5-11-2006

Distributed Computation in an Interactive Entertainment Environment

Robert Zinchak
Trinity University

Distributed Computation in an Interactive Entertainment Environment

Robert J. Zinchak

A departmental honors thesis submitted to the Department of Computer Science at Trinity University in partial fulfillment of the requirements for graduation with departmental honors.

April 11th, 2006

_____ _____

Thesis Advisor Department Chair

_____

Associate Vice President for
Academic Affairs

Distributed Computation in an Interactive Entertainment Environment

Robert J. Zinchak

Computer Science

Trinity University

1 Trinity Place

San Antonio, Texas 78212 USA

Faculty Advisor: Dr. Gerald Pitts

Abstract

The proliferation of multiplayer games has led to an increase in the total network capacity for processing power in games; this capacity, however, is seldom fully utilized or balanced. One prominent problem of distributed processing in a gaming environment is increased latency time-- which causes player disinterest in the game, potentially causing poor sales as well as the termination of future commercial development of this technology. Existing distributed techniques such as OpenMP,

MPI or VMPI are not well suited to gaming applications and may introduce additional overhead.

This thesis describes a simple, yet effective technique (based on existing ideas in both multiplayer network games and parallel processing), that can be implemented into existing networked games to allow for distributed computation-- which, in turn, will balance the work load across all connected machines while simultaneously decreasing visible latency to players.

Most multiplayer games use a client-server architecture, where the server acts as a referee between clients to manage the game state. A game world with many A.I. agents (simple non-playing characters) overloads the server and causes higher latency levels. Load may be balanced by allowing trusted clients to compute these actions. Development and maintenance costs can be kept low while not diminishing production quality, leading to a successful game. In addition to gaming applications, this technique can be incorporated in any existing client-server network application, including: communication, transportation, and process control.

This thesis investigates whether parallel processing offers a viable option for reducing latency and enabling larger, more realistic virtual worlds in multiplayer games. A comparison between the traditional client-server method and the distributed processing approach is provided. An example is created that simulates many A.I. agents in a larger N.I. (narrative intelligence) environment. Empirical tables and graphs shows the relationship between overhead and optimization this technique creates, illustrating whether parallel processing in networked multiplayer games deserves implementation.

Keywords: Interactive Entertainment, Parallel Processing, Narrative Intelligence, Games

Distributed Computation in an Interactive Entertainment Environment

Robert J. Zinchak

Acknowledgements

The author would like to thank first and foremost Dr. Gerald Pitts for years of generous support and advisement. The author would also like to thank Dr. Mark Lewis for his help, especially during the initial research of this project. Further thanks go to my thesis advisory board (Dr. Scott Baird, Dr. John Howland, Dr. Berna Massingill, and Dr. Gerald Pitts) and the faculty and staff of the Trinity University Computer Science department, all of whom have helped the author in some way on this project. Additionally, thanks go to Dr. Scott Baird for his assistance in editing, Mr. Adam Harwell who helped debug code and run alpha tests, and the seven anonymous players who took part in the beta testing. Finally, the author would like to thank his parents, Mr. John Zinchak and Mrs. Yvonne Zinchak, for their loving support.

Table of Contents

List of Tables

List of Figures

Introduction

Narrative Intelligence, hereafter referred to as NI, is the idea that a narrative (such as that used in electronic entertainment) reacts to the player's (or players') choices. In traditional narratives, the player is locked into a story such that despite any actual course of action the players want to take, there is only one predetermined path yielding a single ending. Sometimes this narrative has been expanded by offering multiple preset paths and endings, but ultimately the problem remains the same: players cannot truly affect the world in which they exist, leading to a sense of futility and a loss of immersion. Narrative Intelligence provides the players with the freedom to invent their own story, which will be unique every time they play. True immersion can only be felt in a world with hundreds or thousands of non-playing characters with which to interact. Simulating great numbers of these non-playing characters using AI agents (artificially intelligent entities) is a very complex task. In order to compute such large numbers of AI agents, new concepts in the simulation infrastructure must be introduced.

This thesis investigates one key new step that can help advance NI to greater sizes and therefore more immersive simulations.

Background

Research in NI spans back to the 1970's, when Roger Schank and his research team at Yale strove to understand how humans make sense of natural language. This quickly led to research in NI. Various programs were developed with varying goals: SAM[1] attempted to understand certain situations, while PAM and TAIL-SPIN[2] were goal-based systems. During the next decade, a sharp drop in NI research occurred because research funding for AI diminished, as graphical user interfaces and more applications for AI emerged later, interest in NI was re-kindled, leading to a tremendous growth in the body of NI research in this past decade [1].

The applications of NI in gaming are obvious because games often rely heavily on stories as a key element to their appeal. By greatly expanding these stories one can add more

[1] SAM (which interpreted stories) used scripts to understand stereotypical situations and the relationships within these situations [1].
[2] PAM (which interpreted stories) and TAIL-SPIN (which generated stories) were both goal-based, in the sense that characters had goals and steps to accomplish their goals [1].

challenge and fun to the game experience. NI provides a new way of challenging players. The initial challenge of a game is learning how the game works and the strategy to win. Throughout most games, difficulty is increased simply by throwing more enemies at the player or in some way making one of the existing challenges harder to overcome. This emphasizes the motor skills of the player and creates a conditioned response to game elements. Even as the difficulty of the game increases, players may, however, become disinterested in a challenge that only relies on their basic motor skills. Alternatively, if a player cannot match this challenge, he or she will become frustrated and stop playing the game. It makes sense, therefore, to look for challenges that are cognitively based instead of motor skills based. Such a challenge can be found in Narrative Intelligence. Due to the ever evolving NI world, players cannot develop conditioned responses to the environment, because each play session will be different from the last. Since the environment is free form, a player cannot “lose” this cognitive challenge. Many games are already trying to enhance their replay value by adding character customization or multiple preset endings, but

this does not significantly change the gameplay to create truly unique play sessions.

Several different directions in NI have emerged recently. Karl E. Steiner and Jay Tomkins at the University of North Texas [2], describe a system in which events, actions, and adaptations exist. An event in a story consists of a series of actions to describe that event to the players (such as animations and sounds) as well as adaptations that describe when this event can take place, what to do if the player is too far away to see the event, and other rules for the adaptation manager to follow. Steiner and Topkins' system allows a designer to describe a world narrative for a set situation, but only goes so far as the narrative's program allows. They suggest future research of a narrative generator which can generate new narratives dynamically.

Mark Owen Riedl and R. Michael Young of the Liquid Narrative Group of North Carolina State University [3] describe another technique that is also goal-based, but is heavily agent driven instead of narrative driven. Each agent has specific goals which may not correlate to any directly intended narrative, but

can be interpreted by the player as a narrative depending on which AI agents the player interacts with and what interactions take place.

In contrast to player-interpreted NI is a program called *Agent Stories* by Kevin M. Brooks of MIT Media Lab [4]. This project seeks to create logical narratives by analyzing narrative elements such as introduction, conflict, resolution, diversion, and ending. By analyzing how these elements are placed in relationship to each other, and classifying media clips as an element, *Agent Stories* can generate new narratives.

It should be noted that many other projects have sought to create Narrative Intelligence as well as the ones described above. A complete description of all projects here is impractical, but the ones listed above typify the general field of NI research relating to this author's research. A key interest of recent NI research is in AI agents. By designing individual agents to act on their own goals, players can construct their own narrative based on what is happening around them.

Implementation

This thesis project implements Narrative Intelligence into a multiplayer, networked game. To handle large numbers of AI agents, the work is divided across several processors. In a multiple player situation, it is unlikely that all connected players will be using the full power of their processors, so this may be used for processing AI agents. By taking advantage of the unused resources of connected clients, larger numbers of AI agents can be simulated while retaining a low latency connection.

There are various possible ways to implement a distributed system, but the author chose a *task parallelism* design pattern due to its logical fit with the almost entirely independent work units of an AI agent system. For the workload distribution, the author chose the *Master/Worker* pattern due to its flexibility in changing networks and ease of implementation in an application that already heavily relies on message passing (interactive games). More details on the specifics of this algorithm follows after a brief note about network topology [5].

Game developers often need to worry about issues such as player cheating. Due to the flexibility included in this project's implementation, the developer has total control over which connected machines may execute potentially sensitive NI code. Thus, for small, friendly games, all connected machines may be allowed to execute the NI code, while large competitive games may have only dedicated NI-processing clients effect that computation. These two networking options are represented in the Figure 1 and Figure 2. Figure 1 illustrates an unsecured network where connected clients both execute NI code and handle player input / output. This will be suitable for most game purposes, though there is the chance that the player could attempt to detect what NI code is being executed, giving them a peek into the inner workings of the game. Skilled hackers could even change the NI code to benefit themselves. As an example, take

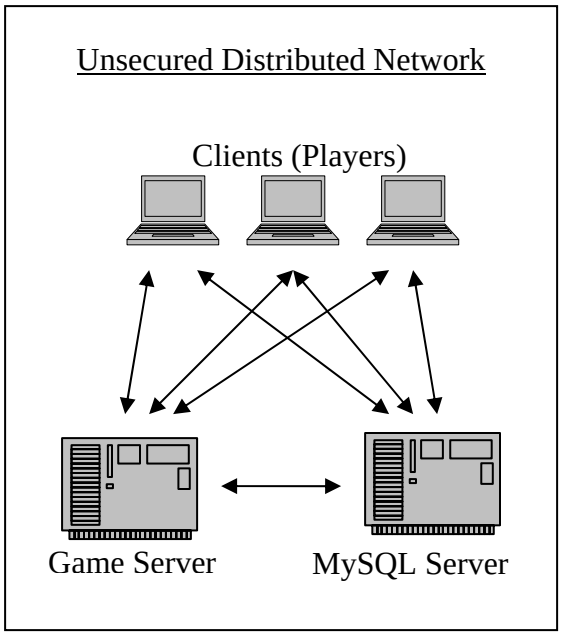


Figure 1: Unsecured Distributed Network

a massively multiplayer role playing game where a NPC (non-player character, here an AI agent) is usually programmed to make a fair trade with players. A skilled hacker could intercept the NI code, if it happens to be executing on his or her client, and change the code to make a trade that benefits the hacker. In the case where the in-game economy is exchangeable with real-world money (a growing trend in this genre), a successful hacking attempt translates to a fiscal loss. When sensitive NI code needs to be processed, a network topology such as Figure 2 should be used. Figure 2 shows how clients have no access to either the NI code or the database. The NI processing machines could be placed behind a secure firewall to protect them from hacking attempts.

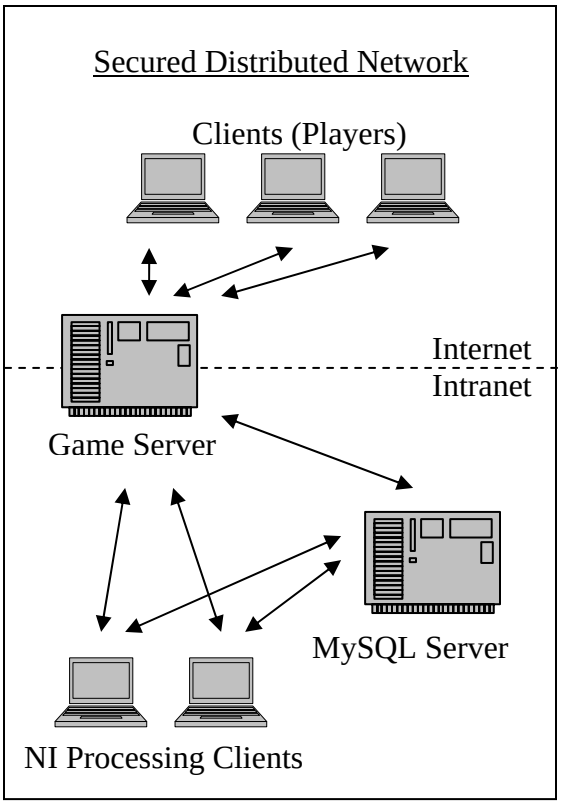


Figure 2: Secured Distributed Network

In the pursuit of creating NI, several code objects were created to help organize

AI agents. As visible in Figure 3, those objects are AIThink, AIThinkManager, AIThinkExec, and AIThinkScan.

