Autopia: An AI Collaborator for Gamified Live Coding Music Performances

Norah Lorway, Matthew Jarvis, Arthur Wilson, Edward J. Powley, and John A. Speakman

Abstract. Live coding is “the activity of writing (parts of) a program while it runs” [20]. One significant application of live coding is in algorithmic music, where the performer modifies the code generating the music in a live context. Utopia is a software tool for collaborative live coding performances, allowing several performers (each with their own laptop producing its own sound) to communicate and share code during a performance. We propose an AI bot, Autopia, which can participate in such performances, communicating with human performers through Utopia. This form of human-AI collaboration allows us to explore the implications of computational creativity from the perspective of live coding.

1 Background

1.1 Live coding

Live coding is the activity of manipulating, interacting and writing parts of a program whilst it runs [20]. Whilst live coding can be used in a variety of contexts, it is most commonly used to create improvised computer music and visual art.

The diversity of musical and artistic output achievable with live coding techniques has seen practitioners perform in many different settings, including jazz bars, festivals and algoraves - an event in which performers use algorithms to create both music and visuals that can be performed in the context of a rave. What began as a niche practice has evolved into an international community of artists, programmers, and researchers. With a rising interest in “creative coding”, live coding is well positioned to find more mainstream appeal.

At algoraves, the screen of each performer is publicly projected to create transparency between the performer and the audience. The Temporary Organisation for the Permanence of Live Algorithm Programming (TOPLAP) make it clear how important the publicity of the live coder’s screen is in their manifesto draft: “Obscurantism is dangerous. Show us your screens” [18].

A central concern when performing live electronic music is how to present “liveness” to the audience. The public screening of the performer’s code at an algorave is often discussed in regards to this dynamic between the performer and audience, where the level of risk involved in the performance is made explicit. However, in the context of the system proposed in this paper, we are more concerned with the effect that this has on the performer themselves. Any performer at an algorave must be prepared to share their code publicly, which inherently encourages a mindset of collaboration and communal learning with live coders.

1.2 Collaborative live coding

Collaborative live coding takes its roots from laptop orchestra/ensemble such as the Princeton Laptop Orchestra (PLOrk), an ensemble of computer based instruments formed at Princeton University [19]. The orchestra is a part of the music research community at the University and is concerned with investigating ways in which the computer can be integrated into conventional musical making. PLOrk attempts to radically transform those ideals [19]. Each PLOrk meta instrument consists of a laptop, multi-channel hemispherical speaker and a variety of control devices such as game controllers, sensors amongst others [19]. The orchestra consists of 12-15 students and staff ranging from musicians, computer scientists, engineers and others and uses a combination of wireless networking and video in order to augment the role of the conductor [19].

UK based live coding ensembles such as the Birmingham Ensemble for Electroacoustic Research (BEER) based at the University of Birmingham have taken influence from ensembles such as PLOrk, but differ in terms of the way they integrate communication and collaboration within the ensemble. The ensemble was formed in 2011 by Scott Wilson and Norah Lorway [22] and began as an “exploration of the potential of networked music system” for structured improvisation[22]. The ensemble works primarily in the SuperCollider (SC) language4 and the JITLib (Just In Time Library)5 classes in SC for basic live coding functionality [22]. In terms of ensemble communication and coordination, BEER uses Utopia (Wilson et al 2013), a SuperCollider library for the creation of networked music application which builds on the Republic quark6 and other such networked performance systems in SuperCollider. Networked collaboration in live coding was present from the inception of live coding where multiple machines are clock-synchronized exchanging TCP/IP network

---

1 Academy of Music and Theatre Arts, Falmouth University, UK. Email: norah.lorway@falmouth.ac.uk, MJ196235@falmouth.ac.uk, AW193362@falmouth.ac.uk
2 Games Academy, Falmouth University, UK. Email: edward.powley@falmouth.ac.uk, john.andrew.speakman@falmouth.ac.uk
3 https://github.com/muellmusik/Utopia
4 https://github.com/supercollider/supercollider
5 http://doc.sccode.org/Overviews/JITLib.html
6 https://github.com/supercollider-quarks/Republic
messages [5]. Utopia aims to provide a more modular approach to networked collaboration, featuring enhanced flexibility and security over other existing solutions. It also provides an efficient way to synchronize communication, code and data sharing over a local network. Unlike an ensemble such as PLOrk which uses a human conductor such as in a traditional orchestra, Utopia eliminates the need for this, allowing for a more streamlined shared approach, where performers collectively make musical decisions.

2 Motivation

2.1 Computational creativity

Using an AI bot within the context of a networked live coding performance, is an idea that builds on a study undertaken by McLean and Wiggins [12], regarding live coding towards Computational Creativity.

