Planning and Research Report
CI301- 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"[**?**]. 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[?]. 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[?].

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' [?]) 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"[?]. 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, and given 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 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 AI-controlled characters in the game. For example, the Creature must learn what objects are suitable to eat based on taste, and what it sees others eating. This can result in the Creature learning to eat the player's followers, which may require the player to take corrective action. 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.

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 Lemmings 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 Lemmings; the characters would enter the level and simply walk continuously until they are killed, or reach an immovable object, after which they return the way they came.

3.1.1 Game Objects

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; I will need to make sure that there is 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.

3.1.2 Interface

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 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 project, and will operate behind the scenes to implement the machine learning techniques I intend to explore in this project. 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.

3.2.1 Basic Functionality

In order for my project to be a success, my AI system needs to have certain basic functionality. Firstly, the system will need 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.

3.2.2 Extra Functionality

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. 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.

3.2.3 Issues

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).

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 Development, 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.

[For a gantt chart of my project development plan, see appendix 1.]

3.3.1 Programming Languages

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, 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 get detailed analysis of the program's performance. 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. However, programming for iOS also brings with it severe limitations in performance and storage capacity.

3.3.2 Game Engines

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 obvious for me 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 the Unreal Tournament, Gears of War and Bioshock games among others) which is similarly cross-platform, and Crytek's CryEngine (used in the Crysis games).

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 is also highly optimised for use on mobile devices, and 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.

3.3.3 Source Code Control

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.

3.3.4 Project Management

I will be managing my project from a web-based interface hosted on my web server called ProjectPier, which enables me to set up tasks, milestones and a project wiki among other things. It also implements bug tracking, which I may use instead of the GitHub system.

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.

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

4.1.2 Quotations

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

4.2 Types of Machine Learning Algorithms (book)

Author: Taiwo Oladipupo Ayodele

4.2.1 Summary and Notes

Gives some nice information on the differenct types of learning:

- Supervised learning

- Unsupervised learning

- Semi-supervised learning

- Reinforcement learning

- Transduction

- Learning to learn

  with examples of algorithms for each, as well as background information.

4.3 The Complete History of Lemmings (webpage)

Url: http://www.javalemmings.com/DMA/Lem_1.htm
Last checked: 2011-10-21
Author: Mike Dailly

4.3.1 Summary and Notes

The history of Lemmings, as documented by one of the lead developers.

4.4 CS229: Machine Learning (webpage)

Url: http://cs229.stanford.edu/
Last checked: 2011-10-21
Organization: Stanford University

4.4.1 Summary and Notes

Homepage of the CS229 module: Machine Learning. Contains all of the lecture slides and handouts etc.

4.5 Machine Learning (webpage)

Url: http://www.youtube.com/view_play_list?p=A89DCFA6ADACE599
Lastchecked: 2011-10-21
Author: Andrew Ng
Organization: Stanford University

4.5.1 Summary and Notes

A YouTube playlist of the lectures for the CS229 module: Machine Learning. Lots of useful information about the different types of learning, algorithms etc.

4.6 Recognizing the Enemy: Combining Reinforcement Learning with Strategy Selection using Case-Based Reasoning (article)

Author: Bryan Auslander, Stephen Lee-Urban, Chad Hogg and Hector Munoz-Avila
Journal: ECCBR Proceedings of the 9th European conference on Advances in Case-Based Reasoning
Year: 2008

4.6.1 Abstract

This paper presents CBRetaliate, an agent that combines Case-Based Reasoning (CBR) and Reinforcement Learning (RL) algorithms. Unlike most previous work where RL is used to improve accuracy in the action selection process, CBRetaliate uses CBR to allow RL to respond more quickly to changing conditions. CBRetaliate combines two key features: it uses a time window to compute similarity and stores and reuses complete Q-tables for continuous problem solving. We demon- strate CBRetaliate on a team-based first-person shooter game, where our combined CBR/RL approach adapts quicker to changing tactics by an opponent than standalone RL.

4.6.2 Summary and Notes

Research into using case-based reasoning to allow reinforcement learning to respond more quickly to to changing conditions. A definite focus on the implementation.

Gives a nice introduction to the use of reinforcement learning in games.

If the system is doing 'well', it stores the current case for later reference.

Some nice, in-depth discussion of the algorithms used in the research, as well as the reasoning behind them - could be useful for designing my own algorithms later (also some pseudocode examples).