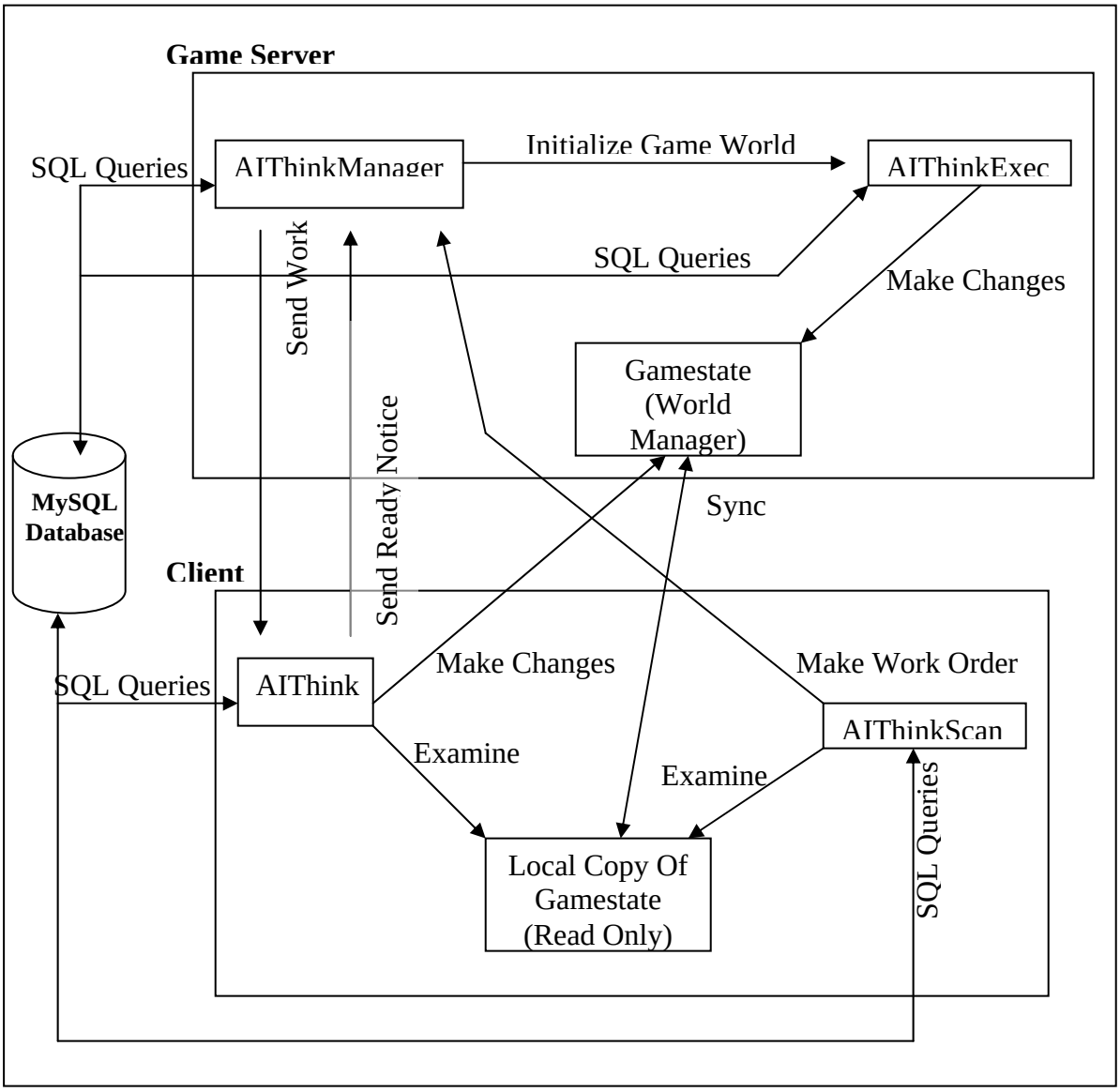


Figure 3: Program Map Diagram

The MySQL database stores all shared data so that all authorized clients may access it and make calculations based on the latest

data. MySQL's replication features may be helpful if the simulation is simulating great numbers of AI agents and a single database server becomes overloaded; but that is beyond the scope of this thesis. AIThinkManager (see appendix D for header files or appendix F for the source code) is located on the game server and handles the distribution of work. After a client connects, AIThinkManager can launch AIThink, AIThinkScan, or nothing at all, on the client-- depending on how the administrators decide to implement the network topology. AIThinkManager keeps track of all launched AIThink's and the AIThinkScan. AIThink (see appendix B for header files or appendix C for source code) handles the actual computation of an AI agent. When an AIThink is idle and there is work available, AIThinkManager sends work to the AIThink. AIThinkScan (see appendix F for header files or appendix G for source code) determines what work needs to be done, and informs the AIThinkManager to send out work. To set up the initial game state, AIThinkManager calls upon AIThinkExec to load all AI agents from the MySQL database and make physical representations of them in the game world.

The MySQL database shares the latest information relevant to AI agent processing. Table 1 describes its format.

Table Name	Field Name	Type	Description
agents	AgentID	int	Unique database ID of agent.
	agentName	tinytext	Human-readable name of agent
	queueTreeRootID	int	Current queuetree's root ID
	agentInGameServerID	int	Agent's physical representation in game state ID
	transform	text	Agent's physical location in game state
	currentlyWorkingOnStatusID	int	ID of which status we are presently trying to satisfy
	currentQueueStepID	int	Which step we are currently executing
	currentQueueStepTimesOutWhen	double	When the step we are currently executing is due to time out
knowledgekernel	kernelID	int	Unique database ID of knowledge kernel
	agentID	int	Which agent thinks this
	stepID	int	What step are we thinking this about
	difficultyRating	double	How difficult this agent thinks this step is
queuenode	nodeID	int	Unique ID of queue node
	parentnodeID	int	Parent of this node
	stepID	int	Step ID this queue node corresponds to
	lastCompleted	double	The last time this agent completed this node
status	statusID	int	Unique ID of status
	info	text	Human readable text explaining this status
	badLevel	double	Numeric of when this level is considered "bad," or when badStuffStepID is executed when this drops below
	badStuffStepID	int	Step ID to execute when status drops below badLevel
	badFrequency	double	After dropping below badLevel, how often to re-execute badStuffStepID
	okLevel	double	Above this level, agents will not try to improve this status
statuschange	changeID	int	Unique ID of change
	statID	int	Which status to change
	howMuch	double	How much to change status
	type	int	0 = Addition 1 = Subtraction
	stepID	int	Execute this status change after successful completion of this

			step ID
statusmap	mapID	int	Unique statusmap ID
	agentID	int	Agent this status is for
	statusID	int	Status this statusmap represents
	value	double	Numeric value of status
step	stepID	int	Unique Step identifier
	info	text	Human readable text, informing reader what this step is
	baseDifficultyRating	int	Approximately how hard this step will be. This will be fine-tuned later as AI agents form their own opinions about step difficulty
	requireAll	int(1)	0 = Only require one leaf node to successfully complete to consider this step successful 1 = Require all leaf nodes to succeed to consider this step successful
	parentStepID	int	ID of parent node
	action	text	Script-engine text of action to execute for this step. If none, set to "0". Example to send an AI agent to a waypoint names house1: "aipath.makepath(<AGENT>.nearestwp(), house1.getID(),<AGENT>);"
	completeCheck	text	Script-engine text to determine whether step was successful. If none, set to "1". Example to see if AI agent is near the waypoint house1: "<AGENT>.nearestwp() == house1.getid()"
	failureTimeout	double	Length until the AI agent should give up on this step
	completeTimeout	double	Length after successful completion to remain successful. If -1, then always remain successful after a success.

Table 1: Database Format

As Table 1 shows, the database is made up of several key tables. The *status* table defines various needs that AI agents will need to fulfill. The *statusmap* table links these needs to AI agents. a*gents* defines each agent in the simulation. s*tep* defines steps that allow AI agents to fulfill their needs.

statuschange allows each step to cause multiple status effects, which AI agents will try to take advantage of by attempting to execute those steps that fulfill their status/needs. *queuenode* helps AI agents map out their progress across several steps' execution. *knowledgekernel* allows AI agents to form unique opinions about step difficulty.

The actual algorithm used by AIThink and AIThinkScan is summarized in Table 2.

AIThinkScan	1. Every set interval, all attached statuses for each AI agent will be evaluated. 2. Choose the status closest to its BadLevel, ignoring statuses above okLevel. 3. If the chosen status is not the same as the active status (or there is no active status), make a work order for this agent. If the current step for this status is overdue, make a work order for this agent.
AIThink	4. The workorder is executed by an AIThink. The status that is closest to BadLevel (ignoring any statuses above okLevel) will be set as the active problem. If this is already the active problem and the current step is not overdue, cease execution and mark self as ready for a new work order. 5. If the active problem is changing, then throw out the old Queue Tree and make a new one. a. Search the steplist for StatusChanges that will increase our status level. b. Add all these steps to the Queue tree. c. Expand the tree to include all substeps and actions. d. Mark completed steps (steps which we already meet the CompleteCheck) e. Count up the difficulty ratings of each step (do not include completed steps), and add them upward to the root of the tree to indicate which step paths will be the hardest f. Choose the step leaf with the lowest difficulty rating g. Discard all other step leaves 6. Run through the tree and evaluate CompleteChecks. Mark Complete steps and unmark steps that are no longer complete. 7. If an action is not already being performed, and the current action has not timed out, then perform a new action. One step at a time, starting with the furthest step from the leaf step, perform actions to achieve each CompleteCheck (ignoring any that are already complete). a. If several actions or steps are required for a step, perform the first step or action first, then the second, and so on. i. If the step fails, increase the difficulty rating by three. ii. If the step succeeds, mark step as complete. Decrease the difficulty rating by one. Execute any statuschanges for this step.

	b. If one action or step is required for a and multiple actions or steps are provided, perform the action or step with the lowest difficulty rating. i. If that step fails, increase its difficulty by three. ii. If that step succeeds, decrease difficulty rating by one. Mark that step and this step as complete. Execute statuschanges for both. 8. The tree should not empty or enter a state where it cannot do any tasks. If no tasks can be found, delete the tree and start fresh on the next interval. Eventually the status should fulfill and the AI Agent will learn the best methods of fulfilling it.

Table 2: Algorithm Summary

This algorithm gives simple AI agent behavior that is easily parallelized by executing multiple AIThink's concurrently. The thesis implementation also features a GUI interface to facilitate launching AIThink and AIThinkScan on connected clients, as well as other tools to control NI processing. Figure 4 is a screenshot of the GUI. For precise details on the inner workings of AIThink, AIThinkManager, and AIThinkScan, read their source code and header files in appendices B-G.

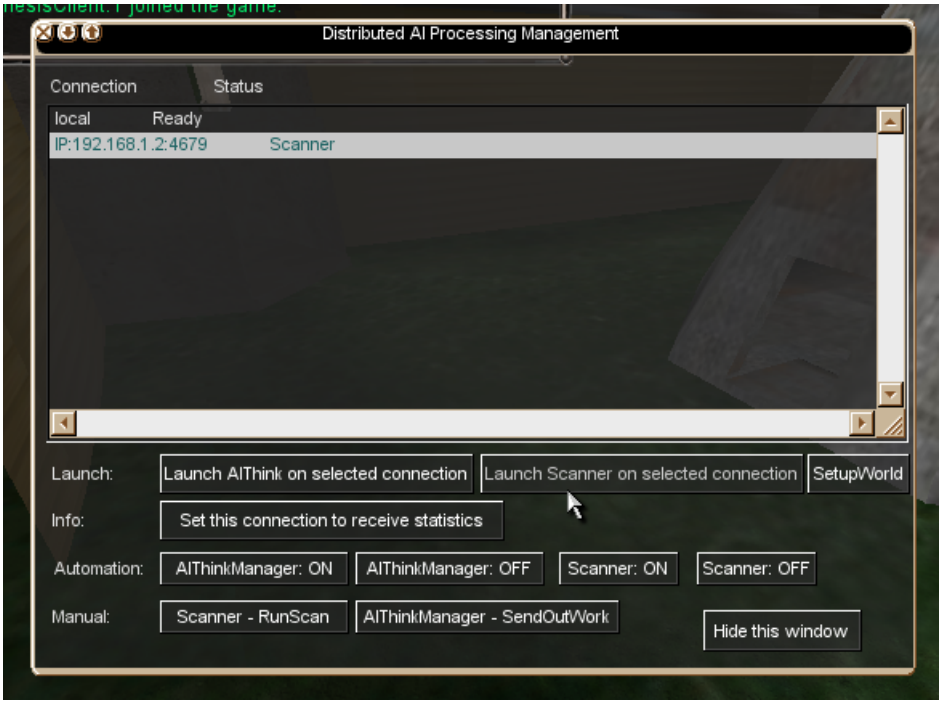


Figure 4: GUI Screenshot

Alpha Testing

To test the effectiveness of the example project's implementation of the algorithm, a simple test was devised. By measuring the client/server latency (in milliseconds) of a client connected to the game server, we can see roughly how smooth the player's experience is. While this is not a perfect measurement, it does provide a statistical approximation of the quality of a player's connection to the game server (disregarding factors such as dropped packets). Several test cases were

devised and lettered A-H. The first test case, A, simulates a single-machine scenario (such as a single-player game) in which only one machine is available for processing. The Game Server, AIThink, AIThinkScan, and client all run on one machine within one process. In test case B, two machines are available. All functions continue to run on one machine, with the exception of the client which runs on the other machine. This division of labor should free up more resources for the client to process input for a somewhat more responsive interface. Far better yet is to have three machines available, such as in test case C, where AIThink and AIThinkScan run on one machine, the Game Server on another, and the Client on the third. Since one machine can now focus on Game Server activities such as managing client interactions and game state with a client machine that can focus on managing user interaction while a separate machine handles intensive AIThink and AIThinkScan functions, an even more responsive environment should be presented to the user. In D-H, AIThinkScan and AIThink are separated into two different machines. In each letter an

additional AIThink machine is added, starting with one at D and ending with five at H.

For each test, the maximum latency was measured once AIThinkManager and AIThinkScan were activated. This was measured until all AI agents had traversed the physical world representation, or until ten minutes had passed, whichever occurred first. The latency measurements were taken by polling a function that got the current latency to the server via a script. For all tests, the MySQL server resided on a separate hyperthreaded 3ghz machine, while the other machines used had consistent specifications between tests running at a speed of (single-core) 3ghz. For pictures from this test, see appendix A.