Computational Creativity can be described as the aim of “endowing machines with creative behaviours” [15], and systems designed to do so can be put to practical uses from simulating and automating existing human processes (creativity as it is), to discovering novel outcomes (creativity as it could be) [15], which could be valuable to the “scientific study of creativity” [21]. In the context of this proposal, we are concerned with the latter.

The McLean and Wiggins study [12], highlighted a view among live coding practitioners that the code resulting from their practice contains an element of the programmers style, and that “many feel they are not encoding a particular piece, but how to make pieces in their own particular manner” [12]. This is a sentiment that is echoed by Wiggins and Forth [21] in the following statement:

“...In a manner akin to the extended-mind theory of consciousness [3], the live coder becomes attuned to thinking with and through the medium of code and musical abstractions, such that the software can be understood as becoming part of the live coder’s cognition and creativity” [21].

Through a process of “reflexive interaction” [21], the human performer(s) and artificial agent each influence the actions of the other. Entering into a “complex feedback loop” [8], the artificial agent becomes an “imperfect mirror” of the human performer(s) [21]. We propose that through the analysis of the artificial agent’s behaviours, we can extend our understanding of what constitutes “valuable” musical output, while challenging existing dogmatic approaches to live coding practice, and techniques relating to the chosen programming language (SuperCollider), where the formalisation and subsequent manipulation of syntax trees can provide new insight to the language’s potential. Finally, it can provide insight into the nature of creativity in general, by analysing emergent behaviour from the bot.

Ultimately, our motivation can be summarised in the following quote: “When the computer becomes a conversation partner, or a boat rocking us in unexpected directions, we may find that the technologies we build become more useful, more musical, more interesting than our original conceptions” [8].

2.2 Gamification

There has been work on the use of gamification to facilitate creativity [9]. This generally draws upon the idea of flow [7] — the idea being that flow is important to creativity, and that including some game-like elements in a creative software or process can help to put users into this flow state. Taken further, this leads to the idea of casual creators [6] — creative tools whose interface is designed to promote a “playful, powerful, and pleasurable” user experience (unlike more traditional creative software where “powerful” would take precedence over the other two). Aiming for playfulness in this context can also promote curiosity and experimentation [13].

Gamification has also been studied in the context of collective creativity [16]. There are obvious analogies between collaborating on creative tasks and playing a multiplayer game, and the ideas used in the latter to foster collaboration (or, in some cases, competition) may prove useful in the former. For instance, the Female Interface Research Ensemble (FIRE) based at the University of Birmingham, used Utopia and gamified collaborative approaches in their algorave performance during The New Interfaces for Musical Expression conference in 2014 in London, UK [11]. As another example, Nilson [14] proposes a number of game-like exercises, many of them collaborative and/or competitive, to be used by live coders in a practice context.

We propose taking a gamified collaborative creative environment and adding a “bot” — an AI agent which interacts in the same way as a human would. Bots in multiplayer games are often used as sparring partners for offline practice matches, or to make up the numbers when not enough human players are available for a game, however the fact that the play style of bots is different to that of humans tends to change the dynamics of the game. We are interested in studying whether the same is true for a collaborative live coding performance — how does the introduction of one or more bot performers change the dynamics of the performance?

3 The bot

In order to truly participate in the performance in the same way as a human performer, the bot must carry out two AI tasks: participating in conversation through the Utopia chat interface, and generating and running SuperCollider code. For the former we will draw on well-established chatbot technology; for the latter we will use genetic programming (GP) [10]. SuperCollider code generally makes heavy use of nested function calls and mathematical expressions, often involving several numerical constants that can be tuned, and so we hypothesise that the language lends itself well to a GP approach.

The bot will implement the Template-Based Object-Oriented Genetic-Programming algorithm [17] in CSharp, set to automatically construct SuperCollider code from a series of pre-defined templates. These templates, are built using a genetic sequence, which is used to select the initial template, usually a single line of SuperCollider code which has been broken into its constituent parts, as strings. The variables used in these templates are filled in as values read directly from the genetic algorithm or as variables created at an earlier point in the automatic construction of the code.

This occurs in 3 phases: an initialization phase, which generates a series of initial sine waves, a modification phase which alters those waves and an execution phase which plays the generated sounds. Each of these phases corresponds to its own library of templates. The generated code can then be re-
ried using JavaScript, at which point it may be inserted into, and executed by SuperCollider.

Code can be generated in a batch and bred together, representing a generation. A call can be made which takes two agents (genetic sequences which may be used to generate SuperCollider code) and breed them together using a simple genetic crossover algorithm to produce a new, offspring agent. Using this technique, multiple generations of agents may be generated which can be used, with selection, to breed against a fitness function.

