Developing an Agent-Based Training Simulation using Game and Virtual Reality Software: Experience Report.

Debbie Richards
Computing Department,
Macquarie University
North Ryde, Australia, 2109
richards@ics.mq.edu.au

John Porte
Computing Department,
Macquarie University
North Ryde, Australia, 2109
jporte@ics.mq.edu.au

ABSTRACT
This paper reports our usage of game and virtual reality technology to build a risk training simulation inhabited by agents. We describe the design and issues in the development of the Risk Management Module using UrealEd 3.0 game toolkit and the BOrder Security System using Vizard so that others with similar goals might be informed by our experiences.

Categories and Subject Descriptors
D.2 SOFTWARE ENGINEERING D.2.2 [**Design Tools and Techniques**] J. Computer Applications J.1 ADMINISTRATIVE DATA PROCESSING: *Military*

General Terms
Design, Security, Human Factors, Languages.

Keywords
Training Simulation, Game Technology, Agent-Based Systems.

1. INTRODUCTION
We began a project in late 2004 to develop a training environment for learning how to manage risky situations. Our initial focus was on crime and terrorism and providing experiential learning environments which exposed police to dangerous and critical decision making incidents without the cost (in terms of safety) associated with on the job training. To leverage the capabilities in sound, graphics, physics engines, etc provided in games engines we decided to take an existing games system as our base. The continued and growing interest in interactive entertainment ensured the development of more sophisticated engines for displaying 3D worlds could be adopted/integrated as necessary in the future. Taking a component-based approach, we wanted to be able to plug in available commercial and research software where appropriate and also add our own modules. This would allow us to concentrate on our specific modules while also allowing some experimentation with other techniques such as exploring the benefits of different planners or reasoners. Our usage of off-the-shelf simulation and game software and freeware was in keeping with our goal to develop a training system aimed at the low end of the market. The components we sought to develop included a natural language generator, story manager, story base, knowledge base, intelligent agent reasoner and knowledge acquisition interface.

Agents were seen to play a pivotal role in our training environment. Agents are programs that perform tasks. They are currently being applied in domains as diverse as computer games and interactive cinema, information retrieval and filtering, user interface design, electronic commerce, autonomous vehicles and spacecraft, and industrial process control. Agents can have any number of characteristics, including being autonomous, having reactivity, social ability and pro-activeness. An autonomous characteristic means that agents can operate without the direct intervention of humans. Reactivity means that agents can perceive their environment and respond in a timely fashion to changes that occur in it. Social ability means that they can interact with other agents, and pro-activeness enables agents to exhibit goal-directed behaviour by taking the initiative. The prototypes we have developed have implemented two alternative views of agents. The non-player characters (NPC) in our simulations are themselves agents. Also, we have designed agent-based architectures [5] in which components of the system have been agents (following the definition given) and also which combine to achieve our NPC agents.

The first implementation of our system, known as Risk Management Mod (RMM), used a modification of the Unreal Tournament 2004 (UT2004) game engine, with agents driven by an external Java based Behavioral Engine, Narrative Engine, and Game Master Controller. Agents were semi automated by the Game Engine and bound by Game Engine rules.

The second implementation of our system used the Vizard virtual reality toolkit as the Graphics Engine, agents were again driven externally from a C++ based Simulation Controller and Knowledge Base. In this system the agents were not automated by the toolkit, all behavior was developed from the ground up. Some of the advantages, disadvantages and obstacles the two systems encountered will be described in this paper. Prior to conclusions we review recent literature using agents in games and simulations.

IE '09, December 17-19, 2009 Sydney, Australia

2. RMM - Games-Based Prototype (Unreal Tournament)

A number of development toolkits and game environments including programmable game engines are available, each with various strengths and weaknesses. Game engines often include a rendering engine to output 2D or 3D graphics, animation package, a physics engine (or other functionality to handle object collision), sound synthesizer and scripting language. Some incorporate more sophisticated tools and techniques such as networking, threading, scene graphs and ideas from artificial intelligence such as agents and rule/knowledge bases. Often the developer/user will be able to choose functionality from a range of options. For instance, Garage Games[i] provides tools for creating 2D, 3D and console games using the Torque game builder and game engine together with academic resources useful for teaching games development.

Different types of games can be developed depending on the game engine, for example, Torque, supports development of First Person Shooter (FPS) Games. Neverwinternights[ii] allows game development, using the Aurora Toolset and game engine, of third person perspective computer role-playing games (CRPG). Neverwinternights is a dungeons and dragons fantasy game which was the first massively multiplayer online RPG (MMORPG), now eclipsed by games such as World of Warcraft, but remains quite revolutionary in allowing users to host their own MMORPG server and to create their own worlds and adventures with up to 64 of their friends[iii].