	A	B	C	D (1 NI)	E (2 NI)	F (3 NI)	G (4 NI)	H (5 NI)
1 agent	55	52	52	57	56	47	53	52
50 agents	322	170	61	54	64	70	65	59
100 agents	1024	2667	72	67	73	80	71	81
500 agents	1023	22587	1488	2433	2855	6028	8407	7158
(All measurements are in units of milliseconds)								

Table 3: Peak latency for client/server connection

While acceptable levels of latency vary between player to player and game to game, latency below 100ms is generally considered excellent. Latency above 1000ms (or one server update per second) is often considered unplayable as it leads to a game that

plays like a slideshow. From the first dataset, one agent, peak latency remains good in the range of 47-55ms. There is no appreciable change between the various test cases, illustrating that with only one agent no one part of the system is stressed.

When 50 agents are introduced into the system, the importance of distributed processing becomes clear. In test case A, with only one machine, latency rises above optimal levels. Even with a server that handles the full load, the client still experiences above optimal levels of latency in test case B, likely due to heavy load on the server. By distributing the server's work into two separate machines in test case C, client latency returns to excellent levels, and continues to stay low in tests D-H. Even with 100 agents, the client can still retain excellent latency when parallel processing techniques are used. When 500 agents are simulated, latency becomes a serious issue due to a single Game Server attempting to synchronize 500 complex moving objects with all connected machines. Additional attempts at increasing agent parallel processing in this situation actually seem to hurt the client's latency as the Game Server

spends more time synchronizing the game world across all connected machines.

Finally, it should be noted that the latency for test case A in situations of 100 and 500 agents may be skewed. It seems that when the Game Server is essentially connected to itself (as it is in test case A), latencies above 1,024ms are not reported (for a single user scenario, a network protocol is probably not used at all). Even if the "network" connection to itself has a latency of 1,024ms, response to user commands may be much, much higher (as it seemed to be during testing).

Beta Testing

While alpha testing showed that the distributed algorithms could produce a high-latency-free gameplay experience, without testing with people it is uncertain whether narrative intelligence can create an immersive world for players. Beta testing was conducted to determine whether the implemented algorithms could power such a world.

For this test, an expanded example was created. In this example, agents had two needs: hunger and fun. The agents would seek food if their hunger was below a threshold of fifty

hunger points (on a scale of 0-100). Below twenty hunger points, the agent would receive damage due to entering a state of starvation. If the agent did not increase its hunger points above twenty in this state, they would die. Agents would also seek fun if they were not actively seeking food. When an agent was spawned, it would randomly have hunger points between zero and one hundred. Fun points would be randomly generated between zero and one hundred as well. All agents spawned in the middle of the virtual world so that they could quickly access either food or fun. For fun, agents could visit a castle and party (as expressed through a waving animation). For food, agents would first try the nearby McThesis restraunt. If players did not serve them food, they would eventually leave and try finding food at the fishing pond. All of these behavioral rules were expressed in the SQL database using the same tables and fields as previously mentioned in Table 1. Finally, if an agent had more than fifty hunger points and more than a thousand fun points, a new agent would be created while the previous agent's points were reset. See appendix A for pictures from this test.

Though it could be construed as un-scientific, the author invited a group of his friends to try the project-- to see if they could construct a narrative out of their experiences. These friends were experienced gamers and would be able to have a much more insightful view into this project due to their knowledge of other games than a random population sample who might have trouble distinguishing this project from other games. The group of seven players met for an impromptu test and played the beta example game for an hour. In addition to the AI Agents that the players could interact with, the players also enjoyed driving vehicles, finding health and ammo packs, and changing their size with help from the mission editor. Three factions of players emerged: those that tried to help the agents, those who actively sought their demise, and those who ignored the agents. Initially, these three factions were balanced, because players would try all three tactics. For most of the game, the faction sizes remained about the same, but with different players at different times. During parts of the game, the faction seeking the agents' demise grew dramatically, as the players formed a firing squad to eliminate new agents. At other

times, the pro-agent and anti-agent factions strongly disagreed with each other, even yelling at each other during the game and killing each other. One player even impersonated an agent by setting his name to that of an agent, and pretended to be an agent by performing agent tasks. The players really became immersed in the game and seemed to enjoy it. At the end of an hour of testing, each player was asked to write a short story about their gameplay experience. The players were encouraged to be creative. Their answers are shown in Table 4 below.

As Crusnik 02 walked around the new world, confused and scared, he realized he had a crossbow. He took the crossbow in his hand and fondled it like his little kitty he had at home. But then! he saw many other appear! He was scared! Worried!! He hid in a sniper outpost and began to shoot them!!! BLOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO OOOOOOOOOOOOOD WAS SPILLED!! But then Crusnik realized that killing other scared people with a crossbow caused the world to crash. After Crusnik died, he reincarnated himself as LoveMachine. As LoveMachine he experienced being really really small and really really big. It was delicious. Yes it was....mmmmmmmmhmmmmm. Pong. Pong. Pong. After a bit, these crazy people came out of this pink little hole and wanted food. Well LoveMachine said screw them! and killed them. We chased a few who weren't willing to give their blood to Lucifer in the virginal sacrifices. We pushed them into the water and drowned them. We purified their bodies in the waters of Lake Minatonka, like Prince when he played basketball with Charlie Murphy. All of a sudden, Rick James came out the sky and started singing Cold-blooded. But little did he know that that was a sin against the holy Lucifer and his blood had to be spilled as well. After that, the phallus of our Lord Beziel showed the passage-way to the pink hole that we all came from. God Bless
I arrived in Thesis Town on the 2nd of April. There were already a number of other company workers present, each of whom had been previously issued armor and a crossbow. I asked why we would need crossbows - I thought that our assignment was to shepherd the new clansmen while they developed the variety of skills necessary for life during the wartime. That's when I noticed the McThesis. "A McThesis? I thought that we were here to see the development of soldiers...bloodthirsty, battle-hardened soldiers!" My companions shrugged. "Well...there are fish over the pond. Maybe the McThesis is just for us." "Well, ok. But the crossbows? What are they for?" That's when I heard the sharp twang of a crossbow being fired. And again. And again. Everything went black. --- Sometime later, I opened my eyes. Still in Thesis Town. My head was a little groggy, but I wanted to get to work. "Have the Blanks arrived?" "Not yet, but Bill is out watching the skies. We figure the first drops should be happening shortly. Here, take your crossbow." The first Blanks were falling as I steppd outside the fort. We all watched in anticipation once he had hit the ground. Our assignments had let us know that the drop procedure sometimes disoriented the Blanks, but eventually their presence of mind would return and their instincts would kick in. We were not to interfere with their development, only make

sure that they grew stronger. This blank, however, had yet to move - what was taking him so long? Enough time had passed for his vision to return, surely he would let us know that he was hungry or would try out his legs, maybe? "This one's a dud," I heard. "Obiously defective." Five more twangs. The Blank was no more. "That's valuable company property!" I shouted. "It could have been, but it obviously wasn't. They'll be more." He was right - after a few more minutes we had a veritable tower of Blanks, none of whom were doing anything. I grew sick of the twangs. Days passed before we received our first functioning Blank, but he unfortunately starved to death while dancing next to our tent. Many of the Blanks seemed to have an affinity for dancing, and others almost instinctively moved towards the McThesis. Once inside, they stood mouths gaping, waiting. We lost a few before it became apparent that they were waiting for us. This was not an entirely acceptable situation to some members of our party. We were here to watch over developing soliders - not feed babies (albeit babies that could dance)! A callousness grew in some of my companions, none of these Blanks were acceptable. Therefore, they needed to be terminated. The twangs were incessant. I did what I could to feed the Blanks, since I did not consider it a duty below myself, but the bloodlust in my companions had grown too much. My poor Blanks would often try to hide in the pond, but it was all too futile. Not a Blank survived our time in Thesis Town. I lost touch with my fellow shepherds when I returned to the company. The war ended a few short weeks later and I tried to find them, but always seemed to run up against bureauacratic tape. Late one night, I gained access to Jones' file and started reading...TWANG.

I woke up to find myself a disembodied ghost, swarming over the deserted town of thesisville. There was a strange vehicle-thing on top of a building, which I tried to get to but never could. Seriously, I spent forever trying to get to that car, and it bugged the crap out of me! Anyway, I corporealized in mid-air and dropped down, to begin my existence as Urgo, an orc-thing with exploding crossbow bolts. Thus began my journey through thesisville, battling mindless agents, gaining medicine at McThesisworld, and trying to get to the top of the building for that vehicle. Then I actually managed to get my hands on a vehicle, and found out they were impossible to steer! So I went back to killing agents, occasionally dying and reincarnating as other orc-things, battling others with my exploding crossbow bolts of doom. Occasionally the gods of chaos and mean-spiritedness would shrink and grow us in size, much to the amusement of my fellow wierd orc-things. In the end it was a grand day of slaughter and death, till the world froze and shattered. The gods of chaos and mean-spiritedness had destroyed the world once and for all, and so now I leave to orc Valhalla, to enjoy an eternity of wine and beer with orc-ish valkeryies of questionable virtue.

Once, many eons ago beyond mortal reckoning, a demi-god was given, by his pantheon, the powers of creation. To test his creation, he went to a secluded land in the sky, surrounded by mountains. In this land, he placed many buidlings, a fun moutnain, and his favorite restaurant, McThesis. He then placed upon his land agents of his will, or "Players" as he called them. These players were given armaments, consisting of crossbows with exploding arrows and bags of infinite ramen. He would grant favor unto his players by changing their size when they asked. He then placed people into his worlds. They resembled his "players", but they lacked their armaments and ability to drive vehicles. They also lacked true free will, only fufilling their basic desires to eat and have fun. The demi-god then let the players do as they wished, whether it was feeding the people at McThesis, letting the people play, watching the people attempt to fish, driving around the vehicles, or just killing the people at random. The demi-god, whose name was Ro-b'e, smiled. For his creation was good. But then, one of his "players" killed another one, and Ro-b'e was displeased. He destroyed his world and recreated it, making the players take on new identities. And thus, the cycle began again.

The land of McThesis was a calm land...until the day, the ascnedents came. The dark portal opened and the minions of the evil lord entered the peaceful world, at exactly 2:46pm. In the begining, the minions formed many factions, but the dark lord refused to allow them to sedate their bloodlust on one another. And so, after many teams attempted to overtake the land of McThesis, the minions finally coalleced into one, cohesive team...save for incompetence, but well..he's incompetent. The dark lord was pleased with the results, but was curious as to the absence of of any other lifeforms. The minions of the dark lord were getting restless when the mindless lifeforms began to spawn. The agents of McThesis began to appear in droves, but the minions of the dark lord were preparred. Many of the angents fell as they spawned, though some of the minions called for patience, that they might tortue the agents, by feeding them

poison, or allowing them the small hope of freedom. The Executor, leader of one of the minion tribes, began to round up the Agents of McThesis to the play area, and then fired into the center of the mob, exploding all of the minions. The power of the Dark Lord rewarded his best minions, granting them hug size, or the abilities to walk through solid matter. However, the Agents of McThesis fought back, causing the minions to fly into the sky, or fall through the ground. In the end, the agents of McThesis were able to force the Minions of the Dark Lord out of their world, and survived to live a happy life.
We looked around and we saw a new strange land. Suddenly, orcish creatures appeared ontop of each other. Some decided that they were hungry. Others thought they wanted to play. We really didn't know what they wanted, but they threatened us with death, so we fed them, then we killed them. The end.
we started by prooving to our selves that the consequences of killing eachother was the crash of the server. once we got done crashing the system, we started to feel, kill and ignore the AI constructs that were the purpose of the game. there are 10 shots that each character can carry, so refilling health and ammo were top priorities. there were several cars that were very dificult to drive but fun none the less. all in all this has the makings of a fun game that could be expounded uppon like WOW has.

Table 4: Players' Short Stories from Beta Test

The beta test showed (1) how significantly the players could immerse themselves in the game and (2) that with creativity players can construct an interpreted narrative based on their experiences. Additionally, a multiplayer test with numerous agents helped validate the technology: players perceived no lag since the workload was placed on two machines without players. A production-level game with a full team working on it could create a far larger game with much more detailed agents that have many more needs with many more steps to fulfill them. The players' responses indicate that even with a minimal example, creative players can construe a universe built of AI agents to form narrative intelligence.

Conclusions and Recommendations

Distributed computation holds great promise for interactive entertainment. Two primary points are illustrated through the alpha and beta tests. The first point is that the technology of simulating AI agents in multiplayer interactive entertainment is not only possible, but (through distributed processing) can actually lead to larger numbers of agents without significantly increasing latency. The second point is that this technology can be used to create narrative intelligence.

The first point is proven by the alpha testing, which indicates that the use of parallel processing can significantly decrease latency for connected clients while allowing for larger, more detailed virtual worlds. The alpha testing also indicates that at a certain point parallelizing agent code alone will not be sufficient, but new techniques may be required in other areas of the Game Server to manage great numbers of interacting entities. For small to medium numbers of AI agents, the developed technology should be sufficient.

The second point is proven by the beta testing, that narrative intelligence can form through the use of AI agents.

Even by creating minimally deep AI agents, game developers can develop an immersive world for players to exist and play in. This immersion is illustrated by the highly creative player responses and reactions during the beta testing.