The GP algorithm will run continuously, and at each generation the fittest individual will be executed through SuperCollider. When other (human) performers execute code and it is shared through Utopia, the GP system will add the code to its own population, to introduce variety to the gene pool and allow Autopia to build upon what the other performers are doing. In the spirit of live coding performers sharing their code, as discussed in Section 1.1, the bot’s screen (showing the code it is evaluating and executing) will be projected so that the audience can see it.

Any evolutionary computing approach requires a fitness evaluation function. We propose to evaluate the fitness of individuals in the population through a basic machine listening process: individuals will be run through a second instance of SuperCollider, and the system will perform a frequency analysis (i.e. Fourier transform) on the resulting audio output. This will be compared to a frequency analysis of the audio output being produced by the other performers. The more similarity in frequency characteristics between the two, the higher the fitness. As a first step this should at least weed out those population members which produce undesirable results (such as silence or white noise), though clearly the refinement of the fitness measure is a fruitful line of future work. Collins [4] suggests a number of more sophisticated machine listening approaches which may prove useful, and provides a JavaScript library implementing several of these techniques.

To introduce an aspect of gamification and to further enhance the GP system’s fitness evaluation, we will add a voting-based points system to Utopia. A similar idea to this was already tested in Republic. This will allow participants (both humans and bots) to vote each other up and down, giving them feedback on their contributions (and for the bot, explicitly shifting the fitness evaluation towards the preferences of the other performers).

4 Conclusions

Using AI in the context of live coding is relatively new and unexplored. The idea of AI collaborators has been well explored in Computational Creativity, including in musical contexts, however the process used by the AI can sometimes be opaque to observers and is almost certainly quite different to the process used by human performers. By combining AI with live coding we hope to overcome this — humans and bots are participating at the same level and in the same way (i.e. by manipulating code) — bringing the human-AI ensemble closer to liveness. This also goes towards achieving the goal, set out by the Birmingham Laptop Ensemble [2] in their manifesto, of “integration, collaboration and the blurring of the distinctions between, composer-performer-collaborator in a democratic non-authoritarian ensemble” [1].

The state of flow is clearly desirable in creative activities. The use of gamification can potentially be a powerful way of getting participants into this flow state, as well as the idea of voting borrowed from multiplayer games helping to facilitate the goals described above. The effect of introducing a bot performer on the human performers’ flow state is less easy to predict — our hope is that the bot will act as a “conversation partner” [8] and thus provide inspiration during a performance.

REFERENCES


https://github.com/sicklincoln/mml


Evolving Self-taught Neural Networks: The Baldwin Effect and the Emergence of Intelligence

Nam Le

Abstract. The so-called Baldwin Effect generally says how learning, as a form of ontogenetic adaptation, can influence the process of phylogenetic adaptation, or evolution. This idea has also been taken into computation in which evolution and learning are used as computational metaphors, including evolving neural networks. This paper presents a technique called evolving self-taught neural networks – neural networks that can teach themselves without external supervision or reward. The self-taught neural network is intrinsically motivated. Moreover, the self-taught neural network is the product of the interplay between evolution and learning. We simulate a multi-agent system in which neural networks are used to control autonomous agents. These agents have to forage for resources and compete for their own survival. Experimental results show that the interaction between evolution and the ability to teach oneself in self-taught neural networks outperform evolution and self-teaching alone. More specifically, the emergence of an intelligent foraging strategy is also demonstrated through that interaction. Indications for future work on evolving neural networks are also presented.

1 Introduction

Evolution and learning are two forms of adaptation. The former is a change at genotypic level of a population, also called phylogenetic adaptation. The latter is a change at phenotypic level of an individual as a result of experience with its environment during lifetime. Thus, learning is a form of lifetime or onntogenetic adaptation. Reasonably, lifetime adaptation takes place at a quicker pace than evolution, preparing the organism for increasingly uncertain environments which may require some survival skill that the slower evolutionary process could not fully offer.

Interestingly, evolution and learning can complement each other through the phenomenon called the Baldwin Effect [1], [14], which was first demonstrated computationally by Hinton and Nowlan (henceforth H&N) [5]. Following this success, there have been quite a few important studies studying the interaction between learning and evolution, in evolutionary dynamic optimisation [9], notably in evolving neural networks [15], and in the NK-Landscape [11]. Regardless of the problem domain and how learning is implemented, most studies focus on how learning and evolution are combined to solve an individual problem in a sort-of single-agent environment. This means each agent has its own problem (though they are copies of each other) to solve. There is no interactive effect between the agents and their solutions to each other. This differs greatly in the case of a multi-agent environment in which agents live in the same environment and may have to compete and cooperate in solving their own problems or problems shared with others.

