Creating and Visualising an Intelligent NPC using Game Engines and AI Tools

N.P. Davies, Q.H. Mehdi and N. Gough

Research Institute for Advanced Technologies
School of Computing and Information Technology
University of Wolverhampton
Wolverhampton, UK
E-mail: N.P.Davies2@wlv.ac.uk

ABSTRACT

Our research is focussed on the creation of human-like Artificial Intelligence (AI) based on the Belief Desire Intention (BDI) paradigm that employs rational decision making processes to achieve behaviour in computer game characters that will appear more realistic than current reactive techniques. To test the agent behaviour we need to develop an architecture that incorporates various systems such as an advanced graphics engine, AI environments, and GameBots.

This paper presents a framework that would help solving problems such as predictable and repetitive behaviour. The architecture and usefulness of this system will also be detailed here.

INTRODUCTION

Computer games developers have traditionally devoted the majority of time and processing power to high performance graphics, as this is the most appealing feature to a potential game player (Namee. 2004). However, this development strategy is producing diminishing returns as modern computer games are now capable of near photo-realistic graphics (Valve Software. 2004). In comparison, AI has been neglected as a development priority, and has evolved to work within the confines of limited processor allocation. This has resulted in AI based on efficient reactive techniques rather than complex reasoning systems. Agents do not 'think' (reason) about actions, but rather perform a set response to a set event. In some cases this has resulted in poor AI, where characters can either appear to be 'stupid' by not being able to react to unpredicted human interaction, or 'cheat' when the developer supplies extra information and abilities to the agents unavailable to human players. Some techniques have been developed that can successfully approximate human-like intelligence very efficiently and effectively (Woodcock 2000), but even these techniques can produce repetitive behaviour as agents do not have the ability to adapt. With graphics now maturing, developers are starting to release more processing power for the inclusion of sophisticated AI that will overcome some of these problems.

The architecture we intend to implement will based on a simplified view of human cognition known as Belief, Desire, Intention (BDI) as postulated by Bratman (1987). The architecture is capable of modelling expert human behaviour, and has been used in applications such as Air Traffic Control (Rao & Geofgeff,. 1995), and Military Simulations (Wallis et al. 2002), where agents are endowed with military tactics and used as opponents in air combat training. More recently, the architecture is gaining interest from the computer game industry with some success (Norling. 2004). By adopting a goal-based approach to reasoning based on human-like decision-making processes, agents will assess and react to high-level goals in a natural way. This will introduce variability to agents' behaviours, as they will be able to react to an environmental change in many different ways, which do not have to be explicitly specified by designers. There are still many unexplored research areas, including the role of memory and emotion on decision-making processes, the effect of physical conditions of an agent e.g. fatigue, and the aspect of team coordination with social hierarchies where tasks need to be distributed.

GAME ENGINE DEVELOPMENT

The integration of high performance graphics and human like AI for computer games and simulation would be very useful for visualising agent behaviour. A system for the creation and rendering of virtual scenes and graphical environments was developed by Davies et al (2004). The initial system was used for the creation of quasi-accurate 3D scenes that incorporated animated characters. The implementation used a custom graphics engine developed in DirectX, graphics developed in 3D Studio Max, and a tool for building maps via a 2D plan in C#. It was possible to create some impressive results in a relatively short space of time and investment. Our implementation loads and orientates 3D models and renders them to screen. Animated characters are also incorporated running pre-created animation sequences. Users navigate around the scene using mouse and keyboard input to view the action from difference locations and angles. However, it becomes increasingly difficult to make significant progress when requirements become more sophisticated, and our graphics engine started to exhibit limitations quite rapidly. The refresh rates for visualisations start to fall below 30fps when more than three characters are present resulting in jerky animations. The engine does not support dynamic

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

animation or many other important features. It is possible to incorporate these features, however, significant allocations of time and resources would be required. Even the basic engine we developed is still quite complex. According to Lewis and Jacobson (2002), the development of highly accurate graphics and virtual reality systems is now mainly the domain of generously funded research establishments and military instillations with six figure budgets. Therefore, we have investigated alternatives to building our own custom graphics engine.

