top.pdf

Evolving Entertaining NPCs
Automated Synthesis of NPC AI with ADA

Project Report

Daniel E. Bruce

April 16, 2012
Halden, Norway

hiofmobapp_small.pdf

Abstract

Keywords: Mobile Applications, Meaning of Life, Rare Mammals

An abstract is a brief summarizing statement, usually between 75 and 150 words long. It gives the reader a synopsis of the problem, method, results, and conclusions of your document. The abstract takes the form of a paragraph, usually with 5-10 sentences. Abstracts are often collected into volumes and must be able to stand alone. They are read by parties who are trying to decide whether or not to read the main document. Sometimes they are read by people who want to get the big picture before reading the main document. Abstracts can save readers an immense amount of time.

Acknowledgements

Prerequisites (OPTIONAL)

Table of Contents

Chapter 1

Introduction

Computer games are a major part of the entertainment industry these days, with major titles having budgets matching Hollywood movies. Even so, there is a lot of potential for improvement on their appeal. Artificial Intelligence is an area where games can improve substantially as limitations of the past are disappearing, creating games that seem more realistic and interesting. Increases in the computing power of the average computer has made it possible to take into use techniques previously thought too computationally expensive. Machine Learning has the power to create NPC characters that seem intelligent and can add more realism to games, and as such make the game more satisfying to the player.

1.1 Motivation

Video games are a massive industry with budgets comparing to the ones of Hollywood movies, with Rockstar's GTA 4 having an estimated production budget of $100 million. It follows that video games are a big part of people's regular entertainment, with young people spending on average two hours per day on video games. (Norwegian Ministry of Culture, 2008)

Since the inception of the game industry, game producers have been heavily focused on creating more and more audio-visually impressive games, and this development has reached a point where the improvements made can only be minor. During this process, the development of the AI for game characters has only done as necessary, to meet the demand for game characters to improve along with the graphics. (Champandard, 2003)

Until the late 90s there had not been much focus on improving the AI utilized by the characters in games(Tozour, 2002), but this is shifting rapidly, as games focus more on intelligent characters, using it as their latest and greatest selling point. There have even been entire game series that focus entirely on AI, such as *Creatures* (Grand et al., 1997), *Black & White* (Lionhead Studios, 2001) and *The Sims* (Maxis, 2000).

Despite this, the majority of game AI still uses simple techniques that have been the cornerstone of game AI since its early days. Simple scripted rules and Finite State Machines still dominate the implementation methods due to their simplicity, performance and ability to function "well enough" to implement simple AIs. (Woodcock, 2001) This creates AI that, while entertaining and believable in most situations, is ultimately limited by the forethought of the AI programmers, and is easily outmatched by most experienced players.

In recent times Massively Multiplayer Online Games have emerged and become immensely popular, through titles such as *World of Warcraft*, and games have grown to have a high focus on Player versus Player multiplayer, with popular shooter game series having either no single player campaign, or an extremely short one. This trend gives a strong indication that players are looking for more intelligent opponents and more varied game play, and that this cannot be provided through the single player opponents in the games.

To alleviate the limitations of simple, static AI opponents, machine learning could be utilized. Machine Learning would allow game makers to create more engaging and interesting computer opponents, and enables the system to scale up to more complex worlds and situations. (Funge, 2004) In the same vein, it is believed that utilizing evolutionary computing can enable the creation of computer opponents that are interesting and interactive, and as such entertaining. (Yannakakis, 2005)

This thesis will explore whether ADATE (Olsson, 1994), a system for inductive programming (an type of evolutionary computing), can be employed to generate entertaining AI opponents. Additional challenges consist of finding a way to quantify the entertainment value of the generated agents, and the creation of a game framework to support the research. Towards this end, there are a set of objectives and questions (shown in the following section) that need to be addressed to reach our goal.

1.2 Objectives and Research Questions

Given the motivation for this thesis, there are a selection of questions to be addressed to further our objectives. The questions that will be answered in this thesis are as follows.

1. How can a player's satisfaction in a game (entertainment) be measured quantitatively?
2. Can the use of evolutionary computing increase a game's interest and ultimately create a more interesting game than static AI?

1. Establish a metric for measuring a game opponent's interest level to the player.
2. Adapt this metric to the ADATE platform and decide upon a test bed suited to the platform and metric.
3. Use the platform to enhance the entertainment level of the computer opponents in the test bed.

maybe change this when a test bed is located?

1.3 Summary

1.4 Thesis layout

Chapter 2

Background

2.1 AI techniques

There is constant work on new AI techniques, both to make AI more efficient, and to make AI seem smarter or more human. Familiarizing oneself with some of these will allow better ability to choose an implementation technique that fits with the goal of using inductive programming.

There are certain differences in approach between the techniques employed in "traditional" AI and game AI, primarily based on the performance of the various methods and their focus on the AI's ability to solve problems and be entertaining, respectively. The two areas seem to be converging, however, as traditional AI is starting to shift from the goal of being "really smart" and solving hard problems, to making computers that seem more human, have an ability to learn and adapt, and maybe act like assistants, as detailed in Ramos et al. (2008) and evidenced by the AI-controlled assistant feature, named Siri, in the Apple iPhone 4S (Apple Software, 2011).

2.2 History

Before going further it would be useful to go through the history of Artificial Intelligence, to see its roots, and how closely it has developed alongside the field of computer science itself (Buchanan, 2002; Luger, 2005).

The field has traces back to antiquity with Greek myths about artificial beings such as the golems and homunculus, which were "programmed" to perform tasks through occultic writing, and to the

writings of many ancient scholars (Aristotle and Euclid, among others) on the topic of reasoning, logic and the mind. Since then, there have been many people working on mechanical automaton through the ages, attempting to have them emulate human behaviour.

There was early work on intelligent machines from the very time computers existed, but computer AI as a field didn't really come into existence until 1950, through a pair of papers.

1. A very important paper was published by by Alan Turing named "Computing machinery and intelligence" (Turing, 1950), where the he defined the "Turing Test" as a way to test whether a machine was truly intelligent.

2. Claude Shannon publishes a paper on programming a computer to play chess, by representing it as a search problem (Shannon, 1950).

The term "artificial intelligence" wasn't coined until six years later when John McCarthy used the term for the first conference on the topic in 1956, where the first running AI program (the Logic Theorist) was demonstrated.

It didn't take long for AI programs to challenge humans at board games, with the first program to challenge a human world champion being made in 1962 by Arthur Samuel The program was aimed at the game of checkers and utilizing machine learning to improve its performance.

Further works into computer-assisted problem solving led to SAINT, that solved calculus at a freshman level (Slagle, 1963); ANALOGY, that solved the kind of analogy questions found on IQ tests Evans (1964); and ELIZA, which could simulate conversation (Weizenbaum, 1966).

Later, there was work into "knowledge-based programs" for artificial reasoning, which described programs able to interpret mass spectra of chemical compounds, solve integration problems in math, and play chess well enough to reach a class-C rating.

In 1969, the beginnings of Neural Networks were appearing with Minsky and Papert (1969) which defined Perceptrons, while several papers on natural language understanding were published during the next few years, along with the introduction of "expert systems" using rule-based programming, and the creation of the first computer to make a scientific discovery, the Meta-Dendral learning program (Buchanan et al., 1976).

