Selected Topics regarding Automated Synthesis of NPCs

Daniel E. Bruce

October 18, 2011

Contents

Chapter 1

Introduction

This document covers a selection of the most relevant previous work related to my Thesis with the working title "Automated Synthesis of NPC AI with ADATE", on the topic of creating AI for game Non-Playing Characters with inductive programming, using the ADATE system.

There are two main areas that have to be covered, specifically the topics of Artificial Intelligence and Inductive Programming.

The work existing on the topic of Artificial Intelligence in games can be roughly divided between work describing various implementation techniques, and work describing how to make game AI more entertaining within the constraints of existing implementation techniques.

The line between these two subtopics is very fuzzy and gradual, so some judgment has been employed when putting the work into one category or the other, in some cases the choice is purely symbolical, such as when the work describes a new implementation technique created for the express purpose of increasing entertainment value (for example Khoo et al., 2002).

There is also some work on the more specific subtopic of using Evolutionary Computing techniques such as Inductive Programming to improve game AI, either for skill or entertainment value, which relates directly to the topic of my thesis.

Chapter 2

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 allowing automatic programming.

There is some differences between the techniques used in "traditional" AI and game AI, primarily based on the performance of the various methods and their focus on 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" to making computers seem more human, have an ability to learn and adapt, and maybe act like assistants, as detailed in Ramos et al. (2008).

When it comes to games (both video games and traditional games), the biggest difference in how to do the AI comes from whether or not the players have perfect information and whether there is chance involved. A perfect-information game will require a completely different approach to AI than a hidden-information or a stochastic game, and this will heavily influence what type of AI is utilized, as mentioned in Schaeffer and Van den Herik (2002, pg. 4-5).

2.1 Traditional AI techniques

As noted in Munakata (2008, chap. 1) there are six main areas of AI techniques, namely:

1. Symbolic AI (also "traditional AI")
2. Neural networks
3. Evolutionary Computing (under which Genetic Algorithms fall)

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.

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".

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 use of uncertainty. These methods are less relevant to the topic of my thesis, and as such will not be covered further.

2.1.1 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, and to the writings of many ancient scholars (among others, Aristotle and Euclid) on the topic of reasoning, logic and the mind. Further there have been many people working on mechanical automaton, which tried to emulate human behaviour.

There was early work on intelligent machines from the very point where there existed computers, 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 AI conference 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, aimed at the game of checkers and utilizing machine learning to improve its performance.

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

Later, there was work into "knowledge-based programs" for artificial reasoning, which created programs able to interpret mass spectra of chemical compounds, solve integration problems in math, and play chess good 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 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), which hosts many conferences and symposia each year, and supports several journals on AI.

Another such boom 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 events in games being played by AI, Deep Blue beat Garry Kasparov in 1997, and TD-Gammon was written, that played backgammon at championship-level. In addition, AI was starting to see use in cataloguing the internet.

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

2.1.2 Symbolic AI

This sub-field of AI 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 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.

2.1.3 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, or more correctly 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 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, among others Genetic Algorithms, Genetic Programming and Evolutionary Algorithms, but the common trait is that they use 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 by the person attempting to solve the problem, which measures the individual's performance at the problem, and the result of this is used to rank the individuals. We further choose one of the individuals in the list, and "reproduce" 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 also tested for fitness and put into the list, then the list is culled to a specific length by removing the least fit individuals. This process is repeated until an optimal solution is found, or the solution is deemed good enough.

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 Chapter 4.

2.2 Game AI techniques

The approach to AI used in video games (as opposed to more traditional board-/card games) is very different from the academic approach, 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.

In games the AI actors might only have a handful of microseconds of CPU time available to reach a decision lest they impact the performance of the game, and the AIs must fulfill the twin goals of being challenging (but beatable by the

majority of players) and entertaining (employ novel methods, and seem human-like).

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

2.2.1 History

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 which has a long history of strong scientific focus. The majority of this information comes from Tozour (2002).

When it comes to video game AI, the methods employed 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.

Continuing on through the article, it is explained that many of the popular early games used very crude AI, basically just a handful of simple rules, with the exception of games that just digitized board games with well established AI research, such as chess.

Sophisticated AI in video games was first embarked upon with turn-based strategy games (such as *Civilization*), then real times 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 is played.

After that the variety of different AI exploded, with games focusing entirely on watching AI "life" grow, with *SimCity* and *The Sims* being very well known staples, and *Creatures* which is famous for being one of the few games that actually uses 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).

2.2.2 Traditional game AI

Traditional game AI or "simple AI" is what game AI started out as, and still to this day mostly uses. 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 algorithms like 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 has little capability of 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 creates interesting behaviours and engaging game play which propelled Pac-Man to one of the most recognized games in video game history, evidence that you don't need fancy high-powered neural nets to create interesting games.

Finite State Machines

Finite State Machines (FSM) are the most used game AI technique, although the FSM 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, 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," as long as the programmer thinks of all the transitions necessary, and makes them seem natural.

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 creations, 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, 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 is 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" which is used by the system to infer which pieces of data apply to 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*, dealing mostly with information.

Usually there is some method of discerning which rules are the most relevant at the current time, 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), which 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 this is that you can create a good set of behaviour with a comparatively small amount of rules, so you aren't suspect to "state explosion" as 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).

2.2.3 Towards more human game AI

In recent times there has been efforts in making game NPC 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 NPC 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 taking into use planning and learning behaviour to make NPC seem smarter and more human-like. Games such as F.E.A.R (Using Goal-Oriented Action Planning as described in Orkin (2006)) use 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 AI of the NPC. Action sets contain actions which satisfy goals, and each action may have preconditions which in turn are goals. Using this information, the NPC can plan its action by what it knows of the world (which it can learn over time) and which actions it has defined, instead of having to have the AI programmers code down every possible such path.

