

**A TEST BED FOR DEVELOPING
INTELLIGENT SYNTHETIC CHARACTERS**

John E. Laird

Mazin Assanie, Benjamin Bachelor, Nathan Benninghoff, Syed Enam, Bradley Jones,
Alex Kerfoot, Colin Lauver, Brian Magerko, Jeff Sheiman, Devvan Stokes, Scott Wallace

October 22, 2001

University of Michigan

Over the last four years, we have been doing research on incorporating advanced AI into computer games. We have focused on developing enemies for actions games such as Quake that have many of the same capabilities as human players, including the ability to use many tactics, create internal maps of the level, and anticipate their enemy's behavior. Although action games such as Quake are one of the most popular game genres, there are inherent limits in the complexity of behaviors required to create compelling bots that are essentially computerized punching bags. Furthermore, these types of games limit the human gaming experience to violent interactions with other humans and bots. Therefore, we are currently working to develop non-violent plot-driven computer games where we really need complex AI characters. The behavior of these characters cannot be determined by a simple script, but must be driven by the interaction of their body with the environment, their goals, their knowledge of the world they inhabit, their own personal history, their interactions with human players, and real-time advice from a director. Our hope is that complex AI characters will lead to games where the human players are faced with challenges and obstacles that require meaningful interactions with the AI characters. We are building on one of the oldest genres of computer games, sometimes called interactive fiction or adventure games, which involve having the human player overcome obstacles and solve puzzles in pursuit of some goal – games such as Adventure, Zork, Bladerunner, and the Monkey Island series. One weakness of these games is that the behavior of non-player AI characters is scripted; so that the interactions with them are stilted and not compelling. Our challenge will be to create AI characters whose behavior is not only human-like but also leads to engaging game play.

In this paper we describe our test bed for pursuing research in developing human-level AI characters within computer games. To date, every existing computer game is an existence proof that you can create a game without human-level AI. Our challenge is to demonstrate that for at least one genre, human-level AI can make a difference – with human-level AI the game play is qualitatively different (and still entertaining). The paper covers the following: our initial script/game scenario, the design of the physiology and sensing of our characters which in turn forces us to support a combination of goal-driven and environmentally-driven behavior, and then the overall software system design. Others papers submitted to the symposium cover two related projects that involve the development of a director for the game (Magerko 2002), and the development of synthetic characters that can accept direction gracefully (As-Sanie 2002).

OUR STORY: HAUNT 2

Using Unreal Tournament (UT), we are creating an adventure game where the player takes on the persona of a ghost-like energy creature trapped in a house. In our game, the human player's goal as the "ghost" is to escape the house and return home to an underground cavern. The ghost is severely limited in its ability to manipulate the environment. It can move or pick up light objects, such as a match or a piece of paper, but it can't move or manipulate heavy objects. Moreover, contact with metal drains the ghost's energy, so the ghost must avoid metal objects. These constraints force the player to entice, cajole, threaten, or frighten the AI characters into manipulating the objects in the world, which in turn forces us to develop AI characters that have enough "intelligence" to make these social manipulations possible and realistic. Initially we are avoiding issues with natural language understanding because the player as a ghost is unable to generate human language. The user must find ways of manipulating the world to influence the behaviors of the AI characters. To provide even more interaction, the ghost is able to "possess" an AI

character as long as the AI character isn't too scared. Possession allows the player, as the ghost, to see some of the AI characters thought processes, and influence decisions where the AI character does not have a strong bias. However, whenever the ghost influences a decision, it feels a bit weird to the AI character, raising the possessed character's level of anxiety. The ghost can maintain possession of relatively calm characters, so too much manipulation leads to the ghost being expelled from the AI characters. In order to make the possession fun and engaging, we will have to develop characters whose internal processing appears human-like to a player possessing them.

AI CHARACTER PHYSIOLOGY

With the AI characters playing such a central role, they must be well grounded in their environment. For example, there is an evil scientist who is immune to fear but is weak and easily fatigued by exertion or cold and wants to capture the ghost character, and there is also a lost hitchhiker (we aren't trying to have the most original story ever) who is easily frightened by the ghost, but is physically strong and driven by curiosity. The game will push our research to integrate the knowledge-based, goal-oriented reasoning that we have concentrated on in the past, with emotions, personality, and physical drives that have been used in simple, knowledge-lean agents in other systems [a la the Oz project, Jon Gratch's work at ICT, and the Sims].

To support the physical drives, we have extended Unreal Tournament so that all of our characters have a model of physiological responses to the environment and to their internal processing. Moreover, the environment has attributes that influence the physiology of the characters. For example, just these games have a measure of ambient light level, we have added ambient temperature. Different regions of the game have different ambient temperatures; outside it is very cold, inside it is moderately cold; when a fire is lit in the fireplace, it is very warm near the fire. All of the physiological properties serve as input into the AI characters, that is, the character is aware of their values. However, the character can only change them indirectly change them by the actions it performs. For example, the characters have a body temperature that can be raised by exertion, by changing the clothes they wear, or by moving to different regions of the level that have different temperature levels, such as near the fire. Changes in one of these attributes can affect others, so that a significant drop in body temperature can make them more tired.