Through the 70s and into the 80s, an explosion of well-known AI programs occurred. Many "expert systems" were made that were capable of reasoning within a limited knowledge space on the same

level as a human expert, using the traditional symbolic methods, before Neural Networks become widely used using backpropagation (first introduced in WERBOS (1974)).

Around the 80s, the American Association for Artificial Intelligence (AAAI, now named the Association for the Advancement of Artificial Intelligence after a name change in 2007) was founded. The AAAI hosts many conferences and symposia each year, and supports several journals on AI.

Another boom in AI research happened in the 90s with major advances in all areas of AI, with significant demonstrations in machine learning, intelligent tutoring, case-based reasoning, multi-agent planning, scheduling, uncertain reasoning, data mining, natural language understanding and translation, vision, virtual reality, games, and other topics.

The 90s also had two significant accomplishments of AI programs for playing games: Deep Blue beat Garry Kasparov in 1997, and TD-Gammon, an AI that played backgammon at championship-level, was written. In addition, AI was starting to see use in cataloguing the internet.

This brings us to today, where AI is proliferating, seeing use in toys (such as robotic pets) and other forms of entertainment, most notably video games. However, AI is still far away from the goal of creating a truly human machine.

2.3 Traditional AI techniques

As noted in Munakata (2008, chap. 1), AI techniques can be divided into six areas, although these divisions can be debated and should not be considered rigorous, namely:

1. Symbolic AI (also “traditional AI”)
2. Neural networks
3. Evolutionary Computing
4. Fuzzy systems
5. Rough sets
6. Chaos

Symbolic AI is the umbrella term for the traditional methods dating from the field’s inception, focusing on abstracting the world and applying logic and rules to reach decisions.

It is also known as “traditional” or “classic” AI, and was the approach that was used during the inception of AI and is still heavily used today. It is characterized by a top-down focus on logic and reasoning, and relies on a symbolic description of the world, such as a set of rules, and is thus often said to be “knowledge-based”.

Techniques included under Symbolic AI are knowledge-based systems, logical reasoning, symbolic machine learning, search techniques, and natural language processing.

The following two methods (Neural Networks and Evolutionary Computing) are different reactions to the insufficiencies of Symbolic AI by attempting to more closely modelling biological processes, so they can be called “biology-inspired AI”. They will be further covered in the next section (Section 2.4).

The remaining three methods are more recent developments which attempt to attack the problem from a different angle, collectively called “soft AI” due to their focus on not giving “hard answers” and reliance on uncertainty. These methods are less relevant to the topic of my thesis, and as such will not be covered further.

2.4 Biology-inspired AI

This term describes two sub-fields of AI, “Neural Networks” and “Evolutionary Computing”, both of which attempt to mimic biological processes, although not in a way that attempts to emulate them faithfully. The two are also part of a branch of Artificial Intelligence called Machine Learning, alongside techniques such as “Decision Trees” and “Bayesian Learning”.

Neural Networks

Neural Networks, more correctly called Artificial Neural Networks (ANN) to separate them from the biological variety, are computer simulations that attempt to mimic biological neural networks through a variety of algorithms, but they all revolve around small units called “neurons” linked in networks of some sort, trained by a learning algorithm.

The simplest variety of ANN is called a Feedforward Neural Network (Wikipedia, 2011b), and consists of one or more layers of neurons connected in a network where information only moves in one direction. The kind of neuron most commonly used in such a network is the Perceptron (Minsky

and Papert, 1969), and one usually uses the Backpropagation (Wikipedia, 2011a) algorithm to train these kinds of networks.

Other kinds of ANN are:

- Radial Basis Function Network.
- Kohonen Self-organizing Network.
- Learning Vector Quantization.
- Recurrent Neural Network.
- Modular Neural Networks.

ANNs are usually trained against a set of data containing both inputs and outputs, using a learning algorithm that dictates how the neurons in the network are to be updated based on the training data. The network is then tested against a separate "validation set" where the input data is sent through the network and compared with the output data to test the network's error rate. This process is then repeated as long as necessary to generate an acceptable error rate.

ANN can be used for multiple purposes, such as function approximation, classification, and robotics, but require a large and diverse set of data to train on to be useful, as ANN have a proclivity towards temporarily overfitting if presented with a long series of less diverse data.

Evolutionary Computing

In Evolutionary Computing the aim is to mimic the biological process of evolution to evolve solutions to problems. There are many approaches that fall under this name, such as Genetic Algorithms, Genetic Programming and Evolutionary Algorithms, but the common trait is that they use, to a certain extent, the four mechanics of biological evolution: reproduction, mutation, recombination and selection.

These methods present the problem by taking a starting point towards a solution, (which can be of any complexity, from nonexistent to a "best-known" program), then "reproduce" it by applying a random set of mutations to generate a certain amount of new solutions, called individuals. These individuals are then tested with a fitness function defined to measure the individual's performance at the problem, and the output of the fitness function is used to rank the individuals.

After ranking all the individuals, we choose one of the individuals in the list by some method, and "reproduce" it to create new individuals either by way of mutations or by using an algorithm to combine it with another individual in a way that mimics the biological process of recombination. The newly generated individuals are then tested for fitness and put into the list of individuals, which is then culled to a specific length by removing the least fit individuals. This process of reproduction and culling is repeated until the solution is deemed good enough or, on rare occasion, an optimal solution is found.

The system used in my thesis, ADATE (Olsson, 1995), is an Evolutionary Computing system that fits into a sub-field of Artificial Intelligence called Inductive Programming, which will be described in section 2.8.

2.5 Game AI techniques

The approaches to AI used in video games (as opposed to more traditional board/card games) are often very different from the academic approaches, where AI programs can take a long time to reach a decision, might require massive amounts of resources and usually have as a goal to perform as excellently as possible at a task.

In games the AI actors might only have a handful of microseconds of CPU time available to reach a decision without harming the performance of the game, and the AI must fulfill the twin goals of being challenging, but beatable by the majority of players, and entertaining.

In addition, game AI is a relatively young field when compared to the research on AI using traditional board and card games, a field which evolved alongside computer science, since "solving" board games was one of the driving forces behind computer science (Schaeffer and Van den Herik, 2002).

2.5.1 History of AI in games

Before covering specific techniques, it is prudent to go through a history of AI as used in games, to show the field's evolution in contrast to the field of traditional AI with its long history of strong scientific focus. (Tozour, 2002)

The methods employed in game AI have been, and still are, marked by the heavy performance

requirements and the fact that very little emphasis has been put on AI sophistication until recently, as quipped about in the following quote:

> Even today, game AI is haunted by the ghosts of Pac-Man's Inky, Pinky, Blinky and Clyde. Until very recently, the video game industry itself has done all too little to change this perception.

Many of the popular early games used very crude AI, basically just consisting of a handful of simple rules, with the exception of games that just digitized board games with well established AI research such as chess.

More sophisticated AI in video games was first embarked upon in the context of turn-based strategy games (such as *Civilization*), then in real time strategy games (such as *Age of Empires 2: The Age of Kings* and *WarCraft II*). Further, good AI started showing up in First Person Shooter games (*Half-Life* and *Unreal: Tournament*) that showed tactical ability and the ability to model several actors simultaneously, while *Thief: The Dark Project* had actors that emulated sense of sight and sound in a human-like fashion, and *SWAT 3: Close Quarters Battle* featured randomized AI parameters that allowed each actor to have a slightly different personality every time the game was played.