With the development of modern computer game engine technology in the commercial sector, it is now possible to create impressive results without having to expend resources developing custom graphics engines. Game companies with decades of development experience, and access to some of the most innovative developers and substantial investments in time and money are creating some of the most sophisticated simulation environments available. Prior to game engines, content and code were integrated into a single system. To modify a game, changes had to be made directly to the source code and the game recompiled, which made it virtually impossible for anyone but the game developers to use the game technology for anything other than its original purpose. The original DOOM™ (ID Software) game broke this mould and created the first modular game engine. This technique splits the technical aspects of the graphics rendering pipeline, which is the domain of a small group of specialist developers, from the game specific aspects which can be left to teams of content writers and game developers. This results in the 'game' being considered the graphic models, animation sets, sound, AI and Physics, and enables specialists to concentrate on their particular area of expertise. To allow this process, developers invest heavily in the creation of tools and scripting languages that content developers use to create the game. A recent trend is for developers to release their development tools to the gaming community, allowing the game to be modified after it has been released. This can extend the shelf life of games, with a huge amount of new content being provided on a regular basis by external sources. These modifications range from the creation of new graphics to complete games types (LudoCraft, 2005). It is possible for these tools to be used by the research community which allows access to sophisticated game engine features inexpensively. The use of game engines is not a complete solution to our problems, but it is adequate for the research we are undertaking. While extremely sophisticated, game engines are not real world simulators. We have to work within the limits imposed by the game engine developers. For example, if we wanted a feature such as Kinematics animation that the engine does not provide it would be difficult, but not impossible, for us to incorporate it. However, even though there are limitations, the benefits and functionality far out way negatives.

We intend to use the game engine Unreal Tournament (Epic Games), which provides a high performance real time 3D graphics platform. The game is extendable via a suit of tools provided by the developer, including a C like language Unreal Script which can be used to customise physics and behaviours, and a graphical map builder that is used design game levels. The game consists of 'maps' which are created by level designers and specify the location of graphics, objects, and a network of nodes or waypoints. The internal game characters play as team mates or opponents to human players and are known as 'robots', which are abbreviated to 'bots'. The bots use the map waypoints embedded in the game maps to navigate around the game. The original game contains multiple game types including team based and individual scenarios, and allows human-human, agent-agent, and human-agent teams as shown in Figure 1. The main team based game is called 'Capture the Flag', where teams on two competing sides work together to capture the oppositions flag and return it to their base whilst also protecting their own flag. Other game types include DeathMatch, where the goal is to kill as many opponents as possible.

The game is fundamentally violent in nature, however, it is possible to customise the game in such a way to minimise the violent aspects via the scripting language replacement graphics. There has been substantial interest in the academic community in the use of Unreal Tournament in research including Sioutis et al (2003) which has arisen partly because of the extension GameBots/JavaBots (Marshall et al., 2004) technology shown in Figure 1, which is an extension to the Unreal Engine written in Unreal Script that sits between the game engine and an external client that receives and distributes messages via a TCP/IP link. This has the benefit that AI can be written in any language or development environment that allows a network connection.

Using this information sent out by the game, we need to implement a suitable AI system. It is relatively straightforward to implement algorithms such as the path-planning algorithm A* in the game using waypoint positions within the game map, and to script more complex behaviours in set pieces. The current game AI is quite standard, for example, in Death Match games, bots work on a finite state machine reasoning with states such as roam, flee, and engage enemy. Different skill levels are achieved by increasing the bots speed, accuracy and field of view. The AI is developed in this way for a number of reasons. Games are processor intensive. Clock cycles are at a premium when other factors such as physics and graphics are also competing for processor time. As far as the game developer is concerned, as long as the behaviour appears variable and reasoned, it is irrelevant how it is achieved. Performance and optimisation are the dominant factor. Another issue is the fact that for many computer games human level sophistication is simply not required. The