Although there should possibly be a mixture of flavour, this paper aims at two main things. First, we present a technique called evolving self-taught neural networks, or neural networks that can teach themselves without external supervisory signals. This is an important aspect of this contribution. Second, we simulate a multi-agent foraging world to test the performance of our proposed method and see the effect of interest. More specifically, we shall be seeing how evolution and the ability of self-teaching interact with each other in creating more adaptive and autonomous foraging agents, those that have little knowledge about the world.

In the remainder of this paper, we initially present some prior research relating to the Baldwin Effect, including some review on learning and evolution in neural networks. We then describe the detail of the neural network and the simulation undertaken. The results from these experiments are analysed and discussed, and finally, conclusions and several interesting future research opportunities are proposed.

2 Related Work

2.1 The Baldwin Effect

In nature, the organism with learning ability may be able to learn some new skill or knowledge to adapt as the environment becomes harder or unpredictable that what evolution has provided is not sufficient to survive. The Baldwin Effect is often understood as, over generations, that skill or knowledge becomes innate or closer to be innate so that the future organism can quickly adapt to the environment with fewer or even without any learning effort undertaken [17]. This shows how learning, or lifetime adaptation, can influence the evolutionary pathway of a species.

The idea that learning can influence evolution in Darwinian framework was discussed by psychologists and evolutionary biologists over one hundred years ago through ‘A new factor in evolution’ [1], [17]. However, it gradually gained more attention since the classic paper in 1987 by the British Cognitive Scientist Geoffrey Hinton and his colleague Steven Nowlan at CMU ([5]). Hinton and Nowlan (henceforth H&N) demonstrated an instance of the Baldwin effect in a computer simulation. They used a Genetic Algorithm to evolve a population in a Needle-in-a-haystack landscape showing that learning can help evolution to search for a solution when evolutionary search alone is ineffective. Through the Baldwin-like effect in H&N’s simulation, the correct behaviour (solution) can gradually emerge by the interaction between learning and evolution, but cannot happen by both learning or evolution alone [5].
The model developed by Hinton and Nowlan, though simple, is interesting, opening up the trend followed by a number of research papers investigating the interaction between learning and evolution. Following the framework of Hinton and Nowlan, there have been a number of other papers studying the Baldwin effect in the NK-fitness landscape – a ‘tunably rugged’ fitness landscape. Problems within that kind of landscape are shown to fall in NP-completeness category. Several notable studies of the Baldwin effect in the NK-model include work by Giles Mayley [11] in which the Baldwin Effect was shown to occur as learning can guide evolution to cope with the rugged fitness landscape.

2.2 Learning and Evolution in Neural Networks

Following the work of Hinton and Nowlan [5], There have also been several studies on the topic of learning and evolution in Neural Networks. Notable studies include [6] in which the authors used a genetic algorithm to evolve the initial weights of a digit classifier neural network which then can be learned by backpropagation. They found that if the amount of learning is used properly, learning can take advantage of starting weights produced by evolution to further the classification performance.

Todd and Miller [20] proposed an imaginary underwater environment in which each agent in one of the two feeding patches, and has to decide whether to consume substances floating by, without any feedback given to an individual agent that could be used to discriminate between food and poison. Each agent uses its neural network to associate the colour (red or green) and the substance (food or poison). Hebbian learning [4] in combination with evolution was shown to do better than both evolution and learning alone in this scenario.

Nolfi and his colleagues made a simulation of animats, or robots, controlled by neural networks situated in a grid-world, with discrete state and action spaces [15]. Each agent lives in its own copy of the world, hence no mutual interaction. The evolutionary task is to evolve action strategies to collect food effectively, while each agent learns to predict the sensory inputs to neural networks for each time step. Learning was implemented using backpropagation based on the error between the actual and the predicted sensory inputs to update the weights of a neural network. It was shown that learning to predict can enhance the evolutionary search, hence increasing the performance of the robot.

Generally learning in neural networks can be thought of as part of neural plasticity. There have been some other ideas, like evolving local learning rules to update the weights [2], evolution of neuromodulation which facilitates the information transfer between neurons in hopes of creating meta-learning [3]. Please refer to [18] for more recent studies on evolving plastic neural networks. In short, most of the work use disembodied and unsituated neural networks in single-agent environment, having no mutual interaction as they solve their own problems, having no effect on other’s performance.

In this paper, we propose a neural architecture called self-taught neural networks – neural networks that can teach themselves without an external teacher or reward. This differs greatly from traditional supervised learning in which a learning machine is provided with labels served as the external teacher. This technique also differs from reinforcement learning in which a learning agent has a reward provided by its external environment. Indeed, the agent controlled by the self-taught network can perform learning on its own without external reward. This type of network can be considered sort-of intrinsically motivated, hence the agent controlled by the network. Moreover, the self-taught network can both learn (self-teaching) and evolve. We shall be seeing how learning and evolution interact with each other in producing self-taught neural networks in later sections.