After that the variety of different AI exploded, with games focusing entirely on nurturing "AI life" in a manner the player decided, with *SimCity* and *The Sims* being very well known staples. Another well-known AI life game was *Creatures* which is famous for being one of the few games that actually used a biological model for its "Norns", both modelling biological processes with great detail and using neural networks for the AI (see Grand et al., 1997). Other games fit into the "God games" category, such as *Populous* and *Dungeon Keeper*, alongside *Black & White*, which has the distinction of being the first major game to focus the player's attention entirely on the game's AI capabilities, and including a learning AI, a topic which is considered to be the next "Big Thing" in gaming.

It should also be noted that for all the recent complexity, it is still the case that the game AI community favours simple "traditional" methods implemented through finite state machines, decision trees and rule systems, for their excellent performance and relative simplicity. These are then further augmented to add human-like behaviour, simulating planning and learning (see Isla and Blumberg, 2002; Khoo et al., 2002; Mateas and Stern, 2002; Orkin, 2003).

2.5.2 Traditional game AI

Traditional game AI or "simple AI" is what game AI started out as, and still to this day mostly is. It's based upon simple rules or simple logical systems, and has traditionally used a sampling of simple techniques, such as simply hard coding the AI in a single routine, utilizing Finite State Machines, or using rule-based systems. In addition, path finding has always been a topic with AIs, where simple, well known, algorithms such as A* have been dominant for a long time.

The common factor in traditional game AI techniques is that they tend to be static, the NPCs are only capable of the things they were programmed to do beforehand, and have little capability for learning or planning.

Hard-coded AIs

The first game NPCs utilized simple hard-coded AIs, which were basically a short routine that ran every tick of the game with simple behaviours for the opponents, not following any formalized methods for AI. This method is still used to this day, as it's easy to create, and usually many minor opponents in games don't need more sophistication than this method creates.

A good example of this technique is Pac-Man's ghosts, which used simple path finding, and a selection of "modes" for the ghosts globally, then four different targeting rules to give each ghost a "personality" as their entire AI (Birch, 2010; Pittman, 2011). Even this very simple AI created the interesting behaviours and engaging game play which propelled Pac-Man towards becoming one of the most recognized games in video game history, evidence that you don't need fancy high-powered algorithms to create interesting and engaging games.

Finite State Machines

Finite State Machine (FSM) systems are the most commonly used systems in game AI, although the FSM systems used by game developers do not necessarily work the same as the ones described by Computer scientists. They take certain shortcuts which violate the traditional definition of FSM, which make them more applicable to games (Rabin, 2002).

FSM are used to formalize an NPC's behaviour in a simple way, as states and transitions. States correspond to a specific behaviour or action, whereas a transition correspond to a change in behaviour

due to an event in the game. Using this, one can easily map up simple behaviours that allow an NPC to act in a manner that can be deemed "intelligent."

This method of AI creation has the benefits of being simple to understand, create and debug, as well as being very versatile by virtue of being a general method that can lend itself to most any problem. Of course, there are downsides to FSM as well. They can easily grow out of hand in more complex AI systems, need to have every necessary transition programmed in, and they don't have the capability of combining states (so you can't have an NPC be in the *run away* and *attack* states at the same time, to have it do a tactical retreat, without explicitly programming the option in).

For a more thorough coverage of implementing an FSM system, one can consult Rabin (2002) or Kirby (2011, chap. 3).

Rule-based systems

As with FSM, Rule-based systems stem from traditional AI research, but are used in a more loose form in video games (Christian, 2002). The basic idea of rule-based systems is that of a database (the data can be information, actions or other things), where each piece of data has a "matching rule" used by the system to infer which pieces of data apply to the current situation.

In the context of games, this boils down to series of rules coupled with actions or behaviours, many of which can apply at the same time, which together make up the AI for the given NPC. These are formally named *reaction rules*, and are just one type of rule, with the other major one being *consequent rules* that deal mostly with information.

Usually there is some method of discerning which rules are the most relevant at the current time if many of them match the current situation, to prevent the NPC from doing too many things at once (many simple AIs only allow the NPC to do a single action per tick). These can range from randomly choosing a rule, to weighting each rule based on its specificity and choosing the most applicable one (Freeman-Hargis, 2002).

The biggest benefit of rule-based systems is that you can create a good set of behaviour with a comparatively small amount of rules. This means you aren't suspect to the "state explosion" you might have in an FSM, where adding a single new state will result in a cascade of new transitions having to be written to handle every single case where the NPC can switch in and out of the new state. In a rule-based system you just add the new rule and write one matching function for it, and let your system handle the rest.

Of course, the downside in this case is that creating a good AI like this requires a bit more thinking to create a set of actions that acts well in the most common cases, and still handles uncommon and unexpected cases decently well, since one can't write a rule for every situation. This usually requires a human with some expertise to either formulate or actually code the rules.

More info on writing rule-based systems can be found in Kirby (2011, chap. 4).

2.5.3 Towards more human game AI

In recent times there has been efforts towards making game NPCs act more human-like, by introducing planning and learning behaviours (making the AI adaptable), making their decisions be more unpredictable (but rational), culling any obviously stupid behaviour, as well as engineering the NPCs to introduce more human elements such as adding the feeling of emotion and body-language (Spronck, 2005).

In the last decade or so, game AI has started incorporating planning and learning behaviour to make NPCs seem smarter and more human-like. Games such as F.E.A.R (using Goal-Oriented Action Planning as described in Orkin (2006)) utilize planning to make their opponents behave in a more realistic manner, while other projects have used planning to make AI for previously released games, such as WarCraft II (using Hierarchic Task Networks as described in Brickman and Nishant (Unknown year)).

Planning allows the AI programmer to specify "action sets" and "goals" which together make up the bulk of the AI for the NPCs. Action sets contain actions which satisfy goals, and each action may have preconditions which are also classified as goals. Using this information, the NPCs can plan its actions by what it knows of the world (which it can learn over time) and which actions have been defined as being accessible to it, instead of having to have the AI programmers code every possible such path.

An example of the benefit of using a planning system is described in Orkin (2006), where they describe an NPC being shot while sitting at a computer desk. In the traditional FSM-based system, the NPC had to fully exit its current state before being able to transition to the next. This lead to the situation where the NPC would exit its "work" state by turning off the computer, pushing out the chair, and standing up, before transitioning to the "death" state, letting out a death groan and falling over dead. Not really what one would consider human-like behaviour. In a planning system, that transition would be dynamically created, and the NPC would just slump over dead from its chair, as

one would expect.

Introducing emotion into games can be done in various ways, not just through having the NPC itself show emotion, but also by having various "AIs" change the mood of the game by altering lighting, music, and texture (Kirby, 2011, chap. 9).

NPC difficulty can alter itself according to player skill to fulfill any of these goals, something which many modern games utilize in what's called "Difficulty scaling", which can refer to anything from simply giving you enemies with higher levels (in RPG-style games such as Fallout 3), spawning more or less enemies and changing where they appear (such as the AI Director in Left 4 Dead), to evolving NPC AI using the player's performance as a parameter in a fitness function to create better AI.

Add a reference here?

