goal nodes. The pseudo-code for the IKA is shown in Figure 9.

5. MEASURING REALISM

The last section demonstrated the use of the UT2004 based simulation environment in the development of the Improved Knowledge Algorithm (IKA). The next step is to provide a quantitative approach to measuring its realism. Recall that in this paper we have defined realism to be a representation of human behavior that mimics or emulates the human. The UT2004 based simulation environment and the added monitor player class to the Pogamut Core Library and modifications to GB2004 provide the capability to record human player and BOT actions during a simulation run and store them in a data structure for subsequent numerical and statistical analysis. This feature will enable the development of a realism metric that measures the deviation of BOT actions from the human player. Combining the realism metric with the classic Turing test an objective and subjective assessment of the realism of the specific action being tested is possible.

```
function IKA (start, goal, pf)  % pf = pass flag (0 or 1)
   bpclosedset := the empty BotGPPathNode set
   bpopenset := set containing the initial BotGPPathNode
   closedset := the empty set    % The set of all pathnodes already evaluated.
   openset := set containing the initial node % The set of all tentative
                                              pathnodes nodes to be evaluated.
   g'_score[start] := 0    % Distance from start along optimal path.
   h_score[start] := heuristic_estimate_of_distance(start, goal)
   f'_score[start] := h_score[start]  % Estimated total cost from start to goal
                                        through y.
   if ps = 1        % Compute intermediate route using BotGPPathNodes
      closedset := bpclosedset
      openset := bpopenset
      w2[] := w3[] := w4[] := 0  % traditional A*
   else
      closedset := the empty set
      openset := set containing the initial node
   while openset is not empty
      x := the node in openset having the lowest f'_score[] value
      if x = goal
         return reconstruct_path(came_from,goal)
      remove x from openset
      add x to closedset
      foreach y in neighbor_nodes(x)
         if y in closedset
            continue
         tentative_g'_score := w1 * g_score[x] + w2[x] + w3[x] + w4[x] +
                               dist_between(x,y)
         tentative_is_better := false
         if y not in openset
            add y to openset
            h_score[y] := heuristic_estimate_of_distance(y, goal)
            tentative_is_better := true
         elseif tentative_g'_score < w1 * g_score[y] + w2[y] + w3[y] + w4[y]
            tentative_is_better := true
         if tentative_is_better = true
            came_from[y] := x
            g'_score[y] := tentative_g'_score
            f'_score[y] := g'_score[y] + h_score[y]
   return failure

function reconstruct_path(came_from,current_node)
   if came_from[current_node] is set
      p = reconstruct_path(came_from,came_from[current_node])
      return (p + current_node)
   else
      return the empty path
```

Figure 9 – IKA Psuedo-Code

6. CONCLUSIONS AND FUTURE WORK

The simulation architecture presented in this paper demonstrates an improved simulation architecture for development and testing of BOT actions. This simulation architecture was used to develop the Imperfect Knowledge Algorithm (IKA) which provides a more realistic representation of the BOT route planning action.

Continuation of this research will develop the realism metric, using the methodology described in this paper to provide a quantitative validation of the realism of the IKA.

The design and implementation of the virtual environment that supports the UT2004 based simulation architecture is very labor intensive. Functionality that can be implemented within the UnRealED to automate the terrain analysis for the placement of pathnodes and generation of the path edges should be studied and implemented.

In the current implementation of the IKA, the parameters that affect the route cost are manually set. A method for tuning the IKA parameters to optimize for realism as measured by the realism metric should be studied.

Finally, validation of the realism metric requires human data collected in a real world environment that has been modeled in UT2004. The Fort Benning, McKenna MOUT site was selected for this research to allow for this validation. Future research should be undertaken to collect human data executing a defined scenario that can be compared using the realism metric with human player and BOT data executing the same scenario in the virtual world.

REFERENCES

1. Beeker, E. (2004) "Potential Error in the Reuse of Nilsson's A Algorithm for Path-finding in Military Simulations", *JDMS*, vol. 1, no. 2, Apr, pp. 91–97
2. Brogan, D. & Johnson, N. (2003) "Realistic Human Walking Paths", *Proceedings of Computer Animation and Social Agents (CASA)*, pp. 94–101
3. Burgess, R. & Darken, C. (2004) "Realistic human path planning using fluid simulation", *Proceedings of Behavior Representation in Modeling and Simulation (BRIMS)*
4. Dorfler,M. (2007) "Pogamut 2 IDE for agents in UT2004", http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Architecture, last accessed 27 Jul 2009
5. Funge, J. et al, (1999) "Cognitive modeling: knowledge, reasoning and planning for intelligent characters", *Proceedings of the 26th annual conference on Computer graphics and interactive techniques, SIGGRAPH '99*", July
6. Shen, Z. & Zhou, S. (2006) "Behavior Representation and Simulation for Military Operations on Urban Terrain", *Simulation*, vol. 82, no. 9, Sep, pp 594-607
7. Nilsson, N. J. (1980). "Principles of Artificial Intelligence", Palo Alto, California: Tioga Publishing Company.
8. Beeker, E. (2004) "Potential Error in the Reuse of Nilsson's A Algorithm for Path-finding in Military Simulations". *JDMS* vol. 1, no. 2, April, pp 91-97
9. Hanold, G. & Hanold, D. (2009) "A Method for Quantitative Measurement of the Realism of Synthetic Character (BOT) Actions Within the UT2004 Virtual Environment", MODSIM 2009 World (Submitted)