Ideas for future expansion of the engine include (1) development of a user friendly method of editing steps and other database information instead of directly entering it into the MySQL database, as well as (2) improving handling of massive numbers of AI agents, perhaps using parallel methods.

Ultimately, narrative intelligence expressed through AI agents holds great promise for an immersive gaming experience.

Bibliography

[1] Mateas, M., and Sengers, P., 1999. Narrative Intelligence. AAAI Fall Symposium in Narrative Intelligence 1999, Cape Cod, MA, AAAI Press.

[2] Steiner, K. E. and Tomkins, J. 2004. Narrative event adaptation in virtual environments. In Proceedings of the 9th international Conference on intelligent User interface (Funchal, Madeira, Portugal, January 13 - 16, 2004). IUI '04. ACM Press, New York, NY, 46-53. http://doi.acm.org/10.1145/964442.964453

[3] Young, R. M. and Riedl, M. 2003. Towards an architecture for intelligent control of narrative in interactive virtual worlds. In Proceedings of the 8th international Conference on intelligent User interfaces (Miami, Florida, USA, January 12 - 15, 2003). IUI '03. ACM Press, New York, NY, 310-312. http://doi.acm.org/10.1145/604045.604108

[4] Brooks, K. M. 1996. Do story agents use rocking chairs? The theory and implementation of one model for computational narrative. In Proceedings of the Fourth ACM international Conference on Multimedia (Boston, Massachusetts, United

States, November 18 - 22, 1996). MULTIMEDIA '96. ACM Press, New York, NY, 317-328. http://doi.acm.org/10.1145/244130.244233

[5] Mattson, Timothy G., Sanders, Beverly A., and Massingill, B. 2004. Patterns for Parallel Programming. Boston: Addison-Wesley.

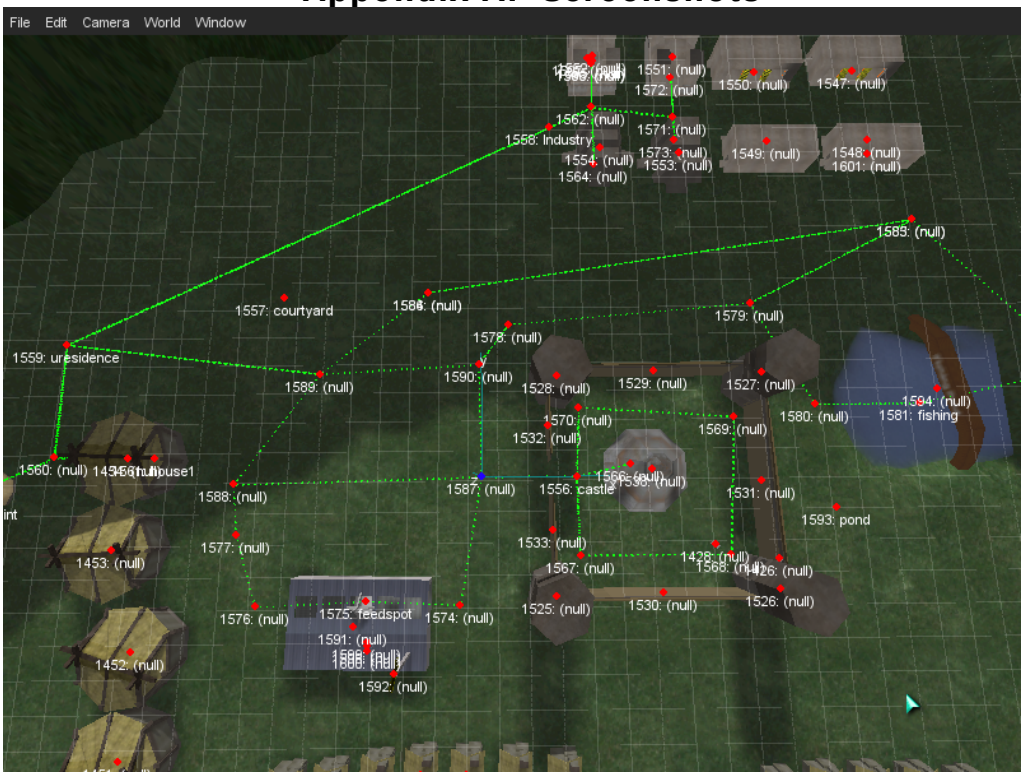
[6] Anstey, J. and Pape, D. 2002. Scripting the interactor: an approach to VR drama. In Proceedings of the 4th Conference on Creativity & Cognition (Loughborough, UK, October 13 - 16, 2002). C&C '02. ACM Press, New York, NY, 150-156. http://doi.acm.org/10.1145/581710.581733

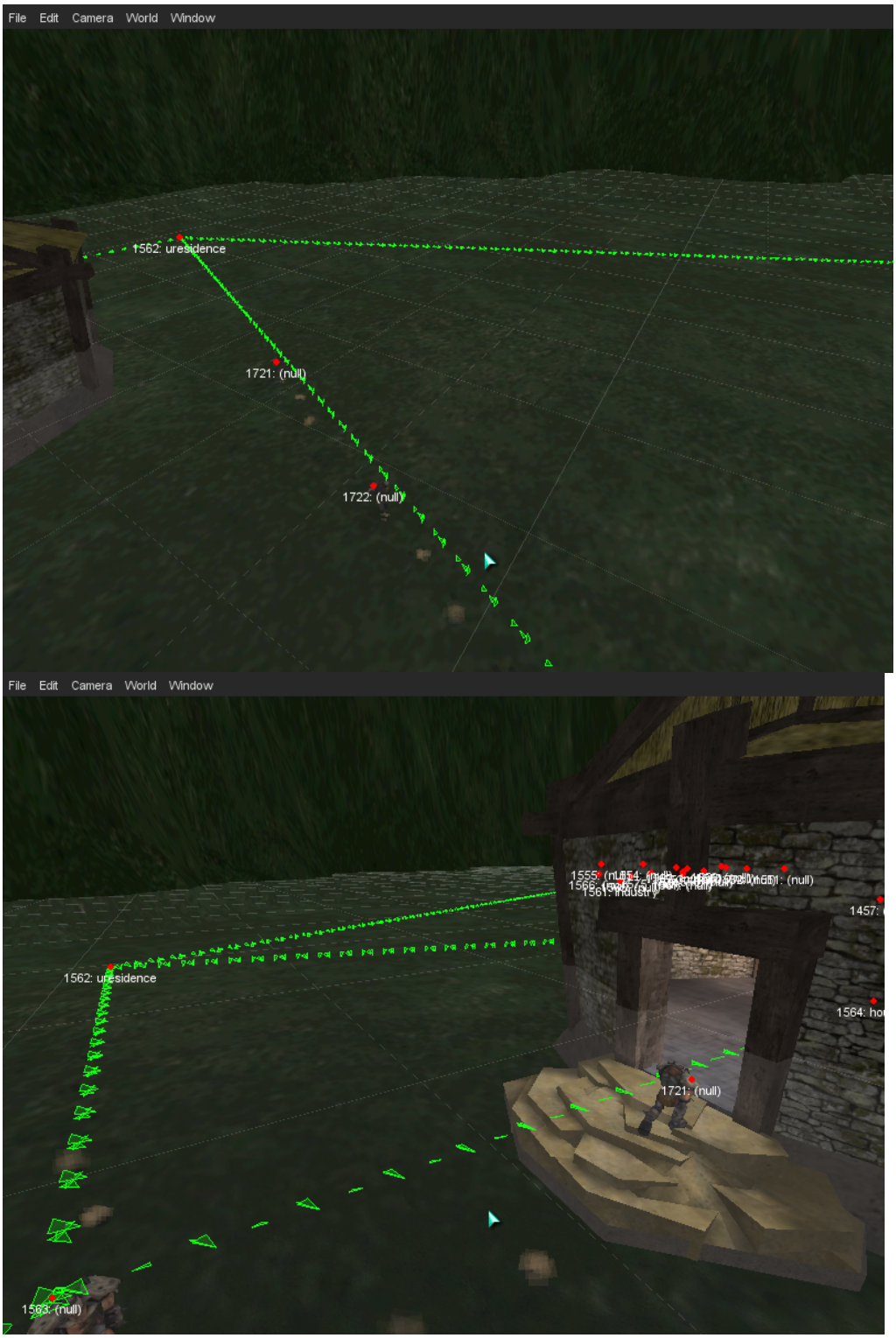
[7] Elliott, C., Brzezinski, J., Sheth, S., and Salvatoriello, R. 1998. Story-morphing in the affective reasoning paradigm: generating stories semi-automatically for use with “emotionally intelligent” multimedia agents. In Proceedings of the Second international Conference on Autonomous Agents (Minneapolis, Minnesota, United States, May 10 - 13, 1998). K. P. Sycara and M. Wooldridge, Eds. AGENTS '98. ACM Press, New York, NY, 181-188. http://doi.acm.org/10.1145/280765.280799

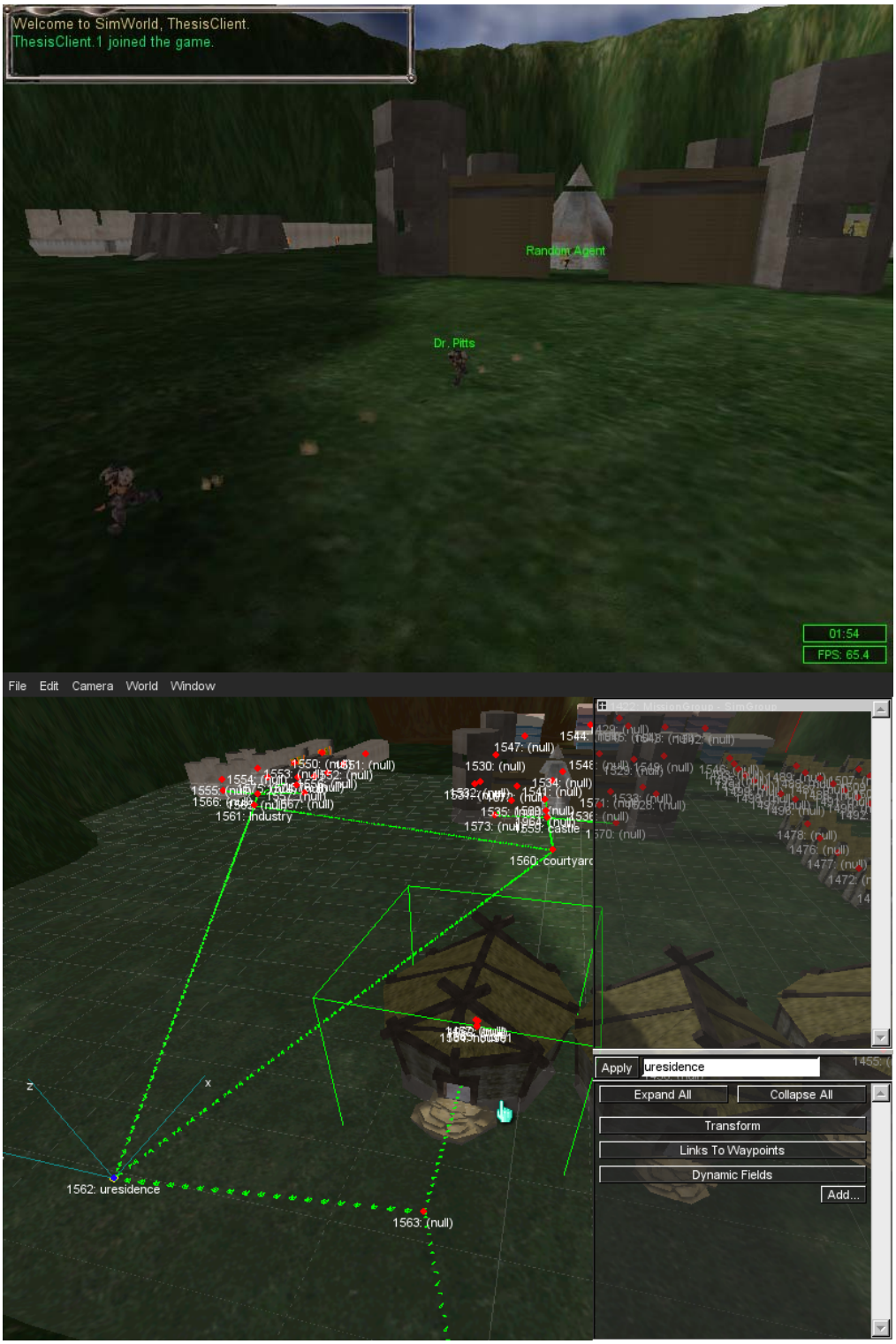
[8] Riedl, M. O. and Young, R. M. 2004. An Intent-Driven Planner for Multi-Agent Story Generation. In Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, 186-193. http://dx.doi.org/10.1109/AAMAS.2004.63

[9] Szilas, N. Interactive Drama on Computer: Beyond Linear Narrative. In Proc. AAAI Fall Symposium on Narrative Intelligence (North Falmouth MA, November 1999), AAAI Press, 150--156.

Appendix A: Screenshots




File Edit Camera World Window
1562: uresidence
1721: (null)
1722: (null)
File Edit Camera World Window
1562: uresidence
1721: (null)
1563: (null)


Welcome to SimWorld, ThesisClient.
ThesisClient.1 joined the game.
Random Agent
Dr. Pitts
01:54
FPS: 65.4
File Edit Camera World Window
1547: (null)
1530: (null)
1534: (null)
1561: industry
1560: courtyard
1562: uresidence
1563: (null)
Apply
uresidence
Expand All
Collapse All
Transform
Links To Waypoints
Dynamic Fields
Add...