2.5.4 Advanced game AI

More advanced game AI utilizes more heavy-duty AI techniques such as Genetic programming or Neural Networks that learn and train as the game goes by, in an effort to create an ever-changing experience for the gamer. The most well known games which utilize these kinds of techniques are Creatures (Grand et al., 1997) and Black & White.

One technique used to allow these advanced agents is called Reinforcement Learning (Merrick and Maher, 2006; Sutton and Barto, 1998), which differs from the normal learning/training method of machine learning, termed Supervised Learning, in that it isn't given any examples to learn from to learn how to behave. In Reinforcement Learning the agents start out with a defined set of actions and goals, and have to learn through trial and error which ones are the best series of actions to fulfill a certain goal, which can easily change during play as the human player changes its tactics.

Beyond this, the usage of more advanced AI techniques in games has been mostly within the realm of research and done after the release of the games, either through altering the source code of released games, or by creation of bots that use a certain AI technique for research purposes and pitting it against humans and/or other bots in a multiplayer setting.

2.6 Evaluating entertainment

When AI is used for NPCs in games, the main goal is not to perfectly emulate a human, nor to create the most skilled opponent possible, but rather to create an opponent that *seems* human, and possesses behaviour that creates an entertaining experience, without being unduly challenging.

There are many methods of doing this, some of which have been described previously in section 2.5. These range from simple difficulty-scaling methods to match the player's skill level, through having NPCs that perform "human" tasks such as planning and cooperating smartly, to having NPCs and AI that show emotion and alter their behaviour and how they are perceived based on their moods.

Before you can make AI more entertaining, you need a way to measure if you're actually reaching your goal. This can be done in one of two ways, you can measure the response from real human beings playing your game, or you can try to quantify entertainment value in an objective manner through various means.

2.6.1 Measuring people's entertainment

Measuring a real human being's response to a game will give you the most accurate measurement of entertainment, but it comes at a cost. The methods of doing so can be very expensive, in equipment and/or time. There is also ample room for bias either by the one doing the measurements or the measured person, conscious or subconscious.

Techniques employed here can range from very primitive to very advanced, and include everything from surveys, to hooking a player up to gear to measure brain waves.

Simple surveys or questionnaires are easy and cheap to implement, but suffer from not being able to capture complex data. In addition, and suffer from bias on the part of the person filling out the data, where they subconsciously fill out data closer to what they think you might be looking for than what they actually think (Mandryk et al., 2006).

Another technique that can be used is recording a person playing the game, and analyzing the recording to pull out data about how the person playing the game feels about it. This can provide a very rich source of data, but suffers from drawbacks, the biggest of which is that time spent analyzing is very high compared to the amount of video recorded, ranging from 5 times to 100 times longer. Another problem, as always, is bias, this time on part of the researcher (Mandryk et al., 2006).

The previous techniques are mostly indirect measurements, but there are ways to directly measure a person's enjoyment of a game, by utilizing physical measures such as Galvanic Skin Response, Electrocardiography and respiration. This involves hooking the players up to a lot of equipment, and is thus a lot more expensive, time consuming and intrusive. One way of doing this is described in Mandryk et al. (2006), another in Yannakakis et al. (2008).

2.6.2 Quantifying entertainment

There is also work going into methods of quantifying entertainment in games, with several articles proposing theories on what makes a game fun (e.g. Federoff, 2002; Koster, 2004; Lazzaro, 2004; Malone, 1981; Read et al., 2002), and others proposing methods for augmenting or optimizing entertainment value in games (e.g. Yannakakis, 2008; Yannakakis and Hallam, 2004b, 2007b, 2009).

The presentation "Why we play games" (Lazzaro, 2004) states that there are four keys to making a fun game: "Hard Fun", in which the joy of overcoming difficult challenges drives the player; "Easy Fun", where immersion into the game world is more important than challenge; "Altered States", in which the game elicits some change of the player's internal state; and "The People Factor", where the social aspect of the game adds to, or creates, the entertainment. These could be used as loose measures of how fun a game is, and to whom it appeals, by quantifying the amount each of these keys are exhibited in the game. These views are echoed and expanded heavily upon by Federoff (2002), which mostly comments on the first two keys, as "challenge" and "immersion".

A related view, as argued by Yannakakis and Hallam (2004a,b), is that interactive or interesting opponents is the key to making fun games, and the key to having interesting opponents is for them to adapt to the player, so that the player doesn't get to a point where the AI is too simple for them. The key to doing this, it's argued, is having a way of directly quantifying "fun".

As demonstrated in the reports, being able to do this quantification is important for being able to evolve "fun" AI offline. It is prohibitively difficult to make a fitness function for entertaining NPCs otherwise, as it is unreasonable to expect to be able to have a player play against every new individual and rate the AI's fitness.

The method utilized in the reports is to pit the evolved NPCs against a sampling of pre-programmed AIs standing in for the player, and then using a quantification formula of the NPC's interestingness and difficulty as the fitness function or "measure of fun".

2.7 Creating entertaining NPCs

As noted in the section 2.6, one of the primary factors in a game NPC's entertainment level is realized by having them be "interactive" opponents. In that vein there is some work on increasing how human-like game AI is behaving, as a way of increasing entertainment value, which involves planning behaviours, anticipation, learning and adapting (Orkin, 2003, 2004; Spronck, 2005; Yannakakis and Hallam, 2009), as described in previous sections.

There is an approach called character-based AI collecting many of these techniques (Isla and Blumberg, 2002), which is gaining grounds in the game industry as a way to give the game's AI agents a reasonable facsimile of human behaviour and personality, or a "character", to make game opponents and allies more entertaining. The term is applied to AI which collect many techniques and behaviours that make the AI seem more human, like the aforementioned planning, learning and adapting, but also the ability to sense patterns, the ability to anticipate, and a certain model of the AI actor's perception that limits its available information to a more "realistic" amount.

Another way of making NPCs more entertaining is by having them present a constant challenge to the player, without them being too challenging. One way of doing this is having the NPCs utilize Reinforcement Learning (Merrick and Maher, 2006; Sutton and Barto, 1998) to present a continuously learning opponent. Another is to use one of the methods of difficulty scaling to present different NPC opponents that correspond to the player's skill level, a technique already utilized in many games at various levels of sophistication.

Another possible way of increasing an NPC's entertainment level is by increasing the NPC's interestingness by making them act in interesting ways by way of automatic programming. This method utilizes offline learning, and possibly online learning, to generate NPC behaviours that would create a more interesting experience. Yannakakis (2005)

2.8 Inductive Programming

As noted in section 2.4, ADATE (Olsson, 1995) fits into a sub-field of Artificial Intelligence called Inductive Programming, which recently emerged from being fragmented across the fields Inductive Logic Programming, Genetic Programming, and Machine Learning (Kitzelmann, 2010).

The aim of Inductive Programming, as described by Approaches and Applications of Inductive

Programming (AAIP, 2010), is to create programs based on incomplete specifications. These specifications are given as a set of inputs with a matching ouput specification. The output specification can either be a matching set of outputs, or an output evaluation function that evaluates how well the output generated by the program matches an expected output. This type of specification differs from the specifications required by a deductive approach, where one starts with a high-level description of the wanted system, then have the system generate code that matches the description.

The biggest difference between the two approaches is that the Deductive approach guarantees that a solution is correct, but it requires the problem to be well- understood and a specification to be written. On the other hand, the Inductive approach does not guarantee a fully correct solution, but requires a much simpler specification only containing input and output data to create its solution.