Middleware such as the general purpose Gamebryo[iv] System Development Kit (SDK) has allowed the development of robust and complete games and even supported the development of more customized/purpose-built toolkits such as The Elder Scrolls IV: Oblivion which use the gamebryo[v] engine. Similarly the IdTech Engine, developed by Id Software in association with Valve[vi], launched in the first FPS Wolftenstein 3D[vii] in 1991, has been an integral part of successful games such as Call of Duty, Soldier of Fortune, Half-Life, Medal of Honor: Allied Assault, Star Trek: Elite Force, Heretic, Hexen and not to forget DOOM and QUAKE.

Having decided to use a game platform, we considered a range of game engines, development environments and toolkits including Torque, AURORA, Hammer 4, Havok 2 and Unreal Ed 3.0. The Torque engine at the time was the most popular "indie" game development engine, used by freelance programmers worldwide. It was versatile but was not as advanced as Unreal or Hammer 4 and not simple to use. It was coding heavy, however, it featured a built in world editor. Torque[viii] was a cross-platform game generator, and is fairly well supported community-wise.

The AURORA the toolset for the Bioware produced the CRPG Neverwinter Nights game series. While not as realistic and adaptable as UnrealEd 3.0 and Hammer 4, the AURORA toolset allowed for virtually unlimited customization and came with a built in games master/instructor feature, which was a huge bonus for anyone wishing to include a live instructor. AURORA has however mostly been used in fantasy-themed MODs, not for real-world inspired combat MODs as Hammer 4 and UnrealEd 3.0[ix] has. Hammer 4 (HL2-SDK) was the modelling tool (Source SDK) that was used to generate Half Life 2. Sporting amazing graphics and the Havok dynamics engine, it set the standard for game graphics and physical realism together with UnrealEd 3.0[x]. Havok 2 was a dynamics engine, not a development tool as such. Havok 2 was used in the commercial game Full Spectrum Warrior, a modified version of a simulation originally developed to train US infantry in urban combat by the ICT. Also used in the WWII game Medal of Honour. The Havok engine[xi] was one of the best for simulating physics, gravity, objects etc.

UnrealEd 3.0[xii] was developed by Epic Games. The Unreal engine was perhaps only second to the new Doom 3 engine at that time. The toolset is highly advanced, even in the free version that comes with most retail games using the engine, and is supported by a massive community. We choose to use Unreal Ed 3.0 as our game platform as it was cutting edge at the time with a lot of features including the ability to make your own modifications, availability of other people's NPCs and reasonably priced.

To be able to use UT2004 for our own research purposes we needed to create a modification. A modification in the game world, better known as a mod, is the alteration to a game to make something within the game function in a different way to which it was originally specified. We created the Risk Management Mod (RMM) which alters UT2004 from a first-person-shooter game to become a first-person training simulation. In Unreal specifically, there are three types of mods. Mutators are mini-mods and have limited functionality. An example mutator is altering the SniperRifle so that it would shoot more rapidly. Or a Vampire mutator so that each shot drains the life from the enemy and gives you life. The advantage of mutators is that they can be mixed with other mutator mods. Mutators do not change the gameplay.

GameTypes are much larger mods, basically the rule of thumb is, if it can't be implemented as a Mutator, then it is a GameType mod. GameTypes will often include many new weapons, pickups, AI features, or special actors. It is a whole new gameplay. Other types of mods include a total conversion mod. It is usually an entirely new game that uses the same engine. It is similar to a GameType mod, but it is not accessed within the Unreal GameType menu. We created a GameType mod.

With similar goals to ours and also choosing to use UT, was the Gamebots project at the University of Southern California's Information Sciences Institute that sought to turn the game Unreal Tournament into a domain for research in artificial intelligence. Gamebots is a freely available[xiii] modification that allows external programs to connect via sockets to Unreal Tournament and control the characters in the game. The game feeds sensory information for the character over the network connection. Based on this information, the client program can decide what actions the character should take and issues commands back over the network to the game to have the character move, shoot, talk, etc. Following are sample external client programs that control the bots for the Gamebots mod:

- Java Bot – a Java application for launching and managing Java based bots.
- Soarbot – a simple bot built with TCL/TK and SOAR.
- Tclbot – a very simple bot built in TCL.

Specific maps called DOM-Stalward and CTF-Simple are also required to run the the Gamebot mod.