game domains are relatively discreet requiring only a limited array of behaviours. We intend to create more sophisticated behaviour in more complex game types, and to produce a more robust cognitive architecture. This will require us to model human like behaviour by capturing expert knowledge for a particular game type, and convert this information into a knowledge base that an AI agent can utilise. However, the information received will determine the AI that can be produced. Similar work was carried out by Norling (2002) using the Quake engine. A number of limitations were reported by them where certain behaviours cannot be modelled due to the deficiencies in information sent from the game server. Specifically this related to the behaviour of throwing a grenade at an archway so it would bounce off and hit a chasing bot. As the game engine did not send information identifying when an agent was under an archway, this could not be modelled. With the use of Unreal, which is more customisable, we may be able to address these deficiencies. In effect we have an environment that is partially observable where the agent cannot see everything that is happening everywhere in the environment, and dynamic, the environment is constantly changing. There are many factors happening simultaneously, which cannot be anticipated. Actions are non-deterministic, and it may be that a goal will not be achieved even though an agent is committed to achieving it. For example, an agent may decide to increase its health by searching for a health pack. It cannot guarantee that this action will be successful, and other events may occur that intervene. On route to collecting the health pack, the agent may sustain further damage and die. The AI engine needs to account for this using information sent from the game server, and adapt its behaviour accordingly. The agent will have to build an internal representation of the environment, plan a course of actions which will be limited to available commands, execute the plan, and observe the results. This scenario is getting close to a real-life open system used in robotics and its associated technologies, and would support research into many areas including reactive systems, deliberative systems, cognition, memory and learning, planning etc.

The Unreal Tournament and GameBot system specifies messages into two types; server messages and client messages. Messages from the server provide sensory information detailing what a bot can 'see' in the game, as well as information about the bots physical status e.g. health, armour etc. Other information is also sent, such as if the agent has taken damage, and the direction a shot came from. Messages from the client take the form of commands that instruct the agent to perform tasks such as rotate, walk, and shoot. The GameBots system allows the client to have access to the internal native AI system, and allows instructions such as GETPATH which returns a list of waypoints between two locations, and RUNTO that will direct the bot to a specified waypoint. Another benefit of the GameBots / JavaBots system is that it allows for an extendable gaming environment.

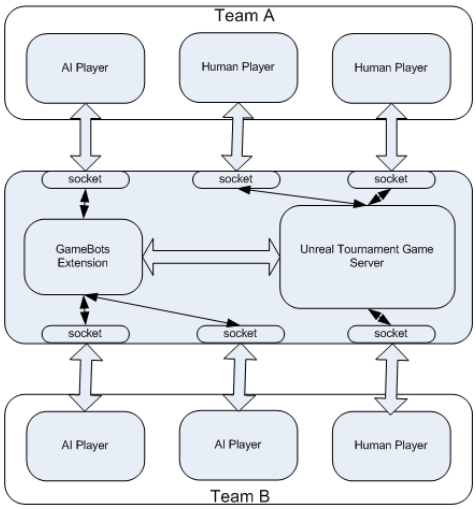

Figure 1: GameBot system

BDI AI ARCHITECTURE

To develop an AI agent that will have the ability to recognise situations, formulate plans based on the current situation, and execute those plans, while executing plans, the agent should be able to monitor the environment in order to ensure the plan is still relevant. The agent has to be embodied with the ability to adapt and change plans in order to add variability to behaviour, which will make the agent appear less predictable. In addition to plan modelling, a model for temporal characteristics that influence the decision-making process such as emotional, social and physical characteristics will be created. This includes elements of team working, where agents will collaborate with each other by allocating tasks. Finally, the application needs to execute plans in a timely manner on a resource bound machine.

The AI architecture identified as suitable candidate that exhibits some of these features is the Belief-Desire-Intention architecture (BDI) as originally postulated by Bratman (1987), shown in Figure 2. In this architecture, an agent is characterised by its beliefs about the state of its environment, goals (desires) that it wishes to achieve in this environment, and a set of plans and partial plans it can use in order to satisfy its desires. As the model for the AI agent needs to be executed on a machine with limited resources, the concept of intention is introduced which limits the deliberation an agent is required to perform. This information will be limited to information regarding tasks the agent has committed itself to achieving. An intention is formed when the agent commits to a particular goal, and retrieves a plan from the plan library that

contains a particular sequence of steps to perform in order to achieve a goal. The steps themselves may be atomic actions, or they may be sub goals, which can be satisfied by other plans. The beliefs, goals and intentions of an agent are maintained by the BDI reasoning engine. This engine will help to drive the agent, update beliefs, monitor and update goals, form intentions, and select plans to achieve goals.