Physiological effects that we have implemented include: temperature, exertion, fatigue, sleepiness, hunger, and thirst. There are other attributes that impact the character's actions, such as its strength, speed and dexterity. Commercial computer games have had complex physiological effects for the human player's character as well as many character attributes (an important part of role-playing games). For example, the Sims has a set of attributes for the computer characters, which drive their behavior. We are attempting to extend the set to be more comprehensive and to explore the interplay between physiological drives and goal-driven behavior, which The Sims lacks.

ENVIRONMENTAL SENSING AND ACTION

As in our previous work with Quakebots, we are committed to giving our characters realistic sensing and actions in their environments. However, this is challenging because of the difficulty of sensing walls (and doors) in these environments so to start with we are fudging a few things. To start with, we are annotating the map with regions that give the name of each room so that the characters can thus directly sense which room they are in. We are creating navigation nodes in the map that are placed at important locations (doors, windows). The characters will use these nodes for navigation between rooms, but will move more freely within a room based on their sensing of objects and other characters. The characters will move using controls similar to those used by a human player (turning left and right; thrusting forward, backward, left, and right). This is more challenging just moving to nodes or objects, but gives more flexibility in controlling the character during the game and we have been successful using such a scheme in the past.

SYSTEM SOFTWARE DESIGN

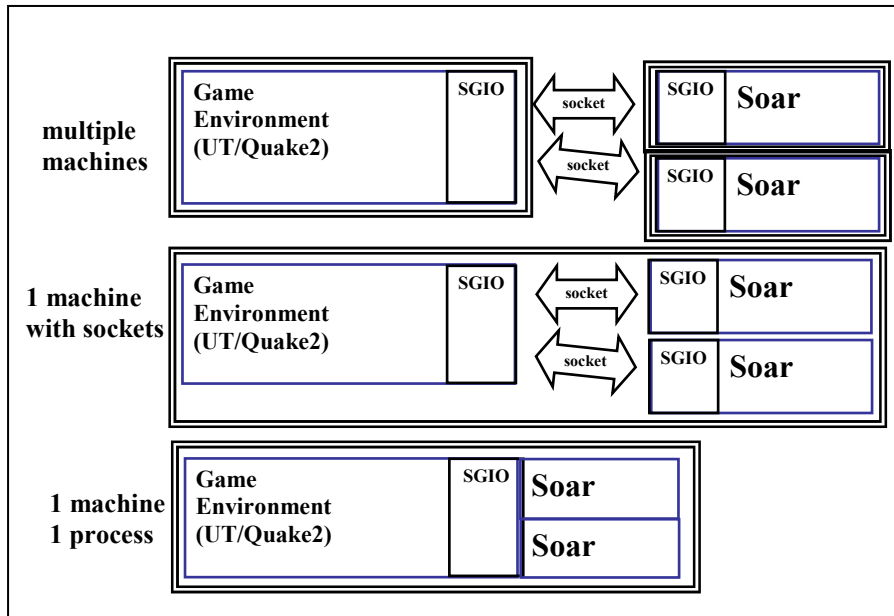
To support the development, experimentation, and evaluation of our AI characters, we need a game environment with the following properties:

1. Flexible and low-cost development:
 - We should use existing game software and development tools (level and character editors) whenever possible.
 - The underlying software should be easily modifiable so that we can test out alternative designs.
2. Debugging/development environment:
 - Our design should allow us to use all of our existing debugging and development tools for creating AI characters.
3. High performance:
 - We should not sacrifice performance in the underlying AI engine or in its interface to the game.
 - We should be able to support a large number of AI characters in the game without sacrificing graphics performance.
 - We should be able to run the game and the AI engine on a single laptop.

In order to meet many aspects of the first goal, we have followed in the footsteps of other projects (NCSU: Mimesis, ESC Online, Deus Ex, Gamebots) by using the Unreal Tournament (UT) engine. UT provides an off-the-self, high-quality 3D game engine that can be easily extended (Deus Ex was voted as best action game of 2000 by PC Gamer Magazine and it is built on UT.) A copy Unreal Tournament costs about ~\$20. Moreover there are many free level editors available for creating your own virtual environment and all of the game physics and interface is coded in a powerful internal scripting language (UnrealScript) that is completely accessible.