We chose not to use Gamebots because the map files have *.unr extensions meaning that it is created for the first version of Unreal Tournament which was released in 1999. We wanted to use the current version and features in Unreal Tournament 2004 which had different map files with the extension *.ut2. Since Gamebots is created for an older version of Unreal Tournament, the code cannot compile with the 2004 version we were working with. Koester et al. [6] give similar reasons for not taking advantage of Gamebots themselves. They also note that for their purposes of building autonomous robots they needed a Peer2Peer approach rather than a client-server architecture in which the model/knowledge is held by the server.

As presented in this section, there were numerous game engines and development toolkits that were possibilities for the risk management training environment we were seeking to develop. From this selection we chose Unreal Tournament 2004 because of the features it supported but created our own Mod so that we could support the narrative and agent aspects of our research prototype. However, as we describe next, we were to discover that UT2004 posed a number of issues.

2.1 Issues with Unreal Tournament

Initial problems with Unreal Tournament included common issues faced with using a game development kit such as not having characters which are appropriate for the domain to be simulated. We created an airport world (Fig 1a) and then tried to find characters to act as our agents (customs officer trainee, customs supervisor and passenger). To find suitable characters we searched for available characters that could be used in UT. UT is a first-person shooter game. The learning curve to use the game engine to create our own games/simulations was steep even for an experienced programmer. As you can see in Figure 1b, the characters we found (e.g. agent Smith from the Matrix) had guns in their hands and it took considerable effort to remove that gun. In our initial scenarios and studies our animations were awkward, for example, characters moving sideways through doors, sliding along or performing actions we had not specified but which had been automated by UT agents in order to make the experience more “action-packed” and dramatic. These automated actions sometimes enhanced the scenario by creating ambient movement, and at other times were entirely inappropriate, such as an agent getting jammed between a seat and a door and exploding into a fountain of blood and gore.

Within the Airport World we have created a number of customs scenarios. The virtual airport, shown in Figures 1a and 1b, was created using UnrealEd 3.0, which is the virtual environment editing tool that comes with UT2004. Dialogue was achieved via the use of speech bubbles as shown in Figure 1b and also generated using the text-to-speech (TTS) language generator in UT2004.

Creating a virtual environment in UnrealEd like most other editing tools requires the user to build into an empty space that is portrayed in Cartesian dimensions: X, Y and Z being the length, height and depth of the space. Rooms and objects are then placed into this empty space by adding or subtracting from primitive objects like cubes and spheres to form more complex looking objects. Each object can then be textured to add more realism otherwise all objects would be plain white. Lighting is added to create shadow effects and to add more depth in the environment. More detailed information on creating virtual environments in UnrealEd can be found in [3].

(a)

(b)

Figures 1a and 1b: Screenshots of the Virtual Airport using the Unreal Tournament game engine.