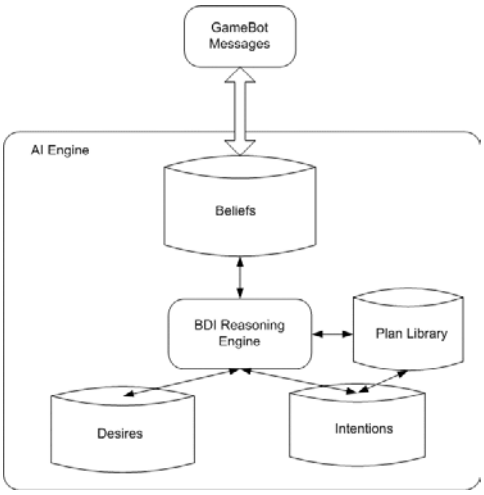

Figure 2: BDI Architecture (Bratman 1987)

The plan library is an important feature of this architecture that specifies how goals can be achieved. Although the plans are fixed sequence of actions, they do not have to be fully specified. For any particular goal, there may be multiple plans to achieve that goal, and while any plan may be fully specified as a sequence of actions, a plan may instead consist of a sequence of sub goals or a combination of actions and sub goals. All plans will eventually decompose into a sequence of atomic events, which will take the form of GameBot messages. If a plan contains sub goals, reasoning to decompose the plan into atomic events can be delayed until the behaviour is required. For example, in the Unreal Game, a plan may be to kill opponent agents. This can be broken into sub-goals of 'locate weapon', 'locate enemy' and finally 'kill enemy'. With this high level plan in place, the agent narrow deliberate to achieve the first stage of the plan, which may involve a path planning or roaming behaviour. Once this part of the plan is completed, the next stage will become relevant, and deliberation commenced upon that stage. While this does not achieve the full range of adaptability that humans display when reasoning about their goals, it does allow considerable flexibility in the agent's planning, and its resulting behaviours, and also allows a balance between reactive and deliberative planning.

In theory, the BDI agents will use a utility-based decision making strategy, where decisions are made to maximize the expected utility of the selected course of action. However, in practice on resource bound machines this is either impractical or unrealistic. In the some systems the default behaviour is simply to select the first applicable plan, regardless of expected utility. In real-world situations, while either of these strategies may be used on occasion, it is argued that neither is commonly used, particularly when the person is operating within their area of expertise (Norling 2003). The DBI paradigm does not incorporate all the required features we are intending to incorporate into our AI, such as physical attributes including emotional states, memory, and learning. However, there is much research ongoing into incorporating these aspects.

BDI TOOLS

The development of a BDI architecture is not a trivial task, and much as the development of commercial game engines can be used by researchers, there are many mature systems based on BDI available ranging from open source to commercial applications. Examples of open source projects are the Java Agent Development Environment (JADE) (TiLab 2005) with the BDI extension JADEX (Braubach & Pokahr. 2005). JADE is a mature platform that is FIPA (Foundation for Intelligent Physical Agents) compliant. FIPA are an organisation that is producing standards for software agent development. The JADE environment is based on the Java environment, and allows the creation of agents via a GUI interface. JADEX is a BDI add-on to JADE that is currently in development. The use of JADEX may be a viable option to base our AI on in the future, however, as development is currently at a BETA stage it was considered JADEX is still not mature enough to consider. Another example of an open source BDI project is Jason, which is an interpreter for the agent oriented BDI programming language AgentSpeak. The environment is also constructed over in the Java environment, and allows the development of agents which can be distributed over a network connection. An alternative to pure BDI environments is the cognitive AI architecture SOAR (University of Michigan) which is a mature environment that has been in development for over 20 years. It attempts to provide a general cognitive architecture for developing agents that can exhibit human-like intelligence. It addresses factors such as knowledge complexity and rational decisions making within these complex domains. An advantage in using this technology is that it has started to address factors such as memory, learning and emotion. The platform has been used in a diverse set of academic applications including training simulations and robotic control. The environment is not platform independent like the Java solutions, however, there are environment available to the major operating systems. However, the platform we have chosen to implement our AI on is the leading edge commercial BDI development environment JACK (Agent Oriented Software). This system provides a comprehensive toolkit, and extensive documentation. Like other system outlined above it is designed as an extension

to the Java language, which makes it platform independent. Agents can be developed via a GUI application, or by the use of keyword extensions to Java. Another aspect that makes JACK favourable is the maturity of the system for use in computer games. Work has already been completed by (Norling) into integrating JACK and Quake, and more recently, (Soitis) have developed an interface between JACK and GameBots, which may be released as open source in the future. There is also interest in developing an extension to JACK called Co-Jack, which is attempting to incorporate moderating factors such as emotion and physical conditions such as fatigue, and has been initiated by the Ministry of Defence (Lucas. 2003.)