Second, we simulate a situated multi-agent system – a system containing multiple situated agents living together and doing their tasks while competing with each other. Each agent is controlled by a neural network but situated (and has a soft-embodiment). This means, the way an agent acts and moves in the world affects the subsequent sensory inputs, hence the future behaviour of that agent. Our simulations are described in the following section.

3 Simulation setup

This section describes the detail of the simulated world containing foods and agents as well as the neural network architecture used to control the agent moving and foraging in the world.

3.1 The Simulated World of Foods and Agents

Suppose that 20 agents situate in a continuous 640x640 2D-world, called MiniWorld, and they have to find resources to feed themselves to survive. There are 50 food particles in the world. Each food particle is represented by a square image with size 10x10. Each agent in MiniWorld also has a squared body of size 10x10. Agents and foods are initially located at separate regions in MiniWorld depending on the world map. We use two world maps (map A, and map B) in our simulations as described as follows:

Let’s denote width and height the width and the height of MiniWorld. Initially, in both maps, all agents are located around the vicinity of radius 40 (4 times the size of an agent) around the point (width/4, height/4) (the central point of the top left quarter, as shown in Figure 1).

Foods in map A have horizontal and vertical dimensions randomly chosen in ranges (width ∈ 5/8, width ∈ 7/8) and (height ∈ 1/8, height ∈ 3/8), respectively. The food region in map A is the square that has the same central point as the top right quarter, and each side of that square has the length of width/4. In map B, the food has its horizontal and vertical dimensions randomly chosen in ranges (width ∈ 5/8, width ∈ 7/8) and (height ∈ 5/8, height ∈ 7/8), accordingly. The food region in map B is the square that has the same center as the bottom right quarter, and each side of that square has the length of width/4. Two world maps are visualised in Figure 1 (Please note that dim green lines are sketched only for the purpose of visualising the world map of foods and agents, MiniWorld is a continuous world, not grid-like). Through the visualisation, it can be temporarily seen that map B is likely to be more difficult than map A since the food source is further to reach.

Initially agents is located far from the food source so that they have to forage to find the food source, to feed themselves. When an agent’s body happens to collide with a food particle, the food particle is eaten, the energy level of the agent increases by 1, and another food piece randomly spawn in the same region but at a different location. The collision detection criterion is specified by the distance between the two bodies (of the agent and of the food). The agent body somehow affects how the agent senses and acts in MiniWorld. By the re-appearance of food, the environment changes as an agent eats a food.

One property of MiniWorld is it has no strict boundary, and we implement the so-called toroidal – this means when an agent moves beyond an edge, it appears in the opposite edge.

Each agent has a heading (in principle) of movement in the environment. Rather than initialising all agents with random headings, to
make it more controllable, all the agents are initialised with a horizontal heading (i.e. with 0 degree). This somewhat explains the purpose of the design of map A and map B. In map A, all agents are initially born with a tendency to move forwards the food source. On the contrary, the agent in map B is born with a wrong direction to the food source. This clearly shows that map B is more difficult than map A. Agents in map B should have to acquire correct foraging behaviour to find the food source first, not to say they have to compete with each other for the energy. This is to say, agents in map B should develop a form of intelligent foraging to effectively seek for resources.

In our simulation, we assume that every agent has a priori ability to sense the angle between its current heading and the food if appearing in its visual range. The visual range of each agent is a circle with radius 4. Each agent takes as inputs three sensory information, which can be the binary value 0 or 1, about what it sees from the left, front, and right in its visual range. If there is no food appearing in its visual range, the sensory inputs are all set to 0. If there is food appearing on the left (front, or right), the left (front, or right) sensor is set to 1; otherwise, the sensor is 0.

Let $\theta$ (in degree) be the angle between the agent and the food particle in its visual sense. An agent determines whether a food appears in its left, front, or right location in its visual range be the following rule:

\[
\begin{align*}
15 < \theta < 45 & \Rightarrow right \\
\theta \leq 15 & \text{ or } \theta \geq 345 \Rightarrow front \\
315 < \theta < 345 & \Rightarrow left
\end{align*}
\]

We let all agents live in the same MiniWorld. They feed for their own survival during their life. The more an agent eats, the less the chance for others to feed themselves. This creates a stronger competition in the population. When an agent moves for foraging, it changes the environment in which other agents live, changing how others sense the world as well. This forms a more complex dynamics, even in simple scenario we are investigating in this paper.