Inductive Programming systems often accept other, non-mandatory, inputs in their specifications to help guide them towards an acceptable solution. These can be in the form of constraints on efficiency and complexity, background information about data types, functions available for use in the generated code, information about the flow of the intended program, an initial starting point for the generated solution, and heuristics or bias to guide the system towards a certain solution.

This thesis will be using an Inductive Programming system named ADATE system, which will be explained in detail in the next section.

2.9 The ADATE system

In this section we will give an overview of the way ADATE functions. Further information is available in two places: the user manual (Vattekar, 2006) contains general information necessary to understand and use the system, and the dissertation describing the system (Olsson, 1994) contains very in-depth coverage.

ADATE's approach to Inductive Programming, formally named Evolutionary Inductive Programming (Crossley et al., 2009), applies evolutionary computing to the problem of making computer programs. In this approach, the basic unit (the "genes") which are manipulated by the evolutionary process are computer code operations, but otherwise the process is very much like the one described in the previous section about evolutionary computing (section 2.4).

All Inductive Programming system generate code in a specific programming language, usually simplified or created for this specific purpose. ADATE uses a subset of Standard ML, called ADATE-ML, for its generated code as well as for the parts of the specification that it will need to access. (see Vattekar, 2006, Chap. 3)

As mentioned before, all Inductive Programming systems make use of a specification with the input and output necessary to frame the problem. ADATE's specifications only make use of an output evaluation function, it does not support, out of the box, the simpler variety of matching input data with output data, although it is trivial to write an output evaluation function that works in such a manner. ADATE also requires an initial program to be part of its specification, although this program may be as simple as an empty function, as the method it uses to generate programs is a form of local optimization, in which a program is continuously improved to find better and better solutions.

ADATE's specification is written in two parts separated by a "%%" token. The first must be written in ADATE-ML and the second can use the full functionality of Standard ML.

The ADATE-ML part contains three different things:

1. The initial program, called f, that the ADATE system will start from and improve. This can either be an empty function or a function that serves as a starting point.

2. A function $main$ which is called by the ADATE's evaluation cycle, and can provide a scaffolding or a context around f. This is essential if f is supposed to be a small part of a larger program which requires the presence of a larger environment to evaluate the generated function's performance. It is also useful if the performance of the program is dependant on the performance of $main$ itself, which may call f several times (perhaps not a constant or determinable amount) during its execution.

3. Definitions of data types and functions that can be used by the code that is generated by ADATE. This will be all the supporting code that is necessary to support the simulation environment, and any functions that f might want to use to make sense of its inputs.

The Standard ML part of the specification define a certain set of callbacks that are called by the ADATE system to perform some of its functions. These callbacks define things such as the input data, validation inputs, the set of functions that should be available for use by f, the function that will evaluate the output of $main$, the parameters governing amount and complexity of generated

individuals, and other minor things concerning the execution of the system (see Vattekar, 2006, Chap. 4.2 for details). This part of the specification can also contain arbitrary user supplied code that is needed as part of the evaluation, but does not need to be accessible by the ADATE-ML part of the code.

As ADATE is an evolutionary system, it uses the same general process as other genetic systems: an infinite cycle of procreation, evaluation and culling of individuals in the search for ever improving performance.

To create new individuals, ADATE mostly uses mutation of already existing individuals; genetic crossover is only used in a very limited fashion. Whenever ADATE needs to create a new individual it starts by picking an individual from the "kingdom" of existing individuals, initially consisting only of f. ADATE then applies a series of transformations called "compound transformations" to create the new individual.

The six basic transformations available for use by ADATE are as follows:

R is the only transformation that can actually change the semantics of a program, and functions by replacing a part of the program with a new synthesized expression.

REQ is an R transformation with the additional constraint that the resulting transformation does not make the program perform worse.

ABSTR abstracts a piece of the program out into a separate function which can then later be reused by the program in several places.

EMB updates an existing abstract function by adding a new parameter to it and ensures all calls to the function take into account the new parameter.

CASE-DIST rearranges code within $case$ and let expressions in a way such that semantics are preserved.

Crossover takes a series of REQ transformation and treats them as alleles in a genetic recombination. This is the only case in which ADATE uses genetic crossover.

The individuals are then evaluated by running $main$ with the current individual as f over all the examples given. The outputs from $main$ from all the inputs are then evaluated by the function specified in the specification file. The evaluation value as well as some other data regarding run time

and the complexity of f is then used to compute a few values regarding the individual's performance. If this individual's performance is better than any of the solutions currently found in the kingdom, it is then grouped with others of similar performance in the kingdom. The kingdom is then culled by removing the lowest ranked individual.

The process then continues by creating a new individual from the same base individual as before, until a given time limit is elapsed, at which point a new base individual is chosen.

This loop continues until the program is halted, while output regarding the state of the kingdom is written to disk while the program runs. This output shows the current state of the system and the current best individuals, along with the generated code. By evaluating this output, the user of the system can decide whether a good enough solution has been created, the system is "stuck" and unable to create better individuals, or the system needs to run for a longer time but is still producing better individuals.

2.10 Evolving Game AI

Using evolutionary techniques for game AI is not an entirely new topic, although doing it with the goal of creating entertaining or fun AI is a recent development. The major body of research on this topic focuses on evolving an agent that can successfully play a game, or execute certain maneuvers, in a human-seeming manner.

Much of the research on evolution in games follows the research on learning in games focusing on First Person Shooter games, in particular well-known ones with good facilities for creating "bots". Examples of such research include using evolution to tune the parameters for a bot in the game *Counter Strike* (Cole et al., 2004), and using evolution to teach a bot to do rocket jumps and dodge fire in an unspecified game (Champandard, 2003).

Other research on using evolution to create game AI, using other kinds of games, include the use of evolution to create an AI to play the RTS game *WARGUS*, a *WarCraft II* clone implemented in the *Stratagus* engine, (Spronck et al., 2004). Another example is the use of an ice hockey environment to explore evolutionary learning (Blair and Sklar, 1999). There are a couple pieces of research on using online co-evolutionary methods against human players over the Internet, using the game Tron (Funes and Pollack, 2000; Funes et al., 1998). In addition, there has been research into using offline and online co-evolution with action games (Demasi and Cruz, 2003).

There is little research on evolving game AI with the direct purpose of creating NPC behaviour with the primary goal of being entertaining; most of the body of work on using evolution for game AI focuses on creating human-like skill. One body of work that does explore the specific topic of entertainment is the work of Georgios Yannakakis, whose Ph.D. dissertation was on the topic of using evolutionary computing with neural networks to create interesting opponents (Yannakakis, 2005).

In his work, Yannakakis describes a collection of metrics that can be used to quantify entertainment (Yannakakis, 2005) and uses a combination of experimental data and user surveys to verify their applicability (Yannakakis, 2008; Yannakakis and Hallam, 2007a). These metrics are combined into an interest function used to train offline neural network agents on a collection of predator/prey games to show that one can create interesting agents through evolution on neural networks. Predator/prey games were used because they have been the basis of many works on evolutionary computing in games (Gallagher and Ryan, 2003; Koza, 1992; Lucas, 2005), and have interesting properties due to being a dynamic multi-agent environment.