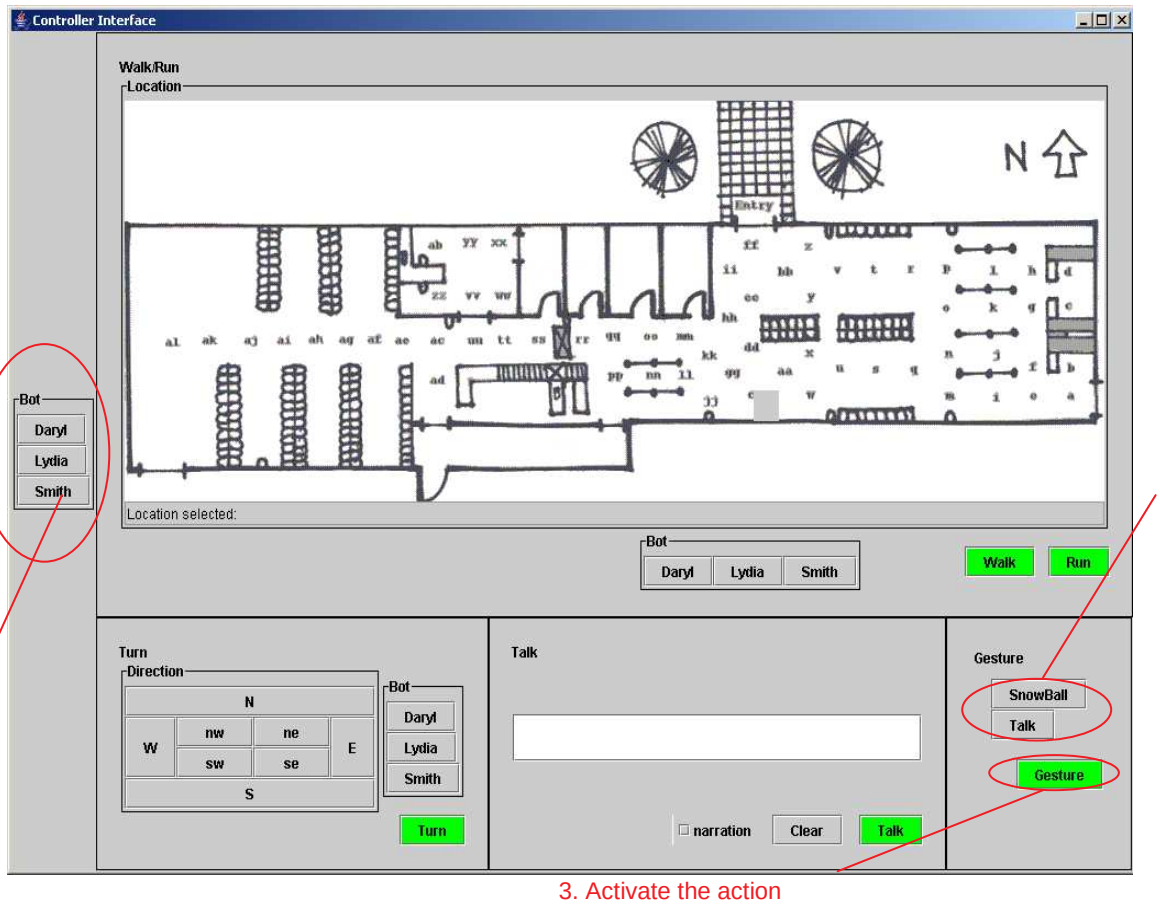

Figure 2: RMM Controller Interface

Figure 3: RMM and our NPCs moving around in our environment.

In order for the Non-Playable Character (NPC) agents to navigate around this virtual environment a network of path nodes are added, which are invisible during gameplay. The AI code that drives an NPC agent doesn't have the power to interpret the 3D world as the user sees it. Therefore the network of path nodes guide the agents around the map allowing the agent to determine which nodes are safe and easy to travel to. Solely creating an Airport World doesn't allow for a Narrative Engine or Game Master Control to plug into U2004.

RMM spawns and allows for control of the agents in the Airport World. It uses socket connections to feed information between the Narrative Engine or Game Master Control to UT2004 and vice versa. In this case both the Narrative Engine and the Game Master Control are external client programs where the agents decisions and actions such as walk, run, turn and talk can be controlled. To allow a user to manipulate or control the characters or bots inside the game, it would be ideal if the controller interface were integrated into the RMM, but due to: uncertainty of what Game Engine we would be using in the future it was thought best to keep the controller interface separate. Furthermore, it would require more programming than the time and budget allowed. The controller interface created is shown in Figure 2. The response in the airport world is shown in Figure 3.The UT2004 game engine provided us with an animation engine, able to execute basic commands like "X walk towards Y", "X says content to Y", etc. The Story Engine (we used IDtension [16]) provided us with a high level actions generator, which outputs commands like "X encourages Y to perform t", "X insults

Y", etc. However, a level was missing between these two engines: The behaviour level. We needed a Behaviour Engine.

A behaviour engine is complex to design, because it has to solve the following AI related issues:

- composing sequences of animations into a behaviour
- provide alternative animations, depending on the context
- provide alternative animations in case of failure
- run animations in parallel (gestures and speech, typicially)
- manage priorities between competing behaviours (those who share the same ressources)
- blend behaviours
- interrupt behaviours and restart them
- synchronize parallel running behaviours
- synchronize joint behaviours between several characters

The resulting systems are usually rather complex, but provide a great level of flexibility. However, these systems are usually complex to author: Based for example on script programming, or finite state machines, they require a programmer. Furthermore, they usually implement a complex AI-based architecture, based on goals, which is problematic in the case of the integration with a Narrative Engine such as IDtension, because goals are already managed at the narrative level.

Thus, we sought to design a Behaviour Engine (BE) [17] with the following properties:

- easy to author
- less powerful than the state of the art BEs, but still more complex than a basic sequencer
- implemented as a independent module, inserted between the Narrative Engine and the Game engine, or called by any other program (like Python scripts for example).

Work on the BE has continued for the purposes of connecting with the Narrative Engine. However, we found that this work was focused on storytelling which seeks to include a plot, goal, surprise elements and was more focused on entertainment than education. For the purposes of a training environment with pedagogical goals and also because we wanted an approach that would allow the trainer to create their own scenarios and update their own knowledge (the narrative engine required knowledge of formal logic) we decided that we would need to find an alternative.

RMM was used in three studies: one evaluating if participants were able to learn by watching a scenario created in a game environment (specifically UT) as compared to a similar video-taped role-play with real people [11]; a second evaluating whether more learning occurred by watching a game-based scenario or by participating in that scenario [10] and a third which evaluated an implementation of the narrative engine.

