Planning and Research
CI301 - The Individual Project

Thomas Taylor
Student Number: 08813043
Supervisor: Graham Winstanley

November 2011

1 Project Aims and Objectives

Aim The primary objective of my project is to investigate the feasibility and effectiveness of incorporating academic machine learning techniques into computer games.

Objectives I intend to do this by developing an AI system which is capable of controlling a number of agents to safely navigate a game environment.

The AI system should be able to:

- Analyse the environment
- Ascertain whether objects are beneficial or dangerous
- Develop a knowledge-base dynamically based on observations made whilst navigating the environment
- Apply this knowledge-base in order to traverse the world

2 Problem Domain

According to John McCarthy, the computer scientist who first coined the term in the 1950's, artificial intelligence is "the science and engineering of making intelligent machines"[4]. An important aspect to creating an 'intelligent machine' is the ability to learn from experience, and modify behaviour accordingly when faced with similar scenarios in the future. This is a problem which has been extensively researched, and techniques have been developed to tackle a variety of learning problems. In fact, machine learning techniques are already in use in a number of commercial applications such as speech recognition, robotic control and machine vision, proving that it can be a very useful solution, especially in solving problems which may have unexpected outcomes that cannot be predicted by the software developer.

There has been substantial research into using machine learning in First-Person Shooter (FPS), Real-Time Strategy (RTS), and more traditional board games. CBRetaliate, a case-based reinforcement learning (RL) system has been developed at Lehigh University Pennsylvania, which utilises case-based reasoning in a team-based FPS game to allow the system to react much quicker than if it were merely using RL[2]. Similarly, NeuroEvolving Robotic Operatives (NERO), a system which uses neuroevolution (i.e. the use of genetic algorithms to evolve a neural network) in real-time to train and battle robotic armies was developed by The Digital Media Collaboratory at the University of Texas[1].

Despite this promising research, the AI systems used in even the most cutting-edge commercial games is far removed from that of academic AI research. Rather, the AI we see in computer games focuses on trying to fool the player into believing the system is more intelligent than it is, using 'smoke and mirrors' and cheap tricks rather than proven academic AI techniques. The reality is that games rarely venture beyond pathfinding algorithms when it comes to using academic AI, with their apparent 'intelligence' usually having been pre-scripted. Many games are even programmed to be intentionally unintelligent (as described in 'Artificial Stupidity: The Art of Intentional Mistakes' [3]) in order for the game not to be deemed too 'challenging'. This often means that "a large part of the gameplay in many games [is] figuring out what the AI is programmed to do and learning to defeat it"[5]. While using basic AI is often an intentional 'design feature' in games, historically AI processing has been given much lower priority than the 3D graphics for example, meaning that highly complex AI systems were just not computationally possible. However, in an age where computing power is advancing at such a rapid pace [Moore's law], and the apparent plateau in the more processor intensive processes such as graphics, we are starting to see more focus and development into game AI. Indeed, in a generation where computer games are incredibly complex and engaging forms of entertainment with increasingly complex worlds, we need AI that can compliment this; gamers will no longer accept sub-par artificial intelligence.

That being said, there are a few examples where AI has been used in commercial games quite effectively in less 'out-of-the-box' scenarios than merely creating highly efficient teamwork-capable enemies in shooting games. One of these is Lionhead Studios' 2001 'god' game Black and White. In it, the player is given a pet 'Creature' to do their bidding, which has the capacity to learn from the player's actions, as well as the actions of the various computer-controlled characters in the game (including other Creatures). For example, the creature learns what objects are suitable to eat based on taste, and what it sees others eating. This can mean that the Creature learns to eat the player's followers, which may require the player to punish the Creature for doing so. This learning is achieved with a combination of decision trees and neural networks, and uses a modified version of Ross Quinlan's decision tree generation algorithm (ID3) to do so.

3 Solution

For the main deliverable of my project, I have chosen to create a game similar to the popular puzzle game Lemmings, and replace the human control element with my AI system. Originally released in 1991 for the PC and Commodore Amiga, the game has a very simple premise: to guide a group of computer-controlled 'lemmings' across a level from the entrance-point to the exit. The lemmings themselves, although computer controlled, have no AI to speak of, and merely walk in one direction until they reach an immovable object (such as a wall) or a trap (water, spikes, big drops etc.), the latter resulting in the demise of the lemming. Each level requires that a certain number of lemmings reach the exit in order for the player to progress. I felt that it would be an interesting challenge to see if I could design a system which was capable of guiding a group of AI characters across a game environment safely, My project can be separated into two distinct parts/deliverables: the game component, and the AI system controlling the agents' behaviour.

3.1 The Game Component

The game component of my project is essentially a stripped-down version of the game with limited tools given to the lemmings (if any at all), and a very limited number of levels. It should be able to work standalone (without the AI system) and would function similarly to the Lemmings game; the characters should enter the level and simply walk continuously until they are killed, or reach an immovable object, after which they return the way they came.