The default velocity (or speed) for each agent is 1. Every agent has three basic movements: Turn left by 9 degrees and move, move forward by double speed, turn right by 9 degrees and move. For simplicity, these rules are pre-defined by the system designer of MiniWorld. We can imagine the perfect scenario like if an agent sees a food in front, it doubles the speed and move forward to catch the food. If the agent sees the food on the left (right), it would like to turn to the left (right) and move forward to the food particle. The motor action of an agent is guided by its neural network as described below.

### 3.2 The neural network controller

Each agent is controlled by a fully-connected neural network to determine its movements in the environment. What an agent decides to do changes the world the agent lives in, changing the next sensory information it receives, hence the next behaviour. This forms a sensory-motor dynamics and a neural network acts as a situated cognitive module having the role to guide an agent to behave adaptively, or Situated Cognition even in such a simple case like what is presenting in this paper. Each neural network includes 3 layers with 3 input nodes in input layer, 10 nodes in hidden layer, and 3 nodes in output layers.

The first layer takes as input what an agent senses from the environment in its visual range (described above). The output layer produces three values as a motor-guidance for how an agent should behave in the world after processing sensory information. The maximum value amongst these three values is chosen as a motor action as whether an agent should turn left, right, or move forward (as described above). All neurons except the inputs use a sigmoidal activation function. All connections (or synaptic

---

**Figure 1.** MiniWorld – The environment of agents and food, 640x640.

**Figure 2.** Agent situated in an environment seeing food

**Figure 3.** Neural network controller for each situated agent. Connection weights can be created by evolutionary process, but can also be changed during the lifetime of an agent.
In the following sections, we describe simulations we use to investigate the evolutionary consequence of lifetime learning.

### 3.4 Simulation 1: Evolution alone (EVO)

In this simulation, we evolving a population of agents without learning ability. The neural network controller for each agent is the one described in Figure 3. The genotype of each agent is the weight matrix of its neural network, and the evolutionary process takes place as we evolve a population of weights, a common approach in Neuroevolution (NE) [21].

Selection chooses individuals based on the number of food eaten in the foraging task employed as the fitness value. The higher the number of food eaten, the higher the fitness value. For crossover, two individuals are selected as parents, namely parent1 and parent2. The two selected individuals produce one offspring, called child. We implement crossover as the more the success of a parent, the more the chance its weights are copied to the child. The weight matrices of the child can be simply described as the algorithm below.

**Algorithm 1 Crossover**

```plaintext
1: function CROSSOVER(parent1, parent2)
2:   rate = parent1.fitness(parent2.fitness)
3:   child.weights = copy(parent2.weights)
4:   for i in range(len(child.weights)) do
5:     if random() < rate then
6:       child.weights[i] = parent1.weights[i]
7:     end if
8:   end for
9: end function
```

Once a child has been created, that child will be mutated based on a predefined mutation rate. In our work, mutation rate is set to 0.05. A random number is generated, if that number is less than mutation rate, mutation occurs, and vice versa. If mutation occurs for each weight in the child, that weight is added by a random number from the range [-0.05, 0.05], a slight mutation. After that, the newly born individual is placed in a new population. This process is repeated until the new population is filled 100 new individual agents. No elitism is employed in our evolutionary algorithm.

The population goes through a total of 100 generations, with 5000 time steps per generation. At each time step, an agent does the following activities: Perceiving MiniWorld through its sensors, computing its motor outputs from its sensory outputs, moving in the environment which then updates its new heading and location. In evolution alone simulation, the agent cannot perform any kind of learning during its lifetime. After that, the population undergoes selection and reproduction processes.

### 3.5 Simulation 2: Evolution of Self-taught agents (EVO+Self-taught)

In this simulation, we allow lifetime learning, in addition to the evolutionary algorithm, to update the weights of neural network controllers when agents interact with the environment. We evolve a population of **Self-taught** agents – agents that can teach themselves. The self-taught agent has a self-taught neural network architecture as described previously and shown in Figure 4. During the lifetime of an agent, the reinforcement modules produce outputs in order to guide the weight-updating process of the action module. Only the weights of action modules can be changed by learning, the weights of reinforcement module are genetically specified in the same evolutionary process as specified above in Evolution alone simulation.
We can interpret this scenario as an agent has an ability to produce reinforcement signals to guide itself. It is evolution that produces these reinforcement signals, or the desire to external stimuli (the sensory inputs in this case), for every agent. In other words, it is evolution that provides the self-teaching ability for each agent. This is how evolution influences learning. And more than this, it is learning during the lifetime that changes the fitness of each agent, hence the fitness landscape which then affects the evolutionary process. This is the interaction between learning and evolution which is being investigated in this paper.