Chapter 3

Methodology

This chapter presents the methodology used to complete the aims of the thesis, primarily the use of inductive programming, via ADATE, to evolve interesting game opponents. With that in mind, we describe the overall experimental method, followed by the description of a prototype game to be used for simulation and then we will briefly cover some details of the actual implementation of the simulation platform and its specification.

We will be basing our experimental method on the one presented in Yannakakis (2005), due their high applicability to our goals. More specifically, we will be using our own implementation of the Dead End game testbed as described in Yannakakis (2005, Chap. 5), as well as the interest metrics described in Yannakakis (2005, Chap. 2). The details of the testbed and the interest metrics, as well as our changes to them, will be noted throughout this chapter.

3.1 The Dead End Game

The simulation environment we will be using is an adapted version of the Dead End computer game used in Yannakakis (2005); it is a predator-prey game where the player (the prey) has to reach the goal before being captured by a group of opponents (the predators). This game environment is sufficiently simple that it's easy to model, but also sufficiently complex and dynamic to provide a good basis for using evolutionary techniques to create interesting AI strategies. The game will be considered from the viewpoint of the opponents (the predators) in an attempt to evolve characters that exhibit the ability to cooperate based on partial information, with the capability to play effectively

against a player following a fixed strategy.

3.1.1 Description

In this section we will describe the Dead End game in sufficient detail to give a good understanding for how it works, as well the ways in which our version differs from the reference implementation. (see Yannakakis, 2005, Chap. 5)

The game is played on a two-dimensional field of size 16cm x 16cm [1] with a rectangular area at the top representing the goal . The characters in the game are the player character, named *'Cat'*, represented as a circle of radius 0.75 cm, and an arbitrary number of enemy characters, named *'Dogs'*, represented as squares of dimension 1.5 cm.

Insert size of goal here?

The aim for the Cat is to reach the exit from its starting position, situated randomly at the bottom of the map, by avoiding the Dogs or to survive until the end of the simulation, i.e. after a certain amount of simulation steps. On the other hand, the aim of the Dogs is to capture the Cat before it reaches the exit or the simulation ends.

Since there are more Dogs than there are Cats on the playing field, they are given a handicap to even the playing field. In our implementation, the Cat moves at a fixed rate of 2 cm per simulation tick, while the dogs move at $3/4$ that speed, 1.5 cm per simulation tick. In addition, it is important to note that the Dogs have the property of permeability, i.e. they can occupy the same space as other Dogs. Given the speed handicap the Dogs have, it is impossible for a single Dog to catch the Cat, so the Dogs must cooperate to trap and catch the Cat. This is where the opportunity arises for emerging interesting AI behaviour.

At the beginning of the game the characters are positioned so they are sufficiently spaced out. As mentioned, the Cat starts at a random position flush with the bottom of the field, whereas the Dogs start at a random position in the upper half of the playing field.

Then, for each simulation tick, the following happens:

1. All entities (Cat and Dogs) in the game gather information from the playing field, run their AI routine, and produces a movement decision: up, down, left, or right.
2. All entities are moved according to their movement decisions.

[1] While we use the unit cm to refer to the units within the game, this is purely for convenience and is a completely virtual unit with no physical correspondence.

3. The field is checked for any collisions. If there are any, the game is marked as over, and the win condition is checked. If the Cat has collided with the goal, the game is considered won, if any of the Dogs collided with the Cat, the game is considered lost, and if both occurs the tie is broken by considering the game won. (The Cat narrowly escapes, with a suspiciously teeth-shaped mark)

4. If the game is not over yet, but the amount of simulation ticks is greater than 50, the game is considered over and won.

5. If the game is over, a new game is started with a new random cat position, but the same dog positions.

Should we do this?

3.1.2 The Cat

To be able to evolve good behaviour for our characters (the Dogs), it is necessary to have a competent player character, as the behaviour of the Dogs is strongly related to the behaviour of the player character. As it isn't feasible to provide the player character with human input during the simulation process, it is necessary to provide one or more strategies for the player to execute in an attempt to reach its goal. For the Dogs to be capable of diverse behaviour it is preferred to have several behaviour for the player, so in our case we have implemented two fixed strategy behaviours for the Cat.

These strategies are implement as AI functions which take as input a complete state of the field containing the following:

Maybe a list like this is overkill?

Self The (x, y) coordinates of the current Dog.

Cat position The (x, y) coordinates of the Cat.

All Dogs A list of the (x, y) coordinates of all the Dogs.

Goal position The (x, y) coordinates of the Goal.

and produce as its output a movement direction: left, right, up, or down.

Exit-Achieving Cat

This is a very simple Cat behaviour. It simply chooses the movement direction that reduces the coordinate which has the greatest relative direction from the Goal. If the Goal has position $(8, 1)$ and the cat has position $(7, 10)$, the Cat would choose to move up, as the Y direction has the biggest difference.

Potential Field-Based Cat

This is a more complex and efficient Cat behaviour based on a Artificial Potential Field (APF), where the cat is attracted or repelled by objects in the playing field, and moves along the path of least resistance towards the Goal. The implementation is heavily based on the one in Yannakakis (2005, Chap. 5.1.1.3) and considers points along the path to the Goal as attractive, and the Dogs as strongly repulsive.

The overall effect of the forces on the field gives the cat a map of movement costs by plotting the function $C(x, y)$:

$$C(x, y) = \Delta_E(x, y) + D(x, y) \tag{3.1}$$

$$\Delta_E(x, y) = \sqrt{(x_e - x)^2 + (y_e - y)^2} \tag{3.2}$$

$$D(x, y) = \sum_{i=1}^{N} \frac{\rho}{|x_{d,i} - x| + |y_{d,i} - y|} \tag{3.3}$$

where N is the number of Dogs, (x_e, y_e) is the center of the Goal, $(x_{d,i}, y_{d,i})$ is the center of the i^th dog, ρ is the cost of a dog in the APF (which in our implementation is 1000, as it is in the reference implementation).

This field allows the Cat to select the direction that will incur the lowest costs by plotting $C(x, y)$ at 0.5 cm intervals for 2 cm in each movement direction from the Cat's center coordinate, computing the average cost of moving in that direction, and selecting the direction with the lowest movement cost.

This strategy is intended to represent a very good strategy for the Cat, on level with the performance

of a Human player, and as such is well suited as a surrogate for a human player in the simulation.

3.1.3 The Dogs

This point marks the biggest deviation from the reference implementation, which utilized Dogs controlled by Feed-forward Neural Networks. (Yannakakis, 2005, Chap. 5.1.2) Our dogs will instead be controlled by a Standard ML function generated by the ADATE system. The generated function accepts input similar to the input given to the the neural network described in the reference implementation, and produce similar output.

Input

The input to the Dogs AI function is a snapshot of the state of the field with the following values:

Self The (x, y) coordinates of the current Dog.

Cat position The (x, y) coordinates of the Cat.

Nearest k dogs A list of the (x, y) coordinates of the k closest dogs.

Goal position The (x, y) coordinates of the Goal.

Take notice that the Dogs do not have access to the position of all the dogs in the field, but only the nearest k dogs, this places limitations on how closely they can cooperate.

Output

The output is the Dog's chosen movement direction: left, right, up, or down.

3.2 Interest Metrics

As given in Yannakakis (2005), there are three criteria that define an interesting — and as such entertaining — game, from which our fitness function will be derived:

1. *The game must not be too hard, nor too easy.* The game is interesting when the opponents strike a balance between "Hard Fun" and "Easy Fun" (Lazzaro, 2004), in that they are not too hard to dodge, but also provide some level of challenge.