4.6.3 Quotations

> In this paper we present CBRetaliate, an agent that uses Case-Based Reasoning (CBR) techniques to enhance the Retaliate RL agent. Unlike most previous work where RL is used to improve accuracy in the case selection process, CBRetaliate uses CBR to jump quickly to previously stored policies rather than slowly adapting to changing conditions

4.7 Rapid and Reliable Adaptation of Rapid and Reliable Adaptation of Video Game AI (article)

Author: Sander Bakkes, Pieter Spronck, Jaap van den Herik
Journal: Transactions on Computational Intelligence and AI in Games
Year: 2009
Volume: 1 **Number**: 2

4.7.1 Abstract

Current approaches to adaptive game AI typically require numerous trials to learn effective behaviour (i.e., game adaptation is not rapid). In addition, game developers are concerned that applying adaptive game AI may result in un- controllable and unpredictable behaviour (i.e. game adaptation is not reliable). These characteristics hamper the incorporation of adaptive game AI in commercially available video games. In this article, we discuss an alternative to these current approaches. Our alternative approach to adaptive game AI has as its goal adapting rapidly and reliably to game circumstances. Our approach can be classified in the area of case-based adaptive game AI. In the approach, domain knowledge required to adapt to game circumstances is gathered automatically by the game AI, and is exploited immediately (i.e. without trials and without resource- intensive learning) to evoke effective behaviour in a controlled manner in online play. We performed experiments that test case- based adaptive game AI on three different maps in a commercial RTS game. From our results we may conclude that case-based adaptive game AI provides a strong basis for effectively adapting game AI in video games.

4.7.2 Summary and Notes

Discusses an alternative method to learning which functions without trials and without resource- intensive learning. Uses a downoaded 'case base' of previously played games to develop a strategy.

Some nice discussion on the different types of learning found in games (adaptive AI, difficulty scaling etc.)

Very nice in-depth descriptions (with examples) of the implementation of the learning. However, the paper is mostly focussed towards Real-Time Strategy games.

4.7.3 Quotations

Nice description of what constitutes an entertaining game:

> The purpose of a typical video game is to provide en- tertainment [1], [4]. Naturally, the criteria of what makes a game entertaining is dependent on who is playing the game. The literature suggests the concept of immersion as a general measure of entertainment [5], [6]. Immersion is the state of consciousness where an immersant's awareness of physical self is diminished or

lost by being surrounded in an engrossing, often artificial environment [7]. Taylor argues that evoking an immersed feeling by a video game is essential for retaining a player's interest in the game [6]. As such, an entertaining game should at the very least not repel the feeling of immersion from the player [8]. Aesthetical elements of a video game such as graphical and auditory presentation are instrumental in establishing an immersive game environment.

it is not uncommon that a game has finished before any effective behaviour could be established, or that game characters in a game do not live sufficiently long to benefit from learning

in our approach we consider that the challenge that is provided by the game AI should be adaptable to fit individual players

Adaptation mechanism:
// Offline processing
A1. Game indexing; calculate indexes for all stored games.
A2. Clustering of observations; group together similar observations.
// Initialisation of game AI
B1. Establish the (most likely) strategy of the opponent player.
B2. Determine to which parameter?band values this opponent strategy can be abstracted.
B3. Initialise game AI with an effective strategy observed against the opponent with the most similar parameter?band values.
// Online strategy selection
C1. Use game indexes to select the N most similar games.
C2. Of the selected N games , select the M games that best satisfy the goal criterion.
C3. Of the selected M games , select the most similar observation.
C4. Perform the game strategy stored for the selected observation.

4.8 Evolutionary Entertainment with Intelligent Agents (article)

Author: David B. Fogel
Journal: Computer
Year: 2003
Volume: 36 **Number**: 6

4.8.1 Abstract

A common limitation of conventional video games is that players quickly learn the positions and behavior of computer-controlled characters, which usually take the form of monsters.

Software developers pre-program these characteristics so, after playing the game several times, the player comes to know exactly how and when the monsters will act.

4.8.2 Summary and Notes

Author is CEO of Natural Selection Inc.

Interesting article documenting an experiment into using neural networks to devvelop a checkers playing program.

The system was run for about 6 months (840 generations of its evolution) and tested on www.zone.com against real players, where it gained a 'master' rating awarded to the top 500 players out of 120,000.