We use the same parameter setting for evolution as in EVO simulation above. At each time step, an agent does the following activities: Perceiving MiniWorld through its sensors, computing its motor outputs from its sensory outputs, moving in the environment which then updates its new heading and location, and updating the weights in action module by self-teaching. After one step, the agent updates its fitness by the number of food eaten. After that, the population undergoes selection and reproduction processes as in Evolution alone.

Remember that we are fitting learning and evolution in a Darwinian framework, not Lamarckian. This means what will be learned during the lifetime of an agent (the weights in action module) is not passed down onto the offspring.

### 3.6 Simulation 3: Self-taught agents alone (Self-taught-alone)

We conduct another simulation in which all agents are self-taught agents – having self-taught networks that can teach themselves during lifetime. What differs from simulation 2 is that at the beginning of every generation, all weights are randomly initialised, rather than updated by an evolutionary algorithm like in simulation 1. The learning agents here are initialised as blank-slates, or tabula rasa, having no predisposition to learn or some sort of priori knowledge about the world. The reason for this simulation is that we are curious whether evolution brings any benefit to learning in MiniWorld. In other words, we would like to see if there is a synergy between evolution and learning, not just how learning can affect evolution.

Experimental results are presented and discussed in the following section.

### 4 Results and Analysis

#### 4.1 Learning Facilitates Evolution

First we look at the performance of the first two simulations, EVO and EVO+Self-taught. All results are averaged over 30 independent runs.

Figure 5 depicts the dynamics of fitness over generations, while Figure 6 presents a statistical comparison of the best and average fitness over runs. A similar trend can be observed is that all experimental settings have higher performance in map A than in map B. This is understandable as map B has been shown more difficult to forage than map A.

How each type of experimental setup performs compared to each other? First we look at the dynamics of the number of food eaten. It can be seen that EVO+Self-taught outperforms EVO alone in all maps with respect to both the best and average fitness. Specifically, by looking at the performance on map A we see that the best agent in EVO+Self-taught, on average, eats around 40 food particles more than the best agent in EVO alone. Additionally, the average agent in EVO+Self-taught eats around 40 food items more than the average agent in EVO alone. This means, as a whole, the EVO+Self-taught system has around 800 energy (each food item accounts for 1 energy) higher than the EVO system alone.

Looking at the performance on map B, it is interesting to see that while the EVO+Self-taught system still can forage for foods, the EVO system cannot eat any food at all.

Please recall the description of our learning agents as well as the map B. Every learning agent is born with an initial horizontal heading that may be changed when the agent experiences the world through its senses and motors. The more the agent encounters, the more likely the agent can change its subsequent movements, hence its heading. However, in map B the food source is located far from the agent, at first, and far from the initial heading of every agent. This means that each agent with its innate ability and horizontal heading cannot move along the correct direction to the food source. After being born, they move horizontally as designed. Importantly, because of the inability to learn to change the behaviour during lifetime, every agent in EVO moves based on its innate behaviour. This explains why the EVO alone system cannot forage and eat food.

Conversely, the self-taught agent can still eat food in map B. One plausible explanation for this is the effect of learning through self-teaching on evolution as follows. Like in EVO alone, every self-taught agent is initially born with a wrong direction to the food source. However, with the ability to teach oneself by leveraging the difference between the action and the reinforcement modules, the weights of the action module of some agent may have been changed during lifetime by backpropagation algorithm. It is this process that may change the movement of some agent, make it more random at first (like performing a random search in the movement space, rather than going in one direction). By doing some random movement, there may have been some agent that somehow could reach the food source (e.g. by any kind of luck). Because of this, the agent...
that can reach the food source has a higher chance of being selected to produce offspring the for next generation. Thus, its genetic information is more likely to proliferate. It is important to note that the genetic information of each self-taught learning agent consists of not only the initial weights for the action module but also the initial weights for the reinforcement module. Thus, when an agent is selected for reproduction, its self-teaching ability is likely to be also promoted at later generations.

The boxplots in Figure 6 present some statistical results on the best and the average fitness over 30 runs. We can easily see the same effect as presented above in both map A and map B. The advantage of EVO+Self-taught over EVO alone is statistically significant.

We can claim that he combination of learning in the form of self-teaching and evolution increases the adaptivity of the population measured by the number of food eaten in any case.

### 4.2 Is That the Baldwin Effect?

We have seen how learning during lifetime facilitates the evolving population of self-taught agents, having higher performance in a multi-agent environment compared to EVO alone. One curious question here is whether the Baldwin-like Effect has occurred?

This is why we conduct the third simulation in which the neural networks of self-taught agents are all randomly initialised, without the participation of evolution. It can be observed in Figure 5 and Figure 6 that in both map A and B, the population of randomly self-taught agents has lower performance than that of EVO+Self-taught, especially when it comes to the performance of the whole population (average fitness in our scenario). The difference is statistically significant as shown in Figure 6.