An example of the benefit of using a planning system comes from Orkin (2006), where they describe an NPC which was working at a computer being shot. In the traditional FSM-based system, each state had to be exited fully before being able to transition, so when the NPC was shot, they would exit their "work" state by turning off their computer, pushing out their chair, and standing up, before promptly falling over dead. Not really what one would consider human-like, or even believable, 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 AI itself can also introduce emotions in the players by being very difficult or very easy, or other behaviours giving rise to player experiences such as "hard

fun", "soft fun" and "serious fun" (Lazzaro, 2004). 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 that uses the player's performance as a parameter in a fitness function to create better AI (one component of the thesis topic, and will be described in more detail later)

2.2.4 Advanced game AI

More advanced game AI utilizes more heavy-duty AI techniques such as Genetic programming and 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, as described earlier, 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, after the games' release, 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.

Chapter 3

NPC Entertainment value

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 in previous sections. These range from simple difficulty-scaling 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.

Link back?

3.1 Evaluating entertainment

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 through various scientific means in an objective manner.

3.1.1 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 costly, in equipment, time, or both. 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 series of equipment via scores of electrodes and wires, and is thus a lot more expensive, time consuming and intrusive. One way of doing this is described in Mandryk et al. (2006).

3.1.2 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, 2007, 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".

3.2 Making entertaining NPC

As noted in the section 3.1, 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 automatic programming or other forms of genetic algorithms. This method would utilize offline learning, and possibly online learning, to generate a selection of NPC behaviours which would create a more interesting experience. This method has been described by Yannakakis (2005), and is going to be described in more detail in Chapter 5.

Chapter 4

Inductive Programming

As noted in section 2.1.3, ADATE (Olsson, 1995) fits into a sub-field of Artificial Intelligence called Inductive Programming, which is actually a field recently emerging from being fragmented across Inductive Logic Programming, Genetic Programming, and Machine Learning (Kitzelmann, 2010).

4.1 Introduction to Inductive Programming

As described by the Approaches and Applications of Inductive Programming work group (AAIP, 2010), the aim of Inductive Programming is to create programs based on incomplete specifications. These specifications are given in the form of a set of inputs with matching outputs or an output evaluation function that are used to evaluate a generated program's fitness. This is opposed to Deductive approaches which start 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, although it requires the problem to be well-understood and a specification to be written, whereas the Inductive approach does not guarantee a fully correct solution, but only requires input and output data to create its solution.

To help guide it towards an acceptable solution the Inductive Programming system can also take other inputs in their specifications, in the form of constraints on efficiency and complexity; background information about data types, functions available for use in the generated code and information about the flow of the intended program; heuristics or bias to guide the system towards a certain solution. These are, however, not required.

As my thesis will be using the ADATE system, I will be explaining the details of Inductive Programming in context of the ADATE system, as described in

Olsson (1994) and Vattekar (2006).

4.2 The ADATE system

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 and the units which are manipulated (the "genes" if you will) are computer code operations. Apart from that, the process which is followed is very much like the one described in the previous section about evolutionary computing (section 2.1.3).

All Inductive Programming system generate code in a specific programming language, usually simplified or streamlined for the purpose, and ADATE is no exception. It uses a subset of Standard ML, called ADATE-ML, for its generated code, as well as for parts of its specification.

Fill out details about ADATE ML here

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, one in ADATE ML and the other in Standard ML, separated by a "%%" token.

The ADATE ML part contains three different things:

1. Definitions of data types and functions that can be used by the code that is generated by ADATE.
2. The initial program, called f, that the ADATE system will start from and improve, which takes the inputs and returns an output.
3. A function *main* which is called by the ADATE's evaluation cycle, and can provide a scaffolding or a context around f. This comes in handy if f is supposed to be a small part of a larger program, but 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.

The Standard ML part of the specification can contain any user supplied code that might be used as part of the evaluation, but must define a certain set of callbacks that are called by the ADATE system. These callbacks define things

such as the input data, any validation inputs, the set of functions that should be available for generated code, the evaluation function, the parameters governing amount and complexity of generated individuals, and other small details (consult Vattekar (2006, sect. 4.2) for details).

As ADATE is an evolutionary system, it uses the same general process as other genetic systems, being that of a cycle of procreation, evaluation and culling in the search for ever improving performance. These steps will now be described as they apply to ADATE.

The process of creating new individuals in ADATE functions like most other Evolutionary Computing solutions, in that it utilizes both mutation and genetic crossover (albeit only in a limited fashion). When ADATE creates a new individual it starts out with an individual from the "kingdom" existing individuals (which at the beginning is only one program), 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 it does this by replacing a part of the program with a new synthesized expression.

REQ is an R transformation that is guaranteed to not make the program 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 function by adding a new parameter to it, and updates all calls to the function to 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.

The individuals are then evaluated by running the evaluation function specified in the specification file on *main*, using the current individual as f, over all the examples given. The evaluation value from this function is then used to compute a few values regarding the individual's performance (among others the time it takes to run and the complexity of the program).

If the individual evaluates better than any of the solutions currently in the kingdom, the evaluation value is then used to group the individual 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 forever until the program is halted, and output is written to disk while the program runs. This output shows the current state of the system, and the current best individuals along with their code. By evaluating this output, the user of the system can decide if a good enough solution has been created, or if the system needs to be stopped for other reasons.

Chapter 5

Evolving Game AI

Chapter 6

Conclusion

Bibliography

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

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

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.

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

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.

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.

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.

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

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.

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.

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.

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.

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.

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. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–972, 2007.

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.