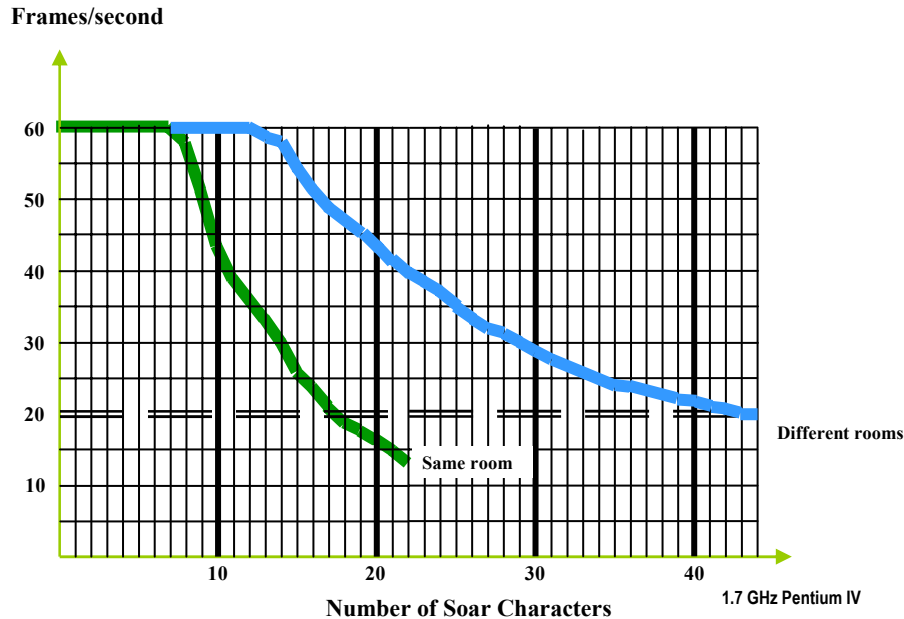
Once the game engine/environment is selected, the next critical design choice is how to interface the environment to the AI engine. The final three goals are often contradictory – having a very high performance interface demands that the AI engine run as an embedded application in the same process as the game environment. That forces us to sacrifice our development environment. If we insist on using our complete development environment, that impacts computational performance and in some cases makes it necessary to run the AI characters on separate machines. In attempt to finesse these problems, we developed a high-level interface that is really three low-level interfaces. The high-level interface hides the differences at the low-level so that and where the user can select from among the three low-level interfaces at run time without requiring any changes to the game engine and the AI engine.

The overall interface is called the Soar General Input/Output (SGIO) and is a domain independent interface between Soar and an external environment – in our case the two commercial computer games, Quake 2 and Unreal Tournament.



The three low-level interfaces that SGIO supports are shown above and are as follows:

1. A socket-based interface across separate computers. The AI system runs on one computer, while the game engine runs on the other. One part of SGIO is embedded in a dynamically linked library (DLL) that communicates directly with the game and uses a socket to communicate with the AI system on the other machine. This arrangement makes it possible to run the full Soar debugging environment on one machine and the game in full-screen mode on the other machine. There is some overhead in SGIO to send information over the socket, and it does add some network latency for the communication to the AI engine, but it does not have a noticeable impact.
2. A socket-based interface between multiple processes on the same computer. In this case, both the game and the AI system are run in separate windows on the same machine. This has the advantage of still allowing the complete debugging and development environment. The disadvantages relative to the first option are that the game cannot run in full screen and that the socket interface and the AI engine are using up memory and processor resources. Neither of these is a significant problem during development.
3. A C-based interface between code running in the same process. In this configuration, Soar is completely embedded within the UT process. There is a low-level C interface that the DLL uses to call Soar functions, and this is very efficient. The disadvantage of this configuration relative to the earlier one is that Soar cannot display any information on the screen during runtime – it is completely embedded. However, this approach eliminates the overhead of the socket interface as well as the overhead of the Soar debugging environment. The figure below shows the performance of UT with Soar using this approach as we spawn more and more characters in a game. This was run on a 1.7MHz Pentium IV that had a GeForce 2 graphics card. The y-axis is the frame rate of the game – that is, it is the number of times a second that the game loop is executed. This includes executing every AI character and drawing the graphics. The graphics system we were using had a maximum frame rate of 60 Hz, and this is maintained until a significant number of AI characters are created. Each AI character is a small Soar agent doing only a minimal amount of processing. The left-most curve shows the case where the characters are all in the same room, while the right-most curve shows the case where the characters are in different rooms. The reason for the poorer performance in the left curve is that the sensing calculations are being done for all of the characters in view of each other.



Although SGIO was designed for Soar, many aspects of it, including the communication language are independent of Soar. SGIO supports the creation, deletion, and modification of graph structures represented as attribute-value triples as well as some simple meta-commands, such as stopping and starting the simulation.

CONCLUSION

In this project, we are creating an environment in which we can do research and development of new computer games where human-level AI plays a critical role. To be successful, we need not only develop human-level AI characters, but also demonstrate that they are critical for creating a compelling game. These intertwined goals make the project both exciting and challenging. A critical part of our success rests on developing the infrastructure to support the research. Some of the infrastructure we borrowed from our earlier work on Quake 2 and all of it is informed by those experiences. One of the biggest lessons we have learned is to create modules with well-defined interfaces so that we can reuse our software. Quake 2 and Unreal Tournament are themselves excellent examples of this. We encourage our colleagues to work with us to define additional modules that can be shared among our community. For example, it may be possible to create a generalized version of SGIO that can be used with non-Soar architectures. Furthermore, we should consider creating common models of physiology and sensing that we can share among research groups. This will not only speed our own research, but it will encourage others to participate in research in AI and games.

REFERENCES

Will fill in for final version.