File Edit Camera World Window
1457: (null)
1451: (null)
1568: newwaypoint
1563: (null)
1562: uresidence
1963: (null)
1564: house1
1962: (null)
x
y


Welcome to SimWorld, ThesisClient.
ThesisClient.1 joined the game.
Random Agent
Dr. Pitts
bob
01:15
FPS: 102.7
Welcome to SimWorld, ThesisClient.
ThesisClient.1 joined the game.
Distributed AI Processing Management
Connection
Status
local
Ready
IP:192.168.1.2:4679
Scanner
Launch:
Launch AIThink on selected connection
Launch Scanner on selected connection
SetupWorld
Info:
Set this connection to receive statistics
Automation:
AIThinkManager: ON
AIThinkManager: OFF
Scanner: ON
Scanner: OFF
Manual:
Scanner - RunScan
AIThinkManager - SendOutWork
Hide this window
01:06
FPS: 201.6


Welcome to RZ Thesis, ThesisUser.
BROWN
03:25
FPS: 59.8
Welcome to RZ Thesis, ThesisUser.
Fishing Pond
03:47
FPS: 150.4


Welcome to RZ Thesis, ThesisUser.
McThesis
VILLIAMS
DAVIS
01:37
FPS: 7.1

Appendix B: AIThink.h source code

```
//-----------------------------------------------------------------------------
// Robby Zinchak Thesis
// Copyright (C) Robby Zinchak
//-----------------------------------------------------------------------------

#ifndef _AITHINK_H_
#define _AITHINK_H_

#ifndef _SIMBASE_H_
#include "console/simBase.h"
#endif
#ifndef _TVECTOR_H_
#include "core/tVector.h"
#endif
#ifndef _mysql_H
#include "game/mysql/fimysql.h"
#endif

struct LocalStepNode
{
	int sqlID;
	bool expanded;
	Vector <LocalStepNode*> branches;

	// Extra info for decision:
	bool isDone;
	int diffRating;
	bool requireAllLeaf;
	bool isSmallestPath;
};

/// This class manages AIPlayer (AI Agents)
class AIThink : public SimObject
{
   typedef SimObject Parent;
   MySQL dba;
   char * querytext;
   void NodeExpander(LocalStepNode *node);
   void NodeDeleter(LocalStepNode *node);
   void NodeDisplay(LocalStepNode *node, int spaces);
   int NodeToSQL(LocalStepNode *node, int parentSQLid);

   void NodeExpanderQueue(LocalStepNode *node, int globalID, int ingameID);
   int NodeChooseEasiest(LocalStepNode *node);
   void NodeDisplayDifficulty(LocalStepNode *node, int spaces);
   int NodeFigureOutNextStep(LocalStepNode *node);
   void AgentSTRSearch(char *text, int agentID);
public:

   AIThink();
   ~AIThink();
   void Think (S32 globalID, S32 ingameID);

   DECLARE_CONOBJECT(AIThink);
};

#endif
```