3. BOSS – Graphics Engine-Based Prototype (Vizard)

In 2006 the project was in a state suitable for experimental use (as reported in the previous section), and further development slowed for a time. A virtual reality laboratory was being founded and it was hypothesized that immersion could have a positive impact on training, it was decided to make a new risk management training environment/simulation system taking advantage of the new equipment. In order to make use of specific VR technology such as stereo 3D graphics and panoramic cave field of view, and PC clustering, Unreal Tournament was no longer a possible platform. We developed the system in Vizard, a scene graph package that uses Open Scene Graph as the back end, and Python scripting for the front end.

We developed a system, known as BOrder Security System (BOSS) in which the user views a scenario and interacts with an avatar in an immersive 3D virtual environment (see Figure 4). Our system uses agents to drive the avatars and some of the functionality required was available within the Vizard scene graph environment; however we had additional modules that were to be part of the system and could not be connected directly to the graphical engine. The Scenario Controller was developed to link to additional external modules such as the Knowledge Base, speech recognition / speech synthesis engine, and emotion recognition, as well as to manage the flow of scenario events.

Our agents needed access to the following:

- Logic contained within the knowledge base. In the form of ripple down rules the agent can pass known information from the scenario and receive decision making information in return.
- Speech recognition/synthesis in order to achieve verbal input/output.
- Spatial information from the graphical engine. For example, to determine if the agent is within reach of another object, or to know which direction to turn to before speaking.
- The avatar representation of each agent.

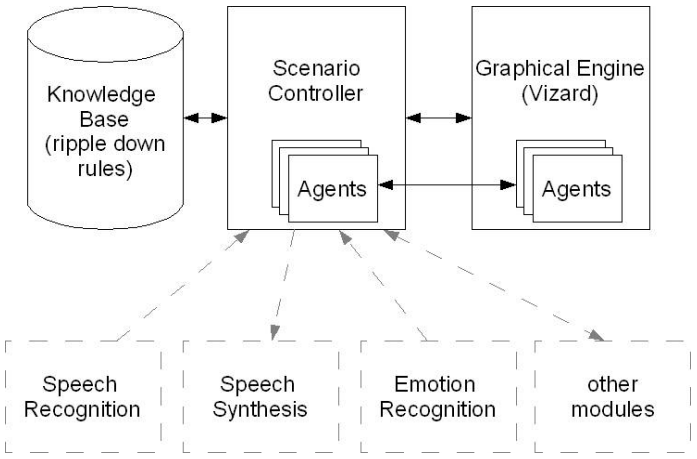

Figure 4. Structure of the system, showing core processes in solid and optional modules in dashes.

In order to achieve the needs above we implemented the agents in such a way that modules of each agent exist within both the Scenario Controller and the Graphical Engine. These modules of each agent are kept in sync where necessary. Actions dealing with the Knowledge Base, or speech interfaces are performed within the Scenario Controller module of each agent, while actions dealing with movement, animations, and interactions are performed within the Graphical Engine module of the agent.

A TCP/IP socket connection was used for communication between the Scenario Controller and the Graphical Engine. While this results in somewhat slower communication between the halves of the avatars we found that many of their states did not need to be kept tightly in sync. For example the speech synthesis components of the avatar does not need to know about current animation displayed. Those actions are independent of each other and in general can complete themselves in parallel and return to idle without further control.

An advantage of the division of processes was that different programming languages could be used to their best advantages. While the Graphical Engine running in Vizard uses Python, we chose to develop the Scenario Controller in C++, simplifying the connection to API's (Application Programming Interfaces) required for Speech and the connection to our emotion recognition module. This also had the effect that some of the agent code modules are developed in C++ while other modules are developed in python. Although it may have introduced some complications, it also allowed us to choose which environment was more suitable for the development of various agent processing modules. Additionally the agents became more modular themselves allowing functionality to be enabled / disabled as system modules are added or removed, such as the Speech Recognitions, Speech Synthesis, and Emotion Recognition modules.

Figure 5 shows the system running. The Graphical Engine, Vizard shown on the right, is run first and awaits a connection from the Scenario Controller, SynthIDE shown in the center. The Scenario Controller is a C++ developed application; hover it implements the programming language lua for its own scripts and to aid in the design of scenarios. When the Scenario Controller launches a scenario modules of each avatar are created and a message is sent to the Graphical Engine telling it to create the corresponding modules of those avatars. The scenario script continues to run within the Scenario Controller giving the agents actions to perform and dialogue to speak. These actions may be performed in either or both of the two agent modules. During the scenario the user can pause the scenario and bring up a dialogue box showing the Knowledge Base, shown on the left. The Knowledge Base contains information populated by the Scenario Controller, calculates the level of risk, and recommends the action that should be taken by the agent. The advantages and disadvantages of developing a modular agent are summarized in Table 1.