2. *The behaviour of the opponents must be diverse over games.* This requires the opponents to change their tactics or behaviour over subsequent games, which make them less predictable, and thus more entertaining.

3. *The opponents must be dynamic and exhibit spatial diversity.* The opponents should move around and try to cover the entire field, instead of staying in a certain area of the field. This gives the impression that they're actually intelligent entities, and thus making them more entertaining.

From these criteria, a fitness function is derived (Yannakakis, 2005, Chap. 2.2.2) that combines three metrics, each quantifying a specific criterion. The game is played N times (for a maximum of t_{max} ticks each time), and we record the amount of ticks t_k taken to finish the game in addition to the amount of times ν_{ik} each 1cm x 1cm cell i of the field is visited for each game k.

Appropriate level of challenge

Following the first criterion, quantifying the level of challenge in the opponents, we get (3.4):

$$T = [1 - (avg(t_k)/max(t_k))]^{p_1} \tag{3.4}$$

where $avg(t_k)$ is the average number of ticks taken to finish the game over the N trials; $max(t_k)$ is the maximum number of ticks t_k taken over the N trials, bounded by t_{max}; p_1 is a weighting parameter.

This metric increases as the diversity in the amount of ticks taken to finish the game increases, ensuring that it is not too easy or too hard.

Diversity over games

Quantifying the second criterion, diversity across games, gives us (3.5a):

$$S = (\sigma_{t_k}/\sigma_{max})^{p_2} \tag{3.5a}$$

$$\sigma_{max} = \frac{1}{2}\sqrt{\frac{N}{N-1}}(t_{max} - t_{min}) \tag{3.5b}$$

where σ_{t_k} is the standard deviation of t_k across the N trials; σ_{max} is an estimate of the maximum value for the standard deviation of t_k; t_{min} is the estimated minimum amount of ticks required for the game to be over (3.0 in our implementation); p_2 is a weighting parameter.

This metric increases as the amount of ticks taken to finish each game diverges over the N trial, promoting opponents that have high diversity in their behaviour.

Spatial Diversity

The third, and final, criterion, quantifying the spatial diversity of the opponents, gives us (3.6):

$$H_n = \left[-\frac{1}{logV_n}\sum_i \frac{\nu_{in}}{V_n} log\left(\frac{\nu_{in}}{V_n}\right)\right]^{p_3} \tag{3.6}$$

for each game in the N trials, where V_n is the total number of visits for all cells, and p_3 is a weighting parameter.

Here, H_n is a measure of the entropy of cell visits for a single game, and given this, the metric for spatial diversity is the average value $avg(H_n)$ over the N trials. So the higher the average entropy of cell visits, the more interesting the opponents.

Metrics combined

Combining these three criteria gives us (3.7)

$$I = \frac{\gamma T + \delta S + \varepsilon * avg(H_n)}{\gamma + \delta + \varepsilon} \tag{3.7}$$

where γ, δ, ε are weighting parameters for each criterion.

3.3 Experiment

Also use online learning methods to improve (?).

The experimental method followed had three main steps: implementing the simulation platform and specification, generating promising candidate opponents for further evaluation, and evaluating the generated candidate opponents vs. pre-programmed opponents.

The simulation platform consisted of two parts: a simulation environment and a specification for use in ADATE. The simulation environment was an ADATE-ML implementation of the Dead End game, included in the specification, and an accompanying Python version for visualizing. The specification, in addition to the specification, contained an input generation function, an evaluation function for generated opponents, some support code needed by these, and various parameters to configure the ADATE system.

Insert time here

We then let the ADATE system run for a period of XXX to generate individuals. The generated individuals were examined and various individuals were picked out for further examination were picked out based on their interest metrics and some other factors.

To examine the chosen opponent individuals further we put them up against pre- programmed AI player characters, while monitoring their behaviour and performance in higher detail. This data was then collated and compared against the performance of a selection pre-programmed opponent behaviour.

3.4 Simulation platform

The simulation platform used in our experiment had two main parts: the simulation environment, the Dead End game as described in Section 3.1, and a specification for the ADATE system that will be used to generate opponents.

3.4.1 Simulation environment

When we implemented the Dead End game, our simulation environment, we made two separate implementations of it with slightly different goals. One was implemented in ADATE-ML to be used as the simulation platform in our specification; the other was implemented in Python using the Pygame framework (Pygame, 2012) to serve as a way to visualize the game, as well as to leave the option open for having humans play the game in place of AI if necessary.

As the ADATE-ML implementation was created with the sole goal of being used in the specification, it lacked any of the structure needed for it to resemble a real consumer game; therefore, it was not well suited for visualizing the platform, supporting alternative methods of play or collecting data other than what was needed to measure the performance of individuals.

To address this problem, and to give us a better platform to study the behaviour of generated individuals, the game was reimplemented in Python using the Pygame framework (Pygame, 2012). This implementation was done in a manner more resembling a consumer game, included facilities to allow humans to play the game in place of an AI player character, and contained much better facilities for gathering data about the performance of the characters in the game.

Include picture here

3.4.2 Specification

Interest function

Chapter 4

Results

Chapter 5

Discussion

Chapter 6

Conclusions

References

AAIP. Introduction. http://www.inductive-programming.org/intro.html, July 2010. Approaches and Applications of Inductive Programming.

Apple Software. Siri. http://www.apple.com/iphone/features/siri.html, October 2011.

C. Birch. Understanding pac-man ghost behavior. http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior, December 2010.

A. Blair and E. Sklar. Exploring evolutionary learning in a simulated hockey environment. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

N. Brickman and J. Nishant. htn planning and game state management in warcraft ii, Unknown year. http://users.soe.ucsc.edu/~nishant/CS244.pdf.

B. Buchanan. Brief history of artificial intelligence. In *American Association of Artificial Intelligence*, volume 26, 2002.

B. Buchanan, D. Smith, W. White, R. Gritter, E. Feigenbaum, J. Lederberg, and C. Djerassi. Applications of artificial intelligence for chemical inference. 22. automatic rule formation in mass spectrometry by means of the meta-dendral program. *Journal of the American Chemical Society*, 98(20):6168–6178, 1976.

A. Champandard. *AI game development: synthetic creatures with learning and reactive behaviors*. New Riders Pub, 2003.

M. Christian. a simple inference engine for a rule-based architecture. *AI. game. programming. wisdom*, pages 313–320, 2002.

N. Cole, S. Louis, and C. Miles. Using a genetic algorithm to tune first-person shooter bots. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 139–145. IEEE, 2004.

N. Crossley, E. Kitzelmann, M. Hofmann, and U. Schmid. Combining analytical and evolutionary inductive programming. In *Second Conference on Artificial General Intelligence*, pages 19–24. Citeseer, 2009.

P. Demasi and A. Cruz. Online coevolution for action games. *Int. J. Intell. Games & Simulation*, 2 (2):80–88, 2003.

T. Evans. A program for the solution of a class of geometric-analogy intelligence-test questions. Technical report, DTIC Document, 1964.

M. Federoff. *Heuristics and usability guidelines for the creation and evaluation of fun in video games*. PhD thesis, Citeseer, 2002.