4.9 Creating Intelligent Agents in Games (article)

Author: Risto Miikkulainen
Year: 2006
Volume: Winter

4.9.1 Abstract

Video games provide an ideal platform for the development and testing of machine-learning techniques.

4.9.2 Summary and Notes

Article Focuses on neuroevolution, and gives some examples of where machine learning has been used recently (early 00s). It also has a nice summary of research in the area including NEAT and NERO.

Describes the NeuroEvolution of Augmenting Topologies (NEAT) - evolving neural network appropriate for video games

4.9.3 Quotations

> a large part of AI development is devoted to path-finding algorithms

> a large part of the gameplay in many games is figuring out what the AI is programmed to do and learning to defeat it

Evolutionary computation:

> Each solution is evaluated in the task and assigned a fit- ness based on how well it performs. Individuals with high fitness are then reproduced (by crossing over their encodings) and mutated (by randomly changing components of their encodings with a low probability). The offspring of the high-fitness individuals replace the low-fitness individuals in the population, and over time, solutions that can solve the task are discovered

Neuroevolution is particularly well suited to video games because (1) it works well in high-dimensional spaces; (2) diverse populations can be maintained; (3) individual networks behave consistently; (4) adaptation takes place in real time; and (5) memory can be implemented through recurrency (Gomez et al., 2006; Stanley et al., 2005)

Entirely new game genres can be developed, such as machine-learning games, in which the player explicitly trains game agents to perform various tasks

To rtNEAT, [the sliders] represent coefficients for fitness components. For exam- ple, the sliders specify how much to reward or punish agents for approaching enemies, hitting targets, getting hit, following friends, dispersing, etc.

4.10 AI Game Programmers Guild (webpage)

Url: http://gameai.com/
Last checked: 2011-11-6

4.10.1 Abstract

Founded in 2008, the AI Game Programmers Guild currently consists of over 200 professional game AI developers from all across the industry and from around the world. Our mission is to develop and promote excellence in game AI through education, community, and recognition.

4.10.2 Summary and Notes

Blogs, papers, presentations, videos etc. posted by industry professionals in the area of game AI.

4.11 Artificial Intelligence in Games (unpublished, possibly under-graduate assignment)

Author: James Wexler
Year: 2002

4.11.1 Abstract

With its latest release 'Black and White', Lionhead Studios has set the new standard for artificial intelligence in games. Creatures in this game have an incredible ability to learn that is implemented through a variety of AI algorithms and techniques. There is much that could be added to the game given ample time and computing resources. These additions include a partial order planner for creatures' actions, a game-state search

algorithm and planner for a computer-controlled opponent, and a dynamic lookup table to create better battling creatures.

4.11.2 Summary and Notes

Gives a nice history of video games with respect to their use of AI. Started in the mid-sixties with Pong (and similar).

Also has a nice description of the type of AI techniques used in Black and White - an example of a decision tree/weighting of actions

- Poorly written in places

- Needs more references

- Numerous subjective statements ("wonders of AI", "incredible")

4.11.3 Quotations

> A decision tree is built by looking at the attributes which best divide the learning episodes into groups with similar feedback values. The best decision tree is the one that minimizes entropy, a measure of how disordered the feedbacks are. The algorithm used to dynamically construct decision-treesto minimize entropy is based on (Ross) Quinlan's ID3 system

Richard Evens (Black and White AI developer)

4.12 Playing Smart - Artificial Intelligence in Computer Games (article)

4.12.1 Abstract

With this document we will present an overview of artificial intelligence in general and artificial intelligence in the context of its use in modern computer games in particular. To this end we will firstly provide an introduction to the terminology of artificial intelligence, followed by a brief history of this field of computer science and finally we will discuss the impact which this science has had on the development of computer games. This will be further illustrated by a number of case studies, looking at how artificially intelligent behaviour has been achieved in selected games.

4.12.2 Summary and Notes

Section 2 gives quite a nice overview of the history of AI, and some major developments/events in AI from 1931-1997.

Section 4 gives a nice summary of the main game AI techniques (rule-based, machine learning, extensible AI, knowledge-based) Black and While uses decision trees to implement reinforcement learning.

- Not particularly well written

- Some unsubstantiated claims

4.13 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

4.13.1 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.

4.13.2 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. Paper nicely documents the development process (used the spiral method). The game uses Garage Games' Torque game engine.

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.

4.13.3 Quotations

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.