Table 1. Advantages and Disadvantages of Developing a Modular Agent

Advantages	Disadvantages
- Agents can take advantage of the strengths of multiple programming languages where appropriate. - Agents exists with direct links to external API's regardless of their location. - Agents become modular and can expand as the system foundation expands	- There is some additional overhead maintaining communication between agent components/modules - It can cause some confusion over where the agent actually exists.

The BOSS research platform we have created has enabled us to conduct a number of studies, notably one on immersion [12] and others still in progress. The ability to easily modify scenarios and indeed the work which allows the trainer to enter their own content and knowledge provides a platform to test out questions related to language technology, knowledge and reasoning and agents. Our work is increasingly focused in the believable and intelligent agent space. Thus, before concluding we consider other work in this space.

4. Related Agent and Game Work

Decision making is a key reason why agents are used in games. In the absence of sufficient human domain experts in a role playing game, Briot et al. [2] have chosen to create a decision-making agent which inhabits the SimParc environment. SimParc is aimed to provide experience with biodiversity management of protected areas (e.g., national parks). An RPG is used as there are many stakeholders and participants in such an environment. The Park manager has been designed to justify its behaviours and also generate a participatory decision. A decision theory framework is used by the manager to assist the cognitive decision making process and provide supporting rationale. This work shows an increasing use of agents in games to support human-based simulations. The work grapples with the issues of how to reconcile differences between the decisions of individuals/agents. Currently the system only considers static information about the park and the votes by players and they are looking to develop a dynamic decision model which allows decisions to evolve via negotiation.

For the purpose of education and encouraging inquiry by students, van Krevelen [7] further considers the complexity of different types of users and propose the generalized agent-based modeling environment (GAME) comprising: the users (expert, scientist and player), the information (test and train scenarios, behavior models and interaction traces) and the processes (simulate, analyze, visualize, evolve and induce).