Appendix C: AIThink.cc source code

```
//-----------------------------------------------------------------------------
// Robby Zinchak Thesis
// Copyright (C) Robby Zinchak
//-----------------------------------------------------------------------------

#include "thesis/AIThink.h"
#include "console/consoleTypes.h"
#include "core/resManager.h"
#include "core/fileStream.h"

IMPLEMENT_CONOBJECT(AIThink);
//AIThinkManager gAIThinkManager;

//-----------------------------------------------------------------------------

AIThink::AIThink()
{
	dba.setupDefaultThesisParams();
	querytext = new char [3000];
	Con::printf("AIThink - Init complete");
}

//-----------------------------------------------------------------------------
AIThink::~AIThink()
{
	dba.Close();
	delete [] querytext; querytext = NULL;
	Con::printf("AIThink - Destruct complete");
}

//-----------------------------------------------------------------------------


void AIThink::Think(S32 globalID, S32 ingameID)
{
	//return;

	Con::printf("======= THINK: ===== Doing AgentID %d in think", globalID);

	// First:  Is that status the same status as the one in
Agents.CurrentlyWorkingOnStatusID?
	// If so, update, otherwise makenew

	bool makenew = false;
	S32 result;
	int statusIDtofix = -1;

	int agentid = globalID;
	/*char * querytext = */ //new char [300]; omgwtfbbq?
	dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT status.statusID as
'should', agents.currentlyWorkingOnStatusID as 'is' FROM statusmap, status, agents
where statusmap.statusID = status.statusID and statusmap.agentid = %d and
statusmap.agentid = agents.agentID and (statusmap.value < status.okLevel or
status.okLevel = -1) order by (statusmap.value - status.badLevel) asc limit
1",agentid);
	Con::printf("%s",querytext);
	dba.Query(querytext);
	result = dba.StoreResult();
	//delete querytext;
	for (int i= 0; i< dba.NumRows(result); i++)
```

```
	{
		dba.FetchRow(result);
		int shouldn = atoi(dba.GetRowCell(result,"should"));
		int isn = atoi(dba.GetRowCell(result,"is"));
		Con::printf("Should/is: %d vs %d", shouldn, isn);
		if (shouldn != isn)
		{
			//This needs processing
			//Con::printf("Not equal");
			makenew=true;
		}
		statusIDtofix = shouldn;

	}
	dba.FreeResult(result);

	// ================= MAKE NEW TREE ====================

	if (makenew)
	{
		Con::printf("  *Making new tree on %d to improve %d", globalID,
statusIDtofix);
		Vector <int> leafIDs;

		leafIDs.clear();

		// First, get rid of the old tree
		// --> Since this is technically optional anyway, we'll ignore it
for now.

		// Second, figure out which steps can solve this status ID
		// (look in statuschange table for statuschanges that will fix
this, restricted to those that act on the
		// current statID we're trying to work on.  Merge that with steps.)

		//querytext = new char [300];
		dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT a.stepID as
zestep FROM statuschange a, step b  where statID = %d and a.stepID = b.stepID
order by a.stepID DESC",statusIDtofix);

		// SELECT * FROM statuschange a, step b  where statID = 1 and
a.stepID = b.stepID
		// will give us several leafs to which we shall trace back parents
for each until we reach roots.
		// From there, add that root node to our list of root nodes (thus
deleting any duplicates)
		// Then, recursively copy in our nodeids in our active domain to
the queuenode table

		int leafID;

		dba.Query(querytext);
		result = dba.StoreResult();
		//delete querytext;

		for (int i= 0; i< dba.NumRows(result); i++)
		{
			dba.FetchRow(result);
			leafID = atoi(dba.GetRowCell(result,"zestep"));
			Con::printf("   LEAFID: %d",leafID);
			if (leafIDs.empty() || leafIDs.last() != leafID)
		 leafIDs.push_back(leafID);
		}
```

```
                //Con::printf("   before free");
                dba.FreeResult(result);
                //Con::printf("   after free");

                LocalStepNode * root = new LocalStepNode;
                root->sqlID = -1;

                while (!leafIDs.empty())
                {
                        leafID = leafIDs.last();
                        leafIDs.pop_back();
                        Con::printf("   Backtracing %d", leafID);
                        bool parentfound = false;
                        while (!parentfound)
                        {
                                dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT
parentStepID FROM step where stepID = %d",leafID);
                                dba.Query(querytext);
                                result = dba.StoreResult();
                                dba.FetchRow(result);
                                int parent =
atoi(dba.GetRowCell(result,"parentStepID"));
                                if (parent == 0)
                                        parentfound = true;
                                else
                                        leafID = parent;
                                dba.FreeResult(result);
                        }

                        LocalStepNode * newStep = new LocalStepNode;
                        newStep->sqlID = leafID;
                        newStep->expanded = false;
                        root->branches.push_back(newStep);

                        Con::printf("   Backtraced to %d", leafID);
                }

                //NodeDisplay(root,0);
                Con::printf("   Done with all backtracks, expanding internal
tree");

                // EXPAND INTERNAL TREE
                NodeExpander(root);
                NodeDisplay(root,0);
                int queuenodeid = NodeToSQL(root,-1);
                NodeDeleter(root);

                // Mark agent as having the new tree
                dSprintf(querytext,2900/*sizeof(querytext)*/,"UPDATE agents SET
queueTreeRootID=%d, currentQueueStepID=-1, currentlyWorkingOnStatusID=%d where
agentID = %d",queuenodeid, statusIDtofix, globalID);
                dba.Query(querytext);
                Con::printf(querytext);


        }

        // ================== UPDATE TREE ====================


        Con::printf("  Updating tree on %d", globalID);

        // Do we not have a step we are currently executing?
```

```
        // If not, check status of current working on step.  Is it overdue?

        dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT a.currentQueueStepID,
a.currentQueueStepTimesOutWhen, a.queueTreeRootID, a.agentInGameServerID FROM
agents a where agentID = %d",globalID);
        dba.Query(querytext);
        Con::printf("First update query: %s",querytext);
        result = dba.StoreResult();
        dba.FetchRow(result);
        int currentQueueStepID =
atoi(dba.GetRowCell(result,"currentQueueStepID"));
        int queueTreeRootID = atoi(dba.GetRowCell(result,"queueTreeRootID"));
        int ingameserverID = atoi(dba.GetRowCell(result,"agentInGameServerID"));
        double currentQueueStepTimesOutWhen =
atof(dba.GetRowCell(result,"currentQueueStepTimesOutWhen"));
        dba.FreeResult(result);

        bool choosenewstep = false;
        // FOUR CASES =====
        // Step is currently executing & is within time -> Do nothing
        if (currentQueueStepID != -1 && ((U32) currentQueueStepTimesOutWhen) >
Platform::getVirtualMilliseconds())
        {
                Con::printf("  Step execution is within time");

                // Check to see if it is done:
                dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT
c.completeCheck, b.nodeID, c.stepID FROM agents a, queuenode b, step c where
a.currentQueueStepID = b.nodeID and b.stepID = c.stepID and a.agentID =
%d",globalID);
                dba.Query(querytext);
                Con::printf(querytext);
                result = dba.StoreResult();
                dba.FetchRow(result);
                bool done = false;
                int nodeID = atoi(dba.GetRowCell(result,"nodeID"));
                int stepID = atoi(dba.GetRowCell(result,"stepID"));
                //if (dba.GetRowCell(result,"completeCheck")) // May or may not
have completecheck
                //if (dba.NumRows() > 0) // May or may not have anything...
                // This should not cause a problem!! <_<
                if (true)
                {
                        dSprintf(querytext,2900/*sizeof(querytext)*/,"%s",
dba.GetRowCell(result,"completeCheck"));
                        //AgentSTRSearch(querytext,ingameID);
                        AgentSTRSearch(querytext,agentid);
                        //if (atoi(Con::evaluatef("echo (\"%s\");",querytext )) ==
1)
                        Con::printf("Thinker: %s", querytext);
                        if (atoi(Con::evaluatef("if (%s) { return 1; }",querytext
)) == 1) // SUCCESS!
                                done = true;
                }
                //if (atoi(Con::evaluate(querytext)) == 1)
                if (done) // WE FINISHED A STEP!!!!!!!!!!!!!!!!!
                {
                        Con::printf("FYI: DONE!, Making easier / Done  We (%d)
finished (%d)", globalID, stepID);
                        //Make simpler:
                        dSprintf(querytext,2900/*sizeof(querytext)*/,
```

```
                                "UPDATE knowledgekernel set
difficultyRating=((select difficultyRating) - 3) where agentID = %d and stepID =
%d",
                                globalID,stepID);
                        dba.Query(querytext);
                        Con::printf(querytext);
                        //Mark completed (timestamp)
                        dSprintf(querytext,2900/*sizeof(querytext)*/,
                                "UPDATE queuenode set lastCompleted=%d where
nodeID=%d",
                                Platform::getVirtualMilliseconds(),nodeID);
                        dba.Query(querytext);
                        Con::printf(querytext);

                        //Execute the effects of completion (if any) !

                        dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT * FROM
statuschange a, step b where a.stepID = b.stepID and b.stepID = %d",stepID);
                        //Con::printf("%s",querytext);
                        dba.Query(querytext);
                        result = dba.StoreResult();
                        //delete querytext;
                        for (int i= 0; i< dba.NumRows(result); i++)
                        {
                                dba.FetchRow(result);
                                int statID = atoi(dba.GetRowCell(result,"statID"));
                                int howmuch =
atoi(dba.GetRowCell(result,"howMuch"));
                                int type = atoi(dba.GetRowCell(result,"type"));
                                char typechar = ' ';
                                if (type == 0)
                                        typechar = '+';
                                else if (type == 1)
                                        typechar = '-';
                                dSprintf(querytext,2900/*sizeof(querytext)*/,
                                        "update statusmap set value=((select value)
%c %d) where agentID=%d and statusID=%d",
                                        typechar, howmuch, globalID, statID);
                                dba.Query(querytext);
                        }
                        dba.FreeResult(result);

                        choosenewstep = true;
                } else
                        Con::printf("FYI: Not done");
                dba.FreeResult(result);

        }

        // Step is currenlty executing & is out of time -> New step, Mark old
harder
        if (currentQueueStepID != -1 && ((U32) currentQueueStepTimesOutWhen) <
Platform::getVirtualMilliseconds())
        {
                Con::printf("  Step execution is out of time");
                //MAKE HARDER
                dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT
c.completeCheck, b.nodeID, c.stepID FROM agents a, queuenode b, step c where
a.currentQueueStepID = b.nodeID and b.stepID = c.stepID and a.agentID =
%d",globalID);
                dba.Query(querytext);
                Con::printf(querytext);
                result = dba.StoreResult();
```

```
                dba.FetchRow(result);
                int nodeID = atoi(dba.GetRowCell(result,"nodeID"));
                int stepID = atoi(dba.GetRowCell(result,"stepID"));
                dba.FreeResult(result);

                choosenewstep = true;
                dSprintf(querytext,2900/*sizeof(querytext)*/,
                                "UPDATE knowledgekernel set
difficultyRating=((select difficultyRating) + 1) where agentID = %d and stepID =
%d",
                                globalID,stepID);
                dba.Query(querytext);
        }
        // No current step -> New Step
        if (currentQueueStepID == -1)
        {
                Con::printf("  No step currently executing");
                choosenewstep = true;
        }
        // ===== END CASES

        // ================== NEW STEP GENERATION =====================
        if (choosenewstep)
        {
                // - determine easiest step path from here
                Con::printf("  Choosing new step...");
                LocalStepNode * root = new LocalStepNode;
                root->sqlID = queueTreeRootID;
                root->diffRating = 0;
                root->requireAllLeaf = false;
                root->isDone = true;
                root->expanded=false;
                NodeDisplayDifficulty(root,0);
                Con::printf("Now doing expanderqueue");
                NodeExpanderQueue(root, globalID,ingameID);
                NodeDisplayDifficulty(root,0);
                NodeChooseEasiest(root);
                Con::printf("After Choose Difficulty");
                NodeDisplayDifficulty(root,0);
                int steptodo = NodeFigureOutNextStep(root);
                Con::printf("We're going to do: %d", steptodo);
                if (steptodo == -1) // AIE, no possible way to fix, possible we
used up the queuetree
                {
                        dSprintf(querytext,2900/*sizeof(querytext)*/,"update agents
set currentQueueStepID=-1,queueTreeRootID=-1,currentlyWorkingOnStatusID=-1 where
agentID = %d", globalID);
                        dba.Query(querytext);
                        Con::printf("Potential Warning:  Ran out of things to do,
going to reset stats so as to build new tree");
                } else {
                        // - Mark the first required step in the path as the active
step
                        // - Execute that step / set time outs / etc
                        dSprintf(querytext,2900/*sizeof(querytext)*/,"select
failureTimeout,action from step,queuenode where queuenode.nodeID = %d and
step.stepID = queuenode.stepID", steptodo);
                        dba.Query(querytext);
                        result = dba.StoreResult();
                        dba.FetchRow(result);
                        U32 timeout = (U32)
atoi(dba.GetRowCell(result,"failureTimeout"));
                        U32 timenow = Platform::getVirtualMilliseconds();
```

```
                        timeout+=timenow;
                        // Do actual execution of step ( finally! >:D )

        dSprintf(querytext,2900/*sizeof(querytext)*/,"commandToServer('ExecCMD',
\"%s\");", dba.GetRowCell(result,"action"));
                        //AgentSTRSearch(querytext,ingameserverID);
                        AgentSTRSearch(querytext,agentid);
                        //Con::evaluatef("echo (\"%s\");", querytext );
                        //Con::evaluatef("%s", querytext );
                        Con::printf(querytext);
                        Con::evaluate(querytext);
                        dba.FreeResult(result);

                        dSprintf(querytext,2900/*sizeof(querytext)*/,"UPDATE agents
set currentQueueStepID=%d, currentQueueStepTimesOutWhen=%d where agentID=%d",
                                steptodo, timeout, globalID);
                        dba.Query(querytext);
                        Con::printf(querytext);
                        Con::printf("Done making new step");


                }
                NodeDeleter(root);
        }

        // ================== DONE =====================

        Con::evaluatef("CommandToServer('ProcAmReady');");
}

int AIThink::NodeFigureOutNextStep(LocalStepNode *node)
{
        // Go down the best path until we find a non-done step
        // It's possible we're totally done, in which case we should signal
        // with a -1 to build a new queuetree
        if (node->isDone != true)
                return node->sqlID;

        int steptodo = -1;
        if (node->requireAllLeaf)
        {
                for (int i=0; i<node->branches.size(); i++)
                {
                        int possible = NodeFigureOutNextStep(node->branches[i]);
                        if (possible != -1)
                                steptodo = possible;
                }
        } else {

                for (int i=0; i<node->branches.size(); i++)
                {
                        if (node->branches[i]->isSmallestPath)
                                steptodo = NodeFigureOutNextStep(node->branches[i]);

                }
        }
        return steptodo;
}

void AIThink::NodeDisplayDifficulty(LocalStepNode *node, int spaces)
{
        if (node->sqlID != -1)
        {
```

```
                char * dashes = new char [100] ;
                char dashchar = '-';
                dSprintf(dashes,99,"");
                for (int i=0; i<spaces; i++)
                        dSprintf(dashes,99,"%s%c",dashes,dashchar);

                Con::printf("%s%d (Diff: %d , Easiest: %d, Done: %d)",dashes,node-
>sqlID,node->diffRating,node->isSmallestPath,node->isDone);
                delete [] dashes; dashes = NULL;
        }

        for (int i=0; i<node->branches.size(); i++)
        {
                NodeDisplayDifficulty(node->branches[i], spaces+1);
        }

}


int AIThink::NodeChooseEasiest(LocalStepNode *node)
{
        // Calculate Leaf's difficulty
        int mydifficulty = 0;
        if (!node->isDone)
                mydifficulty += node->diffRating;
        if (node->requireAllLeaf)
        {
                for (int i=0; i<node->branches.size(); i++)
                {
                        mydifficulty += NodeChooseEasiest(node->branches[i]);
                }
        } else {
                int easiestbranchI = -1;
                int easiestbranchX = -1;
                for (int i=0; i<node->branches.size(); i++)
                {
                        int nodediff = NodeChooseEasiest(node->branches[i]);
                        if (easiestbranchI == -1 || nodediff < easiestbranchX)
                        {
                                easiestbranchI = i;
                                easiestbranchX = nodediff;
                        }
                }
                if (easiestbranchI != -1)
                {
                        mydifficulty += easiestbranchX;
                        node->branches[easiestbranchI]->isSmallestPath = true;
                }
        }
        node->diffRating = mydifficulty;
        return mydifficulty;

}

void AIThink::NodeExpanderQueue(LocalStepNode *node, int globalID, int ingameID)
{
        if (node->expanded == true)
                return; //Already done for us :D

// ***
        if (node->sqlID != -1) // Fill in leaf's details.  Likely no information
about root
        {
```

```
		dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT * FROM
queuenode,step,knowledgekernel where parentnodeID = %d and
queuenode.stepID=step.stepID and queuenode.stepID = knowledgekernel.stepID and
knowledgekernel.agentID = %d",node->sqlID,globalID);
		dba.Query(querytext);
		int result = dba.StoreResult();
		for (int i =0; i<dba.NumRows(result); i++)
		{
			dba.FetchRow(result);
			int child = atoi(dba.GetRowCell(result,"nodeID"));

			LocalStepNode * newStep = new LocalStepNode;
			newStep->sqlID = child;
			newStep->expanded = false;
			newStep->isDone = false;
			// Considered done if:
			long LastCompleted =
atol(dba.GetRowCell(result,"lastCompleted"));
			long Now = (long) Platform::getVirtualMilliseconds();
			long completeTimeout =
atol(dba.GetRowCell(result,"completeTimeout"));


			//While this won't be the most efficient, we cannot
			// determine done without also running effects of "done",
so nevermind.
			if ((LastCompleted != -1 && completeTimeout == -1) ||
				(LastCompleted + completeTimeout > Now))
			{
				newStep->isDone = true;
			} else {
				// As a last resort, run the completeCheck (if one
such exists)
				//if (dba.GetRowCell(result,"completeCheck"))
			/*	if (false)
				{
					dSprintf(querytext,querytext),"%s",
dba.GetRowCell(result,"completeCheck"));
					AgentSTRSearch(querytext,ingameID);
					if (atoi(Con::evaluatef("if (%s) { return 1;
}",querytext )) == 1)
						newStep->isDone = true;
					else
						newStep->isDone = false;
				}*/
			}
			newStep->diffRating =
atoi(dba.GetRowCell(result,"difficultyRating"));
			newStep->isSmallestPath = false;
			node->branches.push_back(newStep);
		}
		dba.FreeResult(result);
	}
//***

	for (int i=0; i<node->branches.size(); i++)
	{
		NodeExpanderQueue(node->branches[i], globalID, ingameID);
	}

	node->expanded = true;
}
```

```
void AIThink::NodeExpander(LocalStepNode *node)
{
	if (node->expanded == true)
		return; //Already done for us :D

// ***
	if (node->sqlID != -1)
	{
		dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT stepID FROM
step where parentStepID = %d",node->sqlID);
		dba.Query(querytext);
		int result = dba.StoreResult();
		for (int i =0; i<dba.NumRows(result); i++)
		{
			dba.FetchRow(result);
			int child = atoi(dba.GetRowCell(result,"stepID"));

			LocalStepNode * newStep = new LocalStepNode;
			newStep->sqlID = child;
			newStep->expanded = false;
			node->branches.push_back(newStep);
		}
		dba.FreeResult(result);
	}
//***

	for (int i=0; i<node->branches.size(); i++)
	{
		NodeExpander(node->branches[i]);
	}

	node->expanded = true;
}

void AIThink::NodeDeleter(LocalStepNode *node)
{
	for (int i=0; i<node->branches.size(); i++)
	{
		NodeDeleter(node->branches[i]);
	}

	delete node; node = NULL;
}

void AIThink::NodeDisplay(LocalStepNode *node, int spaces)
{
	if (node->sqlID != -1)
	{
		char * dashes = new char [100] ;
		char dashchar = '-';
		dSprintf(dashes,99,"");
		for (int i=0; i<spaces; i++)
			dSprintf(dashes,99,"%s%c",dashes,dashchar);

		Con::printf("%s%d",dashes,node->sqlID);
		delete [] dashes; dashes = NULL;
	}
	for (int i=0; i<node->branches.size(); i++)
	{
		NodeDisplay(node->branches[i], spaces+1);
	}

}
```

```
int AIThink::NodeToSQL(LocalStepNode *node, int parentSQLid)
{
	// Store this node, then any children
	dSprintf(querytext,2900/*sizeof(querytext)*/,"INSERT INTO queuenode
(parentnodeID, stepID) values (%d, %d)",parentSQLid,node->sqlID);
	dba.Query(querytext);
	/*int result = dba.StoreResult();
	for (int i =0; i<dba.NumRows(result); i++)
	{
		dba.FetchRow(result);
		int child = atoi(dba.GetRowCell(result,"stepID"));

		LocalStepNode * newStep = new LocalStepNode;
		newStep->sqlID = child;
		newStep->expanded = false;
		node->branches.push_back(newStep);
	}
	dba.FreeResult(result);*/


	// Get ID
	parentSQLid = dba.InsertID();//mysql_insert_id();

	for (int i=0; i<node->branches.size(); i++)
	{
		NodeToSQL(node->branches[i], parentSQLid);
	}

	return parentSQLid;

}

void AIThink::AgentSTRSearch(char *text, int agentID)
{
	//Con::printf("Size of text: %d", sizeof(text));
	char *temp;
	temp = new char [300];

	int i=0;
	int j=0;
	int appeartimes=0;
	while (i<290)
	{
		if (text[i] == '<' && text[i+1] == 'A' && text[i+2] == 'G' &&
			text[i+3] == 'E' && text[i+4] == 'N' && text[i+5] == 'T' &&
			text[i+6] == '>')
		{
			temp[j]='a';
			temp[j+1]='%';
			temp[j+2]='d';
			j+=3;
			i+=7;
			appeartimes++;
		} else {
			temp[j]=text[i];
			j++;
			i++;
		}
	}

	//for (int z=0; z<appeartimes; z++)
	//{
```

```
        //      if (z>0)
//                      dSprintf(temp,sizeof(temp),text);

        dSprintf(text,2900,temp,agentID,agentID,agentID,agentID,agentID,agentID,ag
entID,agentID,agentID,agentID,agentID,agentID,agentID,agentID);
//      }
        // Problems: Cannot write to self?  Spaces before/after number?


        delete [] temp; temp=NULL;
}


// ------------------------------------------------------------
// console methods
// ------------------------------------------------------------

ConsoleMethod( AIThink, Awaken, void, 2, 2, ""
                                        "Let server know we're ready\n\n")
{
        Con::evaluatef("CommandToServer('ProcAmReady');");
}

ConsoleMethod( AIThink, Think, void, 4, 4, ""
                                        "Think on an ID\n\n"
                                        "@param   globalID       Global (mysql) ID
of the ai agent that needs processing.\n"
                                        "@param   localID        Local (ingame) ID
of the ai agent that needs processing.\n")
{
        object->Think(dAtoi(argv[2]),dAtoi(argv[3]));
}

//--------------------------------------------------------------------------------
```