J. Freeman-Hargis. Rule-based systems and identification trees. `http://ai-depot.com/Tutorial/RuleBased.html`, 2002.

P. Funes and J. Pollack. Measuring progress in coevolutionary competition. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, pages 450–459. MIT Press, 2000.

P. Funes, E. Sklar, H. Juillé, and J. Pollack. Animal-animat coevolution: Using the animal population as fitness function. *From Animals to Animats*, 5:525–533, 1998.

J. Funge. *Artificial intelligence for computer games: an introduction*. {Peters Corp.}, 2004.

M. Gallagher and A. Ryan. Learning to play pac-man: An evolutionary, rule-based approach. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2462–2469. IEEE, 2003.

S. Grand, D. Cliff, and A. Malhotra. Creatures: Artificial life autonomous software agents for home entertainment. In *Proceedings of the first international conference on Autonomous agents*, pages 22–29. ACM, 1997.

D. Isla and B. Blumberg. New challenges for character-based ai for games. In *AAAI Spring-Symposium on AI and Interactive Entertainment*, 2002.

A. Khoo, G. Dunham, N. Trienens, and S. Sood. Efficient, realistic npc control systems using behavior-based techniques. In *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, pages 46–51, 2002.

N. Kirby. *Introduction to game AI*. Course Technology PTR/Cengage Learning, 2011.

E. Kitzelmann. Inductive programming: A survey of program synthesis techniques. *Approaches and Applications of Inductive Programming*, pages 50–73, 2010.

R. Koster. *A theory of fun for game design*. Paraglyph press, 2004.

J. Koza. Genetic programming: On the programming of computers by means of natural selection, 1992. *See http://miriad. Iip6. fr/microbes Modeling Adaptive Multi-Agent Systems Inspired by Developmental Biology*, 229, 1992.

N. Lazzaro. Why we play games: Four keys to more emotion without story. In *Game Developers Conference*, 2004.

Lionhead Studios. Black & white. `http://lionhead.com/Games/BW/`, March 2001.

S. Lucas. Evolving a neural network location evaluator to play ms. pac-man. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 203–210. Citeseer, 2005.

G. Luger. *Artificial intelligence: Structures and strategies for complex problem solving*. Addison-Wesley Longman, 2005.

T. Malone. What makes computer games fun?(abstract only). In *Proceedings of the joint conference on Easier and more productive use of computer systems.(Part-II): Human interface and the user interface-Volume 1981*, page 143. ACM, 1981.

R. Mandryk, K. Inkpen, and T. Calvert. Using psychophysiological techniques to measure user experience with entertainment technologies. *Behaviour & Information Technology*, 25(2):141–158, 2006.

M. Mateas and A. Stern. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, pages 39–47, 2002.

Maxis. The sims. `http://thesims.ea.com/`, February 2000.

K. Merrick and M. Maher. Motivated reinforcement learning for non-player characters in persistent computer game worlds. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 3. ACM, 2006.

M. Minsky and S. Papert. *Perceptrons: An introduction to computational geometry*, volume 1988. MIT press Cambridge, MA, 1969.

T. Munakata. *Fundamentals of the new artificial intelligence: neural, evolutionary, fuzzy and more*. Springer-Verlag New York Inc, 2008.

Norwegian Ministry of Culture. Report no. 14 to the storting: Video games. `http://www.regjeringen.no/en/dep/kkd/Documents/regpubl/stmeld/2007-2008/report-no-14-2007-2008-to-the-storting.html`, March 2008.

J. R. Olsson. *Inductive functional programming using incremental program transformation and Execution of logic programs by iterative-deepening A* SLD-tree search*. PhD thesis, University of Oslo, Norway, 1994.

R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 74(1):55–81, 1995.

J. Orkin. Applying goal-oriented action planning to games. *AI Game Programming Wisdom*, 2: 217–228, 2003.

J. Orkin. Symbolic representation of game world state: Toward real-time planning in games. In *Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence*, 2004.

J. Orkin. Three states and a plan: the ai of fear. In *Game Developers Conference*, volume 2006. Citeseer, 2006.

J. Pittman. The pac-man dossier. gamasutra (2009). `http://home.comcast.net/~jpittman2/pacman/pacmandossier.html`, june 2011.

Pygame. About pygame. `http://www.pygame.org/wiki/about`, April 2012.

S. Rabin. Implementing a state machine language. *AI. game. programming. wisdom*, pages 314–320, 2002.

C. Ramos, J. Augusto, and D. Shapiro. Guest editors' introduction: Ambient intelligencethe next step for artificial intelligence. *IEEE Intelligent Systems*, pages 15–18, 2008.

J. Read, S. MacFarlane, and C. Casey. Endurability, engagement and expectations: Measuring children's fun. In *Interaction Design and Children*, pages 189–198. Citeseer, 2002.

J. Schaeffer and H. Van den Herik. Games, computers, and artificial intelligence. *Artificial Intelligence*, 134(1-2):1–8, 2002.

C. Shannon. Programming a computer for playing chess. *Philosophical magazine*, 41(7):256–275, 1950.

J. Slagle. A heuristic program that solves symbolic integration problems in freshman calculus. *Journal of the ACM (JACM)*, 10(4):507–520, 1963.

P. Spronck. *Adaptive game AI*. UPM, Universitaire Pers Maastricht, 2005.

P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Difficulty scaling of game ai. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37, 2004.

R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 116. Cambridge Univ Press, 1998.

P. Tozour. The evolution of game ai. *AI game programming wisdom*, pages 3–15, 2002.

A. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

G. Vattekar. Adate user manual. Technical report, Technical report, Ostfold University College, 2006.

J. Weizenbaum. Elizaa computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

P. WERBOS. Beyond regression: new fools for prediction and analysis in the behavioral sciences. *PhD thesis, Harvard University*, 1974.

Wikipedia. Backpropagation. `http://en.wikipedia.org/wiki/Backpropagation`, 2011a.

Wikipedia. Feedforward neural network. `http://en.wikipedia.org/wiki/Feedforward_neural_network`, 2011b.

S. Woodcock. Game ai: The state of the industry 2000-2001: Its not just art, its engineering. *Game Developer magazine*, 2001.

G. Yannakakis. Ai in computer games: Generating interesting interactive opponents by the use of evolutionary computation. *University of Edinburgh*, 2005.

G. Yannakakis. How to model and augment player satisfaction: A review. In *Proceedings of the 1st Workshop on Child, Computer and Interaction, Chania, Crete, ACM Press*, 2008.

G. Yannakakis and J. Hallam. Evolving opponents for interesting interactive computer games. *From animals to animats*, 8:499–508, 2004a.

G. Yannakakis and J. Hallam. Interactive opponents generate interesting games. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 240–247. Citeseer, 2004b.

G. Yannakakis and J. Hallam. Modeling and augmenting game entertainment through challenge and curiosity. *International Journal of Artificial Intelligence Tools*, 16(6):981–1000, 2007a.

G. Yannakakis and J. Hallam. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–972, 2007b.

G. Yannakakis and J. Hallam. Real-time game adaptation for optimizing player satisfaction. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(2):121–133, 2009.

G. Yannakakis, J. Hallam, and H. Lund. Entertainment capture through heart rate activity in physical interactive playgrounds. *User Modeling and User-Adapted Interaction*, 18(1):207–243, 2008.

List of Figures

List of Tables