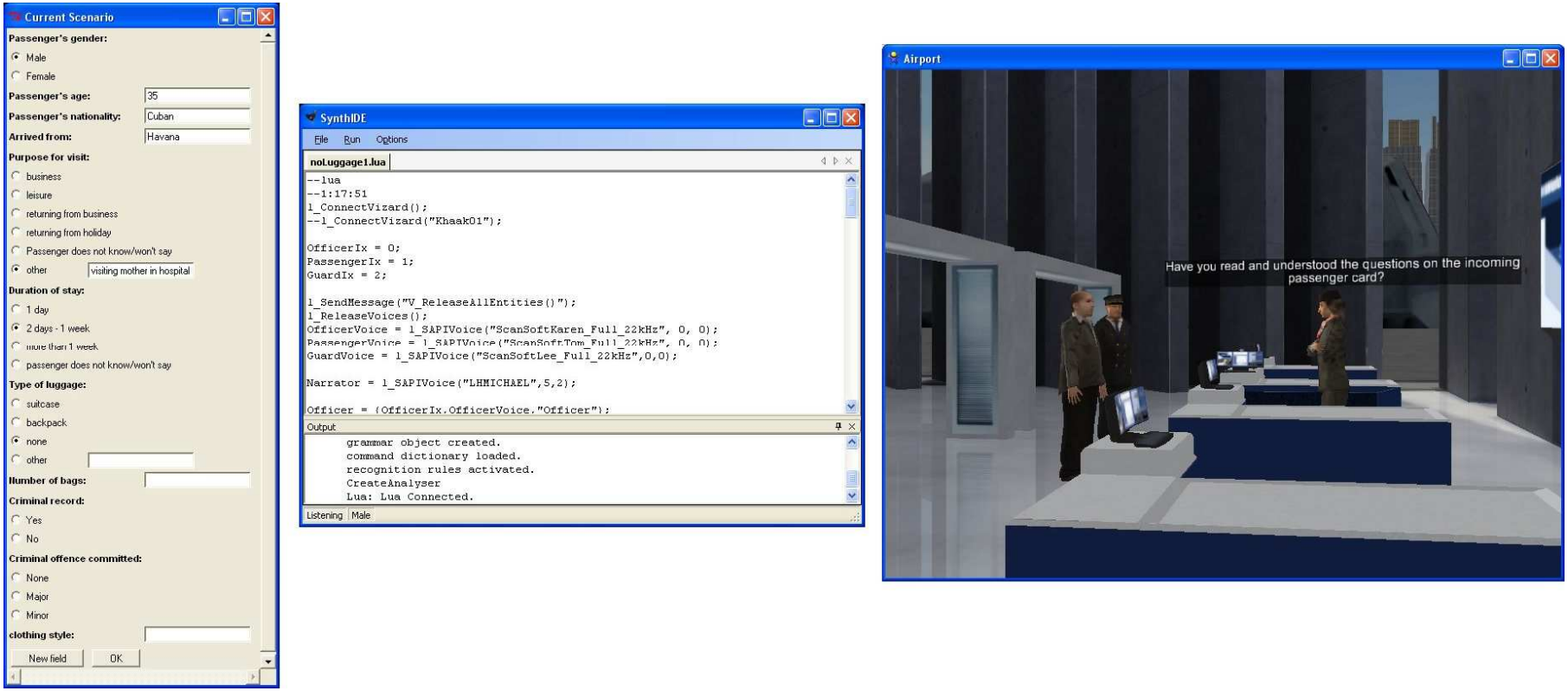

Figure 5. The three core modules running in parallel. (Knowledge Base, Scenario Controller, Graphical Engine)

Also interested in developing physically embodied cognitive agents, such as virtual agents, or non-player characters for computer games is the work by Koester et al, [6] with the programming framework Jazzyk Behavioural State Machines (BSM) which has been used to program two physical bots. As their goal is to develop autonomous robots, the virtual agents needed to be autonomous and separate from their environment. This affected the design of the model of perception and the action execution model resulting in the decision to remotely connect the agents to the environment and execute their actions asynchronously. Each agent/bot stores its own internal state and is able to change the focus of attention rapidly to determine what is happening in the external world.

The use of agents to support believability of characters in the game or simulation is common goal. Westra et al. [18] note that since characters in serious games often have a long life span, over multiple and numerous sessions, the difficulty to making them believable is increased. They have developed a game adaptation model which comprises a user model, game model and agent model to provide different plans for a goal or a range of alternative goals so that each agent to can pursue its own goals and adapt its behaviour accordingly. This agent-based approach is seen to be superior to the more typical central approach used to handle control in many games. Other agent-based work concerned primarily with believability is offered by [8] who seek to support 13-14 year olds to gain cultural awareness via interacting with autonomous characters based on a biologically-inspired theory of human action regulation taking into account perception, motivation, emotions, memory, learning and planning.

A particular perceived benefit of agents with games is the ability for agents to communication and cooperate. Aranda, Botti and Carrascosa [1] see that the existence of a common ontology provides a starting point for agents in Massively Multiplayer Online Games (MMOGs) to exchange semantic content.

Also enabling teamwork but with a particular focus on realtime interaction is the research of [9]. Using UT2004 they have developed a proxy-based tool, based on Joint Intentions, to create what they call partially observable, dynamic and stochastic environments with real-time requirements.

The use of a component-based/modular approach and the creation of test/simulation environment for exploring agent behaviour is a hallmark of agent-based approaches, as in the work we presented. Similarly, the Koko architecture [14] offers the following: (1) an extensible and reusable environment to improve developer productivity; (2) combination of multiple independent applications to provide a coherent user experience and (3) reasoning about the users affective state to support better communication and social situations.

Similarly, Gemrot et al [15] offer, an open source platform known as Pogamut 3 which supports rapid development of behaviour of virtual agents embodied in a 3D environment. They have used UT2004 to provide platform for research and educational projects

5. SUMMARY AND CONCLUSIONS

Game technology has proven to provide a valuable basis for our own and numerous other research projects including those presented in the previous section. This body of work not only utilizes existing game technology it offers the possibility to extend and improve what is currently available particularly with respect to adding intelligence in the form of reasoning, social interaction and coordination, believability and emotion.

As the needs of a specific agent driven application diverge away from gameplay and begin to require connectivity with other applications the platform used comes into question. Game Engines are generally less equipped to deal with connections to other platforms and processes, and may provide automation of your agents that is not constructive or even destructive to the behavior required in your application. Nevertheless it is important to consider the extra complexities that creating a controlling process will introduce, in terms of connectivity and segmentation of processes. It may be possible to use the best of both environments, Game/Simulation Engine, and an Agent Platform, to drive a multi-module spanning agent.

A number of projects address issues this area, some of which were introduced in the previous section. It is pleasing to see that many of the approaches combine and build upon previous work and employ common platforms, techniques, approaches and concepts leading to the expectation that many of the difficult issues facing multi-player games will be addressed.

During the development of RMM and BOSS, the former involving the creation of a mod with UT2004 and the latter involving custom development in Vizard, a number of implementation and conceptual issues arose as presented in this paper. However both continue to offer viable alternatives according to the needs of the project and resources available for exploring the research questions of interest to us in the area of training for risk management.

Acknowledgements: This work has been partially funded by an ARC Discovery DP0558852. Thanks to Jason Barles for his work with UT2004 as described in this paper.