THE PROPOSED ARCHITECTURE

Figure 3 shows the proposed architecture which is based on the work carried out by Sioutis et al. 2003. The architecture uses the JACK BDI programming language and links to the Unreal Tournament game engine via the use of GameBots/JavaBot technology. This allows a structured environment in which to investigate the development of human like artificial intelligence. The architecture is divided into two main sections, with a third section linking the two together. The section to the left of the dotted line represents the commercial game engine, Unreal Tournament, where a custom game can be defined via databases containing custom graphics, and a second database containing game modifications and rules written in Unreal Script. To the right of the dotted line is the BDI reasoning system JACK which receives messages from the game server and uses it to update a Belief data set, describing the current world state. Based on the beliefs, a set of goals and plans will be formulated from a plan library which at the lowest level will take the form of GameBots commands such as Rotate and Walk. The middle linking section contains the GameBots technology that distributes messages and receives commands over a TCP/IP network connection. The architecture is not limited to the use of Unreal Tournament. If an alternative game were used, only the middle link layer would need to be re-written to account for an alternative set of messages. The GameBot/UT link will then be developed and incorporated. This part of this system has been developed by Sioutis et al (2003), which we may adopt in the future.

APPLICATION EXAMPLE

To begin the development of our artificial intelligence we first need to specify and design a game type where human like reasoning would be of benefit. It has been demonstrated that games such as a FPS Death Match can be developed with simple rules and function satisfactorily. Games that would benefit from human like AI will be of a slower pace, and give the game player time to perceive and interact with the intelligence. Games that will exhibit this

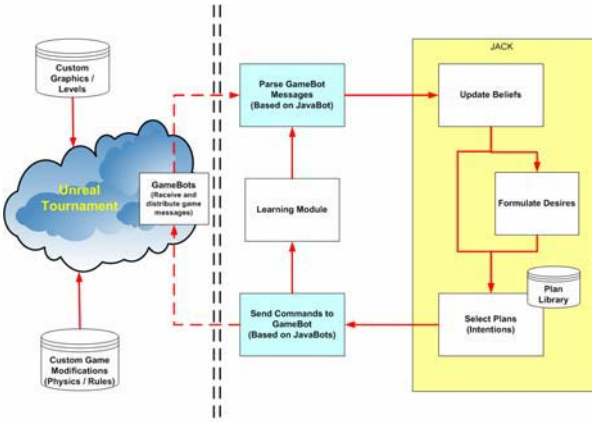

Figure 3 : Unreal Tournament / GameBots / JACK architecture

type of behaviour would include, but not limited to, tactical combat games such as hostage situations or squad based combat missions. Unreal Tournament lends itself to this type of game, and a modification would be feasible to produce. Once the game has been specified, a knowledge base will be developed outlining the tactics that can be employed by the agent. This task will account for a significant amount of development time.

Figure 4: PacMan style game with integrated BDI controlled NPCs

As an initial proof of concept we have developed a simple PacMan game shown in Figure 4, which is linked to JACK through the glue language TCL. PacMan was chosen as it shares many of the features that are common in FPS games such as Unreal Tournament. Agents are required to navigate around an environment based on waypoint nodes, and are blocked for navigating certain paths due to obstructions such as walls. Agents can also pick up key objects such as pills which can change other agents' states. Agents can incorporate team tactics to capture an enemy, and an agent can evade capture from the teams of agents. The game is open and dynamic, and much of the knowledge base will be applicable to further developments using a more sophisticated games engine.