The game itself will consist of a simple game environment viewed side-on, with movement along two planes. The environment will essentially be a number of platforms of varying heights and sizes. In addition to the environment, the game will need a variety of obstacles to hinder the progress of the lemmings. The obstacles that I plan to include in my game will likely be pits, spikes, large rocks, and some form of water/lava. The main goal with the type and number of obstacles in the game is to make sure that there is a suitable complexity/number of items for the system to have to process and have to reason about.

I will obviously also need the lemming characters themselves, which will need some basic animation, as well as some very basic 'AI' to allow them to move across the level. The final thing needed for my game world will be the interface. For the sake of simplicity, I am choosing to remove most (if not, all) of the tools which are given to the player. Instead, I intend to add some sort of interface to the game which will allow the player to tweak various parameters in the AI system. For example, the curiosity of the lemmings, or the level of reward given for certain tasks. This will basically act as a way to test the system in real-time without having to edit scripts or the code directly, and allow for easier (and more robust) user testing of my system.

I will be using a game engine to take care of much of the game-related functionality such as model loading, lighting and physics.

3.2 The AI System

The AI system itself will form the bulk of my work, and will operate behind the scenes of the game component to implement the machine learning techniques I intend to explore in this project. I intend to implement two (or possibly more) alternative machine learning techniques. To begin with, I plan on implementing reinforcement learning and decision trees, as I feel that they are most applicable to the context of this project. However, time permitting, I would also like to implement either neural networks or genetic algorithms.

In order for my project to be a success, my AI system needs to have certain basic functionality. To begin with, the system needs to be able to recognise when an agent has come into contact with any interactive objects, such as walls, spikes, large drops etc. My system then needs to be able to deduce what kind of object it is from its attributes and by querying its existing knowledge-base for any similar obstacles it may have already encountered. My system then needs to be able to make an informed decision as to the best decision for the agent to make, and make it. It will then need to store the resulting outcome, be it positive or negative for later use. At the end of the current game, my system should also be able to evaluate performance using some kind of algorithm to measure the comparative efficiency of each learning technique used. A simple way to do this could be to sum up the number of lemmings which were killed in the process, and measure the time taken to complete the level, with the best techniques obviously killing the least lemmings whilst taking the shortest time. A more complex (and more accurate) algorithm could also take into account other variables such as the number of lemmings that reached the exit, the number of tools used etc, or even analyse the performance of individual lemmings.

There are a number of issues regarding the functionality and implementation of my system which I will need to address during the design phase. One such issue is whether I will use a separate knowledge-base per lemming, or use a shared knowledge-base for all lemmings. The latter will obviously result in a system which is much more efficient at learning, but is perhaps less realistic. Using a separate knowledge-base per agent would no doubt allow for much more interesting analysis at the end of each game. Something else which I will need to address is how I manage the agents' interaction with the environment; whether I will carry out automatic checks to update the system, or use an event-based system to notify the AI of any agent interaction. A final question I will need to address is that of user input; exactly how much the user is able to change, and exactly what parameters will be most useful for the user to be able to manipulate during runtime. It may be interesting for example, to be able to adjust the 'curiosity' of the lemmings (i.e. the likelihood that they will explore unknown paths/obstacles).

In addition to this basic functionality, there are also a number of other features that I would like to implement (time permitting). Firstly, I would like to have two modes in the game: one where the lemming characters respawn at the entrance when they are killed, and one where the lemmings are unable to respawn, and the game ends when all lemmings have been killed. The latter could provide some interesting results, as the user must develop a detailed enough knowledge base with a limited number of agents. Finally, I would also like to implement the ability for agents to learn by 'watching' other agents interacting with the environment, in addition to learning from first-hand experience.

As a final consideration, my system needs to be able to perform all of the above for multiple agents in real-time, so performance will need to be a serious consideration so as not to affect the gameplay or frame-rate, and my system will need to be appropriately designed with this in mind.

3.3 Software and Frameworks

In this section, I will briefly discuss the development of my project with regard to the software and frameworks that I intend to use, as well as my reasoning behind them. The first choice I was faced with was the language that I would use to program my system in. As a significant chunk of my project is a game, the obvious choice would be to use C++, as it offers the best performance, and has a lot of useful memory management features. I eventually decided to use Objective C for my project, as I wanted to set myself the challenge of learning a new programming language. Objective C also has many benefits including most of the performance and memory management features of C++ (both languages being C based); performance obviously being a serious consideration when developing a game. Another benefit to using Objective C is having the option to use Apple's Xcode as a development environment, as well as the incredibly useful performance profiling program Instruments to help catch any memory issues. Xcode also features some nice source control integration. Another benefit to using Objective C and Xcode is the ability to develop my project as an iOS application to work on iPhone and iPad devices.