Appendix D: AIThinkManager.h Source Code

```
//-----------------------------------------------------------------------------
// Robby Zinchak Thesis
// Copyright (C) Robby Zinchak
//-----------------------------------------------------------------------------

#ifndef _AITHINKMANAGER_H_
#define _AITHINKMANAGER_H_

#ifndef _SIMBASE_H_
#include "console/simBase.h"
#endif
#ifndef _TVECTOR_H_
#include "core/tVector.h"
#endif
//#ifndef _mysql_H
//#include "game/mysql/fimysql.h"
//#endif
//#ifndef _PLATFORM_H_
//#include "platform/platform.h"
//#endif
#ifndef _AIEXEC_H_
#include "thesis/AIThinkExec.h"
#endif


struct aithinkinfo // simple data struct which we can look through for a ready
processor
{
        int connID;
        enum status { READY, BUSY, OFFLINE };
        status currentstatus;
};

struct workorder
{
        int aiAgentGlobalID;
};

/// This class spawns, tracks, and manages AIThinks
class AIThinkManager : public SimObject
{
   typedef SimObject Parent;
   Vector <aithinkinfo> knownaiprocessors;
   Vector <workorder> worktodo;
   int ScannerConnID;
   S32 PopAgent(); // Pop off the next agent to process; -1 if none
   bool autoupdates;
   //MySQL dba;
   //char * querytext;
   AIThinkExec *executor;
public:

   AIThinkManager();
   ~AIThinkManager();
   void SendOutWork();
   void MakeWorkOrder(F32 aiAgentGlobalID);
   void DisplayWorkToDo();
   void DisplayProcessors();
   void ProcStatusChange(F32 connID,aithinkinfo::status newstat);
   void AddAIProc(F32 connID);
```

```
   void SetScanID(F32 connID);
   S32 GetScanID();
   void processTick(const *Move);
   void SetAutoUpdates(bool doauto);
   void SetupWorld();
   void NewAgent();

   DECLARE_CONOBJECT(AIThinkManager);
};

extern AIThinkManager gAIThinkManager;

#endif
```

Appendix E: AIThinkManager.cc Source Code

```
//-----------------------------------------------------------------------------
// Robby Zinchak Thesis
// Copyright (C) Robby Zinchak
//-----------------------------------------------------------------------------

#include "thesis/AIThinkManager.h"
#include "console/consoleTypes.h"
#include "core/resManager.h"
#include "core/fileStream.h"

IMPLEMENT_CONOBJECT(AIThinkManager);
AIThinkManager gAIThinkManager;

//-----------------------------------------------------------------------------

S32 AIThinkManager::PopAgent()  // Pop off the next agent to process; -1 if none
{
	if (worktodo.size() < 1)
		return -1;

	S32 aiagent = worktodo.front().aiAgentGlobalID;
	worktodo.pop_front();
	return aiagent;
}

AIThinkManager::AIThinkManager()
{
	ScannerConnID = 0; autoupdates=false;
	executor = NULL;
//	dba = new MySQL();
	//dba.setupDefaultThesisParams();
	//querytext = new char [300];
	//Con::printf("AIThinkManager - Init complete");
}

//-----------------------------------------------------------------------------
AIThinkManager::~AIThinkManager()
{
	//dba.Close();
	//delete [] querytext;
	//Con::printf("AIThinkManager - Destruct complete");
//	if (executor != NULL)
//		delete executor; // Not needed - mission cleanup does this
}

//-----------------------------------------------------------------------------


void AIThinkManager::SendOutWork()
{
	int i = 0;
	bool nowork = false;
	int agentid;
	while (i<knownaiprocessors.size() && !nowork)
	{
		if (knownaiprocessors[i].currentstatus ==
knownaiprocessors[i].READY)
		{
			agentid = PopAgent();
			if (agentid != -1)
```

```
                        {
                                knownaiprocessors[i].currentstatus =
knownaiprocessors[i].BUSY;

        Con::evaluatef("CommandToClient(%d,'RecvWork',%d,%d.GetGhostIndex(%d));",k
nownaiprocessors[i].connID, agentid,knownaiprocessors[i].connID,agentid);
                        } else {
                                nowork = true;
                        }
                        //Con::printf("Status: %d  - READY",
knownaiprocessors[i].connID);
                }
                i++;
        }

        if (autoupdates)
                Con::evaluatef("schedule(1000,0,\"runThinkMgrOnce\");");
}

void AIThinkManager::processTick(const *Move)
{
        //if (autoupdates)
        //      SendOutWork();
        // Ze goggles - ze do nothing!
}

void AIThinkManager::SetAutoUpdates(bool doauto)
{
        autoupdates = doauto;
        //Con::evaluatef("schedule(1000,0,\"AIThinkManager::SendOutWork\");");
        SendOutWork();
}

void AIThinkManager::MakeWorkOrder(F32 aiAgentGlobalID)
{
        bool found = false;
        int i = 0;
        while (!found && i<worktodo.size())
        {
                if (worktodo[i].aiAgentGlobalID == aiAgentGlobalID)
                        found = true;
                else
                        i++;
        }

        if (!found) // don't readd to workorder queue if it's already in it
        {
                workorder neworder;
                neworder.aiAgentGlobalID = aiAgentGlobalID;
                worktodo.push_back(neworder);
        }
}

void AIThinkManager::DisplayWorkToDo()
{
        int i = 0;
        while (i<worktodo.size())
        {
                Con::printf("Status: %d - GlobalID:
%d",i,worktodo[i].aiAgentGlobalID);
                i++;
        }
}
```

```
void AIThinkManager::DisplayProcessors()
{
	int i = 0;
	while (i<knownaiprocessors.size())
	{
		if (knownaiprocessors[i].currentstatus ==
knownaiprocessors[i].BUSY)
			Con::printf("Status: %d  - BUSY",
knownaiprocessors[i].connID);
		else if (knownaiprocessors[i].currentstatus ==
knownaiprocessors[i].READY)
			Con::printf("Status: %d  - READY",
knownaiprocessors[i].connID);
		else if (knownaiprocessors[i].currentstatus ==
knownaiprocessors[i].OFFLINE)
			Con::printf("Status: %d  - OFFLINE",
knownaiprocessors[i].connID);
		i++;
	}
	if (ScannerConnID==0)
		Con::printf("Scanner:  Not found.");
	else
		Con::printf("Scanner: %d",ScannerConnID);
}

void AIThinkManager::ProcStatusChange(F32 connID,aithinkinfo::status newstat)
{
	bool notfound = true;
	int i = 0;
	while (notfound && i<knownaiprocessors.size())
	{
		if (knownaiprocessors[i].connID == connID)
			notfound = false;
		else
			i++;
	}
	if (!notfound)
		knownaiprocessors[i].currentstatus = newstat;
	else
		Con::printf("AITHINKMANAGER - Warning - trying to change a
processor status when processor is not known");
}

void AIThinkManager::AddAIProc(F32 connID)
{
	bool found = false;
	int i = 0;
	while (!found && i<knownaiprocessors.size())
	{
		if (knownaiprocessors[i].connID == connID)
			found = true;
		else
			i++;
	}

	if (!found) // don't readd to aithink queue if it's already in it
	{
		aithinkinfo newproc;
		newproc.connID = connID;
		newproc.currentstatus = newproc.OFFLINE;
		knownaiprocessors.push_back(newproc);
	}
```

```
}

void AIThinkManager::SetScanID(F32 connID)
{
	ScannerConnID = connID;
}

S32 AIThinkManager::GetScanID()
{
	return ScannerConnID;
}

void AIThinkManager::SetupWorld()
{
	//todo
	if (executor == NULL)
	{
		executor = new AIThinkExec();
		executor->registerObject();
		executor->SetupWorld(); // This only should ever be run once
	}
}

void AIThinkManager::NewAgent()
{
	//todo
	if (executor == NULL)
	{
		Con::warnf("Cannot make new agents until world is inited");
	} else {
		executor->AddAgent();
	}
}

// ----------------------------------------------------------
// console methods
// ----------------------------------------------------------

// =========== WORK CONSOLE FUNCTIONS ============================

ConsoleStaticMethod( AIThinkManager, DisplayWorkToDo, void, 1, 1, " () "
                                    "Displays work left in work to do list\n\n")
{
   //gBanList.addBan( dAtoi( argv[1] ), argv[2], dAtoi( argv[3] ) );
	//Con::printf("w00t!\n");
	gAIThinkManager.DisplayWorkToDo();
}

ConsoleStaticMethod( AIThinkManager, MakeWorkOrder, void, 2, 2, " (int globalID) "
                                    "Adds a work order to work order list\n\n"
                                    "@param   globalID       Global (mysql) ID
of the ai agent that needs processing.\n")
{
   //gBanList.addBan( dAtoi( argv[1] ), argv[2], dAtoi( argv[3] ) );
	//Con::printf("w00t!\n");
	gAIThinkManager.MakeWorkOrder(dAtoi( argv[1] ));
}

// =========== PROCESSOR CONSOLE FUNCTIONS ============================

ConsoleStaticMethod( AIThinkManager, DisplayProcStatus, void, 1, 1, " () "
                                    "Displays ai processors known\n\n")
{
```

```
   //gBanList.addBan( dAtoi( argv[1] ), argv[2], dAtoi( argv[3] ) );
        //Con::printf("w00t!\n");
        gAIThinkManager.DisplayProcessors();
}

ConsoleStaticMethod( AIThinkManager, SpawnAI, void, 2, 2, " () "
                                    "Spawns a new AIThink processor on a
connection\n\n"
                                    "@param   connID       Connection ID of the
processor.\n")
{
   //gBanList.addBan( dAtoi( argv[1] ), argv[2], dAtoi( argv[3] ) );
        F32 conn = dAtoi(argv[1]);
        Con::printf("Spawning AI processor on %s\n", argv[1]);
        gAIThinkManager.AddAIProc(conn);
        Con::evaluatef("CommandToClient(%s,'AIThinkSpawn');",argv[1]);
}

ConsoleStaticMethod( AIThinkManager, ProcReady, void, 2, 2, " (int globalID) "
                                    "Updates a processor's status\n\n"
                                    "@param   connID       Connection ID of the
processor.\n")
{
        gAIThinkManager.ProcStatusChange(dAtoi(argv[1]), aithinkinfo::READY);
}

ConsoleStaticMethod( AIThinkManager, ProcBusy, void, 2, 2, " (int globalID) "
                                    "Updates a processor's status\n\n"
                                    "@param   connID       Connection ID of the
processor.\n")
{
        gAIThinkManager.ProcStatusChange(dAtoi(argv[1]),aithinkinfo::BUSY);
}

ConsoleStaticMethod( AIThinkManager, ProcOffline, void, 2, 2, " (int globalID) "
                                    "Updates a processor's status\n\n"
                                    "@param   connID       Connection ID of the
processor.\n")
{
        gAIThinkManager.ProcStatusChange(dAtoi(argv[1]),aithinkinfo::OFFLINE);
}

ConsoleStaticMethod( AIThinkManager, ScanID, void, 2, 2, " (int globalID) "
                                    "Updates a processor's status\n\n"
                                    "@param   connID       Connection ID of the
processor.\n")
{
        gAIThinkManager.SetScanID(dAtoi(argv[1]));
}

ConsoleStaticMethod( AIThinkManager, SendOutWork, void, 1, 1, " () "
                                    "Sends out work\n\n")
{
        gAIThinkManager.SendOutWork();
}

ConsoleStaticMethod( AIThinkManager, NewAgent, void, 1, 1, " () "
                                    "Makes A New Agent\n\n")
{
        gAIThinkManager.NewAgent();
}

ConsoleStaticMethod( AIThinkManager, autoupdates, void, 2, 2, " (int doit) "
```

```
                                        "Enable autoupdates (1/0)\n\n"
                                        "@param   doit (1/0)       Whether or not to
do autoupdates.\n")
{
        gAIThinkManager.SetAutoUpdates(dAtoi(argv[1]));
}

ConsoleStaticMethod( AIThinkManager, getscannerID, S32, 1, 1, " () "
                                        "returns ID of scanner\n\n"
                                        "")
{
        return gAIThinkManager.GetScanID();
}
// ========== MISC CONSOLE FUNCTIONS ============================

ConsoleStaticMethod( AIThinkManager, SetupWorld, void, 1, 1, " () "
                                        "Sets up the world\n\n")
{
        gAIThinkManager.SetupWorld();
}
//------------------------------------------------------------------------------
```

Appendix F: AIThinkScan.h Source Code

```
//-----------------------------------------------------------------------------
// Robby Zinchak Thesis
// Copyright (C) Robby Zinchak
//-----------------------------------------------------------------------------

#ifndef _AITHINKSCAN_H_
#define _AITHINKSCAN_H_

#ifndef _SIMBASE_H_
#include "console/simBase.h"
#endif
#ifndef _TVECTOR_H_
#include "core/tVector.h"
#endif
#ifndef _mysql_H
#include "game/mysql/fimysql.h"
#endif

/// This class figures out which aiagents need to be directed, and sends messages
to the aithinkmanager to act
class AIThinkScan : public SimObject
{
   typedef SimObject Parent;
   MySQL dba;
   Vector <S32> IDlist;
   bool autoupdates;
   char * querytext;

   int delim; // Helps control how often we decrease statuses
public:

   AIThinkScan();
   ~AIThinkScan();
   void RunScan();
   void processTick(const *Move);
   void SetAutoUpdates(bool doauto);
   void AgentSTRSearch(char *text, int agentID);

   DECLARE_CONOBJECT(AIThinkScan);
};



#endif
```