CONCLUSION AND FURTHER WORK

In this paper, the benefits and limitations of developing human like artificial intelligence and high performance graphics engines have been outlined. The benefit of using commercial game engines including Unreal Tournament and its associated tools is to allow the designer to concentrate on AI modelling and simulation rather than devoting time for developing a custom graphics engine that may not be as sophisticated as a commercial engine. The limitations of using commercial game engines have also been identified; however, these limitations have not had a significant effect on the development and implementation of the system. The BDI architecture as proves to be a suitable system paradigm for the requirements of our AI system.

In conclusion a system that includes the use of JACK, GameBot/JavaBot and Unreal Tournament offers a good development platform for testing AI agents in virtual worlds.

Further work will include the development of GameBot / JACK interface that will allow sophisticated techniques such as emotion and physical limitations to be incorporated into the system.

REFERENCES

Agent Oriented Software Pty. Ltd. JACK Intelligent Agents. http://www.agent-software.com

Bratman, M. (1987) "Intentions, plans and practical reasoning." Harvard University Press, Cambridge, Massachusets.

Braubach, L., Pokahr, A., 'JADEX' http://vsis-www.informatik.uni-hamburg.de/projects/jadex/ Accessed 10.03.2005

Davies, N., Mehdi, Q., Gough, N. E. (2004), 'Crime Scene Reconstruction With Integrated Animated Characters', Proceeding of European Simulation Multiconference. Networked Simulation and Simulated Networks, Magdeburg, Germany, 388-393

Epic Games Inc. Unreal Tournament. http://www.unreal.com

Lewis, M., Jacobson, J., (2002), 'Game Engines in Scientific Research', Communications of the ACM, Volume 45, Issue 1. New York, USA

Lucas A.,, Human Variability in Computer Generated Forces, presented at Behavior Representation in Modeling and Simulation (BRIMS'03), 2003.

LudoCraft Ltd, AirBuccaneers. http://ludocraft.oulu.fi/airbuccaneers/ accessed 10.03.2005

Magerko, B., Laird, J., Assanie, M., Kerfoot, A., Stokes, D., (2004) 'AI Characters and Directors for Interactive Computer Games' AAAI Press; Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference, San Jose, CA

Marshall, A. N., Gamard, S., Kaminka, G. J., Manojlovich, Tejada S., Gamebots, viewed 10th Mar 2005, http://planetunreal.com/gamebots/

McBreen, H., Shafe, P. Jack, M., Wyard, P. (2000) "Experimental assessment of the effectiveness of synthetic personae for multi-modal E-Retail applications" in: Proc. 4th Int. Conf. on Autonomous Agents (2000), pp. 39-45, Barcelona, Spain

Namee. M. B., (2004), Proactive Persistent Agents: "Using situational intelligence to create support characters in character-centric computer games". PhD These, University of Dublin, Trinity College

Norling, E. (2004) "Folk psychology for human modelling: extending the BDI paradigm". In : Int. Conf. on Autonomous Agents and Multi Agent Systems (AAMAS), New York, 2004.

Rao, A., Georgeff, M., (1995) "BDI agents: from theory to practice" Tech. Rep. 56, Autralian Artificial Intelligence Institute, Melbourne, Australia

Sioutis, C, Ichalkaranje, N & Jain, LC 2003, 'A framework for interfacing BDI Agents to a real-time simulated environment', Design and Application of Hybrid Intelligent Systems, A Abraham, M Koppen & K Franke (eds), proceedings of the 3rd International Conference of Hybrid Intelligent Systems (HIS'03), Melbourne, Australia, December 2003, IOS-Press, Amsterdam, The Netherlands, pp. 743-748.

TiLab, Java Agent Development Environment (JADE) http://jade.tilab.com/ accessed 10/3/2005

University of Michigan 'SOAR' http://sitemaker.umich.edu/soar Accessed 10.03.05

Urlings, P., Tweedale, J., Sioutis, C., Ichalkaranje, N., (2003) "Intelligent agents as cognitive team members" in: Proc. 10th Int. Conf. On Human Computer Interaction, Crete-Greece, June 2003

Valve Software (2004) Half Life 2, http://www.half-life.com/

Wallis, P., Ronnquist, R., Jarvis, D., Lucas, A., (2002) "The automated wingman – using JACK intelligent agents for unmanned autonomous vehicles", in IEEE Aerospace Conference, Big Sky MT

Woodcock, S., (2000) "Flocking: a simple technique for simulating group behaviour" in Game Programming Gems, Charles River Media