6. REFERENCES

[1] Aranda, G., Botti, B. and Carrascosa, C. (2009). The MMOG Layer: MMOG based on MAS, In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009

[2] Briot, Jean-Pierre, Sordoni, Alessandro, Vasconcelos, Alessandro, Melo, Gustavo de Azevedo Irving, Marta and Alvarez. Isabelle (2009) Design of a Decision Maker Agent for a Distributed Role Playing Game - Experience of the SimParc Project , In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009

[3] Busby, J. (2005) Mastering Unreal Technology: The Art of Level Design, Sams Publishing.

[4] Johnson, W.L.: (2007). Serious use of a serious game for language learning. In: R. Luckin et al. (Eds.), Artificial Intelligence in Education, 67-74. Amsterdam: IOS Press.

[5] Kavakli, M., Richards, D., Dras, M., and Porte, J. (2007) An Immersive Virtual Reality Training Simulation for Risk Management, SimTecT 2007: Simulation Conference: Simulation - Improving Capability and Competitiveness, 4 - 7 June 2007, Brisbane Convention Centre, 1-6.

[6] Köster, Michael Novák, Peter, Mainzer, David and Fuhrmann. Bernd (2009) Two Case Studies for Jazzyk BSM In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009.

[7] van Krevelen, D.W.F. (2009). Intelligent Agent Modeling as Serious Game In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009.

[8] Lim, Mei Yii , Dias, Joao , Aylett, Ruth and Paiva, Ana.(2009) Intelligent NPCs for Education Role Play Game In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009.

[9] Monteiro, Ivan M. and Alvares, Luis O. (2009). A Teamwork Infrastructure for Computer Games with Real-Time Requirements In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009

[10] Richards, D. (2006) Is Interactivity Actually Important? Proceedings of the Third Australasian Conference on Interactive Entertainment (IE'2006), 4-6 December 2006, Perth.

[11] Richards, D. and Barles, (2005)Actors vs. Animation for Adult Learning In J. Yusuf Pisan (ed.) Proceedings of the Second Australasian Conference on Interactive Entertainment (IE.2005), 23-25 November 2005, Sydney, Australia, 163-166.

[12] Richards, D., Dras, M., Porte, J. and Taylor, M, (2009) The Role, Measurement and Delivery of Immersion in Training Environments, Proceedings SimTecT 2009 Simulation Conference: Simulation - Concepts, Capability and Technology (SimTecT 2009), Adelaide Convention Centre, Adelaide, Australia, 15 - 18 June, 2009.

[13] Silva, Daniel Castro, Silva, Ricardo Reis, Luís Paulo and Oliveira. Eugénio (2009) Agent-Based Aircraft Control Strategies in a Simulated Environment In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009.

[14] Sollenberger, Derek J. and Singh, Munindar P. (2009). Architecture for Affective Social Games In Frank Dignum, Barry Silverman, Jeff Bradshaw, and Willem van Doesburg, editors, Proceedings of First International Workshop on Agents for Games and Simulations, volume 5920 of Lecture Notes in Artificial Intelligence. Springer.

[15] Gemrot, Jakub, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Juraj Šimlovič, Radim Vansa, Michal Štolba, Lukáš Zemčák and Cyril Brom. (2009) Pogamut 3 Can Assist Developers in Building AI for Their Videogame Agents In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009.

[16] Szilas, N. (2007) BEcool: Towards an Author Friendly Behaviour Engine. International Conference on Virtual Storytelling 2007: 102-113

[17] Szilas, N. (2007) A Computational Model of an Intelligent Narrator for Interactive Narratives. Applied Artificial Intelligence 21(8): 753-801.

[18] Westra, Joost, van Hasselt, Hado, Dignum, Frank and Dignum, Virginia. (2009) Adaptive serious games using agent organizations In Proceedings of Workshop on Agents for Games and Simulations in conjunction with Autonomous Agent and Multi-Agent Systems (AAMAS) Budapest, May 11, 2009

[i] http://www.garagegames.com/
[ii] http://nwn.bioware.com/
[iii] http://nwn.bioware.com/about/description.html
[iv] http://www.emergent.net/
[v] http://www.elderscrolls.com/games/oblivion_overview.htm
[vi] http://www.valvesoftware.com/,
[vii] http://www.idsoftware.com/business/history/,
[viii] http://www.garagegames.com/pg/product/view.php?id=1
[ix] http://nwn.bioware.com/builders/
[x] http://www.steampowered.com/?area=news
[xi] http://www.havok.com/
[xii] http://www.unrealtechnology.com/html/technology/ue30.shtml
[xiii] http://www.planetunreal.com/gamebots/downloads.html