Appendix G: AIThinkScan.cc Source Code

```
//-----------------------------------------------------------------------------
// Robby Zinchak Thesis
// Copyright (C) Robby Zinchak
//-----------------------------------------------------------------------------

#include "thesis/AIThinkScan.h"
#include "console/consoleTypes.h"
#include "core/resManager.h"
#include "core/fileStream.h"

IMPLEMENT_CONOBJECT(AIThinkScan);
//AIThinkManager gAIThinkManager;

//-----------------------------------------------------------------------------

void AIThinkScan::RunScan()
{

	// 1.  Every set interval, all attached statuses for each AI agent will be
evaluated.
	// 2.  The status that is closest to BadLevel will be set as the active
problem.
	// 3.  If this means a change on the current problem, then throw out the
old Queue Tree and make a new one.

	// First save a list with all ID's to scan
	// select agentID from statusmap order by agentID desc
	IDlist.clear();
	dba.Query("select agentID from statusmap order by agentID desc");
	S32 result = dba.StoreResult();
	if (result == 0)
	{
		Con::printf("ERROR:  StoreResult failed");
		return;
	}

	int agentid;

	for (int i= 0; i< dba.NumRows(result); i++)
	{
		dba.FetchRow(result);

		agentid = atoi(dba.GetRowCell(result,"agentID"));
		if (IDlist.size() < 1 || IDlist.last() != agentid)
			IDlist.push_back(agentid);

	}

	dba.FreeResult(result);



	// **** Should the below part be in AIThink instead of AIThinkScan?  I
think so...
	// Also, this function needs to cover timeouts and other such things!

	// Then go through list and do each ID
	while (!IDlist.empty())
	{
		agentid = IDlist.last();
```

```
		IDlist.pop_back();
		Con::printf("======= SCAN: ======= Doing AgentID %d in Scan",
agentid);

		// First:  Find the status that we *should* be working on
		//SELECT status.statusID, info, (statusmap.value - status.badLevel)
FROM statusmap, status where statusmap.statusID = status.statusID and
statusmap.agentid = *INSERTAGENTID* order by (statusmap.value - status.badLevel)
desc limit 1
		//SELECT status.statusID as 'should',
agents.currentlyWorkingOnStatusID as 'is' FROM statusmap, status, agents where
statusmap.statusID = status.statusID and statusmap.agentid = *ISNSERTAID* and
statusmap.agentid = agents.agentID order by (statusmap.value - status.badLevel)
desc limit 1
		//char * querytext = new char [300];
		dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT
status.statusID as 'should', agents.currentlyWorkingOnStatusID as 'is' FROM
statusmap, status, agents where statusmap.statusID = status.statusID and
statusmap.agentid = %d and statusmap.agentid = agents.agentID and (statusmap.value
< status.okLevel or status.okLevel = -1) order by (statusmap.value -
status.badLevel) asc limit 1",agentid);
		Con::printf("Does this break things: %s", querytext);
		dba.Query(querytext);
		result = dba.StoreResult();
		if (result == 0)
		{
			Con::printf("ERROR:  StoreResult failed");
			return;
		}
		//delete querytext;
		if (dba.NumRows(result) < 1)
		{
			dba.FreeResult(result);
			Con::printf("Warning:  No statuses found for an agent %d",
agentid);
		} else {

			dba.FetchRow(result);
			bool needsprocess = false;
			bool choosenewstep = false;
			if (atoi(dba.GetRowCell(result,"should")) !=
atoi(dba.GetRowCell(result,"is")))
			{
				needsprocess = true;
				dba.FreeResult(result);
				Con::printf("Should != IS");
			} else {
				Con::printf("Things are fine?");
				dba.FreeResult(result);
				// Many other conditions which may require a
process:
				// *** Code from AIThink

				dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT
a.currentQueueStepID, a.currentQueueStepTimesOutWhen, a.queueTreeRootID,
a.agentInGameServerID FROM agents a where agentID = %d",agentid);
				dba.Query(querytext);
				result = dba.StoreResult();
				if (result == 0)
				{
					Con::printf("ERROR:  StoreResult failed");
					return;
				}
```

```
                                dba.FetchRow(result);
                                int currentQueueStepID =
atoi(dba.GetRowCell(result,"currentQueueStepID"));
                                int queueTreeRootID =
atoi(dba.GetRowCell(result,"queueTreeRootID"));
                                int ingameID =
atoi(dba.GetRowCell(result,"agentInGameServerID"));
                                double currentQueueStepTimesOutWhen =
atof(dba.GetRowCell(result,"currentQueueStepTimesOutWhen"));
                                //Con::printf("Timeout at: %f",
currentQueueStepTimesOutWhen);
                                dba.FreeResult(result);


                                // FOUR CASES =====
                                // Step is currently executing & is within time ->
Do nothing
                                if (currentQueueStepID != -1 && ((U32)
currentQueueStepTimesOutWhen) > Platform::getVirtualMilliseconds())
                                {
                                        Con::printf("  Step execution is within time
(thinkscan timeout > now; %f > %d)", currentQueueStepTimesOutWhen,
Platform::getVirtualMilliseconds());

                                        // Check to see if it is done:
                                        // Run complete check only if one such
completetecheck exists

        dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT c.completeCheck,
b.nodeID FROM agents a, queuenode b, step c where a.currentQueueStepID = b.nodeID
and b.stepID = c.stepID and a.agentID = %d",agentid);
                                        dba.Query(querytext);
                                        result = dba.StoreResult();
                                        if (result == 0)
                                        {
                                                Con::printf("ERROR:  StoreResult
failed");
                                                return;
                                        }
                                        dba.FetchRow(result);
                                        //if
(dba.GetRowCell(result,"completeCheck"))
                                        if (true) // should not cause a problem,
everything *needs* a complete check
                                        {

        dSprintf(querytext,2900/*sizeof(querytext)*/,"%s",
dba.GetRowCell(result,"completeCheck"));
                                                int nodeID =
atoi(dba.GetRowCell(result,"nodeID"));
                                                //AgentSTRSearch(querytext,ingameID);

                                                AgentSTRSearch(querytext,agentid);

                                                //Con::printf(querytext);
                                                Con::printf("Scanner: %s",
querytext);
                                                if (atoi(Con::evaluatef("if (%s) {
return 1; }",querytext )) == 1) // SUCCESS!
                                                {

                                                        choosenewstep = true;
                                                        Con::printf("FYI: done");
```

```
                                        } else
                                                Con::printf("FYI: Not done");
                                        }
                                        dba.FreeResult(result);

                                }

                                // Step is currenlty executing & is out of time ->
New step, Mark old harder
                                if (currentQueueStepID != -1 && ((U32)
currentQueueStepTimesOutWhen) < Platform::getVirtualMilliseconds())
                                {
                                        Con::printf("Step is currenlty executing &
is out of time -> New step, Mark old harder");
                                        choosenewstep = true;
                                }
                                // No current step -> New Step
                                if (currentQueueStepID == -1)
                                {
                                        Con::printf(" No current step -> New Step");
                                        choosenewstep = true;
                                }

                                // *** End code from AIThink

                        }

                        if (needsprocess || choosenewstep)
                        {
                                //Let AIManager know that this agent needs to be
processed

        Con::evaluatef("CommandToServer('MakeWorkOrder',%d);",agentid);

        //Con::printf("CommandToServer('MakeWorkOrder',%d);",agentid);
                                Con::printf("making work order");
                        }


                        //dba.FreeResult(result);
                }
        }

        // ============================== BAD STUFF
==============================
        dSprintf(querytext,2900/*sizeof(querytext)*/,"SELECT agents.agentID,
action, agentInGameServerID FROM statusmap, status, step, agents where
badStuffStepID = stepID and value <= badLevel and agents.agentID =
statusmap.agentID and lastBad < badFrequency +
%d",Platform::getVirtualMilliseconds());
        dba.Query(querytext);
        result = dba.StoreResult();
        if (result == 0)
        {
                Con::printf("ERROR:  StoreResult failed");
                return;
        }

        for (int i= 0; i< dba.NumRows(result); i++)
        {
                dba.FetchRow(result);

                agentid = atoi(dba.GetRowCell(result,"agentID"));
```

```
                Con::printf("Scan: Doing bad stuff for %d", agentid);

                // Execute the attached command for bad stuff on this agent:
                dSprintf(querytext,2900,"commandToServer('ExecCMD', \"%s\");",
dba.GetRowCell(result,"action"));
                AgentSTRSearch(querytext,agentid);
                Con::evaluatef(querytext);

                // Update bad stuff time
                dSprintf(querytext,2900/*sizeof(querytext)*/,"update agents set
lastBad = %d where agentID = %d", Platform::getVirtualMilliseconds(), agentid);
                dba.Query(querytext);
        }

        dba.FreeResult(result);

        // ========================== STATUS DECREMENT COUNTER
==============================
        // Make every status a bit less:
        if (delim <= 0)
        {
                dba.Query("update statusmap set value=value-1");
                delim = 10;
        } else
                delim--;

        // ============================ NEW SPAWNER
============================================

        // If we have achieved being not hungry and have 1000 fun, make a new
agent and reset our stats
        dba.Query("SELECT agents.agentID FROM statusmap,statusmap b,agents where
agents.agentID = statusmap.agentID and statusmap.statusID = 1 and b.statusID = 2
and statusmap.value > 50 and b.value>1000");
        result = dba.StoreResult();
        if (result == 0)
        {
                Con::printf("ERROR:  StoreResult failed");
                return;
        }

        for (int i= 0; i< dba.NumRows(result); i++)
        {
                dba.FetchRow(result);

                agentid = atoi(dba.GetRowCell(result,"agentID"));

                Con::printf("Scan: New agent created due to super-happy agent %d",
agentid);

                // Execute the attached command for a new agent:
                Con::evaluatef("commandToServer('ExecCMD',
\"aithinkmanager::newagent();\");");

                // Update agent
                dSprintf(querytext,2900/*sizeof(querytext)*/,"update statusmap set
value = 30 where agentID = %d", agentid);
                dba.Query(querytext);
        }

        dba.FreeResult(result);
```

```
        // ============================== RE-RUN?
================================================
        if (autoupdates)
                Con::evaluatef("schedule(1000,0,\"ScannerScan\");");
}

//---------------------------------------------------------------------------

void AIThinkScan::AgentSTRSearch(char *text, int agentID)
{
        //Con::printf("Size of text: %d", sizeof(text));
        char *temp;
        temp = new char [300];

        int i=0;
        int j=0;
        int appeartimes=0;
        while (i<290)
        {
                if (text[i] == '<' && text[i+1] == 'A' && text[i+2] == 'G' &&
                        text[i+3] == 'E' && text[i+4] == 'N' && text[i+5] == 'T' &&
                        text[i+6] == '>')
                {
                        temp[j]='a';
                        temp[j+1]='%';
                        temp[j+2]='d';
                        j+=3;
                        i+=7;
                        appeartimes++;
                } else {
                        temp[j]=text[i];
                        j++;
                        i++;
                }
        }

        //for (int z=0; z<appeartimes; z++)
        //{
        //      if (z>0)
//                      dSprintf(temp,sizeof(temp),text);

//              agentID = atoi(Con::evaluatef("ServerConnection.GetGhostIndex( %d
));", agentID));

        dSprintf(text,2900,temp,agentID,agentID,agentID,agentID,agentID,agentID,ag
entID,agentID,agentID,agentID,agentID,agentID,agentID,agentID);
//      }
        // Problems: Cannot write to self?  Spaces before/after number?


        delete [] temp; temp = NULL;
}

//---------------------------------------------------------------------------

AIThinkScan::AIThinkScan()
{
//      dba = new MySQL();
        autoupdates = false;
        dba.setupDefaultThesisParams();
        querytext = new char [3000];
        Con::printf("AIThinkScan - Init complete");
        delim = 0;
```

```
}

//-----------------------------------------------------------------------------
AIThinkScan::~AIThinkScan()
{
	dba.Close();
	delete [] querytext; querytext = NULL;
	Con::printf("AIThinkScan - Destruct complete");
}

//-----------------------------------------------------------------------------

void AIThinkScan::processTick(const *Move)
{
	if (autoupdates)
		RunScan();
}

void AIThinkScan::SetAutoUpdates(bool doauto)
{
	autoupdates = doauto;
	RunScan();
}
// ----------------------------------------------------------
// console methods
// ----------------------------------------------------------

ConsoleMethod( AIThinkScan, Awaken, void, 2, 2, ""
                                    "Let server know we're ready\n\n")
{
	Con::evaluatef("CommandToServer('ScanAmReady');");
}

ConsoleMethod( AIThinkScan, Scan, void, 2, 2, ""
                                    "Run a single scan\n\n")
{
	//Con::evaluatef("CommandToServer('ScanAmReady');");
	object->RunScan();
}

ConsoleMethod( AIThinkScan, autoupdates, void, 3, 3, " (int doit) "
                                    "Enable autoupdates (1/0)\n\n"
                                    "@param   doit (1/0)       Whether or not to
do autoupdates.\n")
{
	//Con::evaluatef("CommandToServer('ScanAmReady');");
	object->SetAutoUpdates(dAtoi(argv[2]));
}

//-----------------------------------------------------------------------------
```