As I am essentially creating a game for part of my project, I would need to be able to load character models/sprites and their accompanying textures, animations and bone-structures (if applicable), be able to realistically calculate physics, have a user interface, and so on. Implementing all of this myself would require a considerable amount of programming time, time which could be better spent working on my AI engine. Because of this, and the fact that my project has a clear focus on the AI behind the game, it was an obvious decision to use an existing game engine. There are a number of free game engines currently available, some popular examples being Unity, which uses JavaScript, and is cross-platform (with publishing to iOS and Android also available), Epic Games' Unreal Engine (used in Unreal Tournament, Gears of War and Bioshock among others) which is similarly cross-platform, and Crytek's CryEngine (used for the Crysis game franchise). My choice of engine would also dictate the language that I would need to use to develop my project. Another advantage of programming my project in Objective C is that I would be able to use the popular open source CocoS2D 2D game engine. Cocos2D is perfectly suited to 2D games such as Lemmings, and includes support for asset loading, integrated physics (using the Box2D engine), a particle system and scene management among other things. It also has integration with a high score server, which would be interesting to use during user testing, to compare users results. Cocos2D also includes support for all iOS touch gestures as expected.

I will be using a Git server hosted on GitHub to store the source code and documentation for my project. Using source control is critical for any software development project, as it not only provides a way for multiple users to work on a single project (or even a single file) without the worry of file corruption, but more importantly, it also provides a way to track the incremental development of a project, with the ability to revert certain changes to an earlier version if need be. A big benefit of using GitHub is that it has a very intuitive online interface, and includes some nicely integrated bug tracking.

4 Research

I have documented my initial research in the form of an annotated bibliography. Along with each source's basic information, I have included a basic summary and any other notes where appropriate.

4.1 Artificial Stupidity: The Art of Intentional Mistakes (article)

Author: Lars Liden

Journal: AI Game Programming Wisdom

Year: 2004

4.1.1 Summary and Notes

Discusses the technique of building intentional flaws into AI systems to add to the entertainment value of the game. Largely focussed around the standard shooter genre.

> Fun can be maximized when mistakes made by computer opponents are intentional
>
> As an AI programmer, it is easy to get caught up in the excitement of making an intelligent game character and to lose sight of the ultimate goal; namely, making an entertaining game
>
> The hallmark of a good AI programmer is the ability to resist the temptation of adding intelligence where none is needed and to recognize when a cheaper, less complex solution will suffice
>
> The hallmark of a good AI programmer is the ability to resist the temptation of adding intelligence where none is needed and to recognize when a cheaper, less complex solution will suffice. The challenge lies in demonstrating the NPCs skills to the player, while still allowing the player to win

Tricks to building 'stupid' AI: - Move Before Firing - Be Visible - Have Horrible Aim - Miss the First Time - Warn the Player - Attack Kung-fu Style - Tell the Player What You Are Doing - React to Mistakes - Pull Back at the Last Minute - Intentional Vulnerabilities

4.2 Academic AI and Video games: a case study of incorporating innovative academic research into a video game prototype (proceedings)

Author: Aliza Gold

Journal: IEEE - Symposium on Computational Intelligence and Games (CIG'05)

Year: 2005

4.2.1 Abstract

Artificial intelligence research and video games are a natural match, and academia is a fertile place to blend game production and academic research. Game development tools and processes are valuable for applied AI research projects, and university departments can create opportunities for student-led, team-based project work that draws on students' interest in video games. The Digital Media Collaboratory at the University of Texas at Austin has developed a project in which academic AI research was incorporated into a video game production process that is repeatable in other universities. This process has yielded results that advance the field of machine learning as well as the state of the art in video games. This is a case study of the process and the project that originated it, outlining methods, results, and benefits in order to encourage the use of the model elsewhere.

4.2.2 Summary and Notes

Outlines a university-led AI project at University of Texas. NeuroEvolving Robotic Operatives (NERO) project looks at the evolution of neural networks with a genetic algorithm.

NERO has a real-time training stage which is carried out before the actual game. Player can 'save' the team, and start the game, when learning is no longer taking place.

Documents the development process

Used the Torque game engine (Garage Games)

Use a 'spiral' method of development (design choices made iteratively)

> In NERO, a player trains a group of ignorant robot soldiers by setting learning objectives for the group through an interface. After the objective is set, the robots learn in real time to achieve their goal

References

[1] *Academic AI and Video games: a case study of incorporating innovative academic research into a video game prototype*, 2005.

[2] B. Auslander, S. Lee-Urban, C. Hogg, and H. Munoz-Avila. Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. *ECCBR '08 Proceedings of the 9th European conference on Advances in Case-Based Reasoning*, page 15, 2008.

[3] L. Liden. Artificial stupidity: The art of intentional mistakes. *AI Game Programming Wisdom*, 2(5):8, 2004.

[4] J. McCarthy. What is artificial intelligence?, November 2007.

[5] R. Miikkulainen. Creating intelligent agents in games. Winter:9, 2006.