It is also interesting that in our simulation, the blank-slate population by self-teaching cannot outperform the EVO alone in the easier map A, but has little advantage over the EVO alone in the harder case (map B) when the EVO alone cannot search for any food.

It is plausible here to conclude that learning, as a faster adaptation, can provide more adaptive advantage than the slower evolutionary process when the environment is dynamic like in MiniWorld. However, it is evolution that provides a good base for self-taught agents to learn better adaptive behaviours in future generations rather than learning as blank-slates in Random-Self-taught population. This can also be explained by the understanding of the Baldwin Effect, or the synergy between evolution and learning. Through the evolutionary process, some priori-knowledge about the environment can be encoded in the neural networks controlling agents. Agents having priori-knowledge, or predisposition to learn adaptive behaviours in our scenario, can learn faster and learn more adaptively than blank-slate agents. This is the Baldwin-like Effect – the interplay between learning and evolution.

### 4.3 The Emergence of Intelligent Foraging

Interestingly, it is not just the performance but also the emergence of foraging behaviour – what can be called intelligent in this sense.

Figure 7 depicts the emergence of an intelligent foraging behaviour over time. Due to scope of this paper we only report the emergence on the harder map, where only one system – the EVO+Self-taught has shown a dominant performance. As we can see in the first two images in Figure 7, at earlier generations the population cannot find the way to reach the food source. Some might have moved in some random orientation rather than just following the horizontal direction. In the two following images, we can see that after several generations some agents appeared to find a way to reach the food source, while the rest was still unable to forage correctly, moving randomly but getting better.

In the second-last image, we observe that most agents have found the way to reach the food source except for one agent. However, the last image shows the whole system could reach the food source. They stayed there and competed for resources. All agents seem to know where to forage as a whole. Remember that, every agent in our MiniWorld does not have any idea about the location of the map (very simple sensory inputs) as well as the location of other agents. However, at the end of the day they still can reach the food source. This intelligence is the emergence through the interaction between evolution and self-teaching in the evolution of their brains, or neural networks.

### 5 Conclusion and Future Work

In this paper, we have presented a technique called Evolving Self-taught Neural Networks, and simulated a foraging task in a multi-agent system. Experimental results have shown that the proposed technique which combines evolutionary search and self-teaching in neural networks can enhance the system, better than evolution and self-teaching staying in isolation. An intelligent foraging behaviour is shown to emerge from the interaction between evolution and self-teaching. Self-teaching ability can help an agent better adapt to its environment, changing the subsequent evolutionary pathway of a species. Evolution is shown to provide more adaptive self-taught agents in future generations, better than learning as blank-slates.
There are quite a few avenues for future research building on this study. We can complexify MiniWorld by including more objects like obstacles, more substances with negative rewards (poison) to make the learning task more complex, hence the intelligence required. We are curious to see how the evolved self-teaching ability can better promote the system in more complex environments.

The computational method is simple enough to illustrate the idea, but still has some indications. The idea of self-taught neural networks can be powerful when there is no external supervision (or label provided from external data). This opens a way to produce autonomous intelligence, which might open a route to AGI – Artificial General Intelligence. The algorithm and technique used in this paper can also be a potential technique to solve unsupervised learning, or learning with limited label data (weak supervision, especially in reinforcement learning and games. We are curious whether evolution can provide a better base to learn than learning as blank-slates like what was claimed by DeepMind in games [13]. Indeed, the shallow network used in this paper does not restrict the application of the core philosophical idea into deep neural networks, as long as we can combine evolutionary search and the idea of self-taught neural architecture by employing variants of gradient-based learning.

There is some limitation that should not be neglected, including the use of a fixed neural architecture. One plausible solution could be evolving both the weights and the topology of a neural network [19]. This is an interesting pathway for future work if we can evolve variable self-supervised neural architecture which can be an intrinsically general neural learner.

Delving a little deeper into lifetime learning, this category can be subdivided into asocial (or individual) learning (IL) and social learning (SL). Each is a plausible way for an individual agent to acquire information from the environment at the phenotypic level. SL has been observed in organisms as diverse as primates, birds, fruit flies, and especially humans [8]. Self-teaching can be considered an individual learning process which updates the behaviour of a single agent. The relationship between individual and social learning has raised some important scientific curiosity as whether the organism should rely on social or individual information [7], [10], [12], [11]. Social learning may offer another way to propose where the reinforcement signal comes from. If it learns from observing other agents, then the self-learning could proceed from imitation learning. Future work will investigate this line of research and see if the presence of social learning could result in a more complex intelligent behaviour.

REFERENCES

Towards Enhancing NPCs’ Morality: 
The Case of The Elder Scrolls IV: Oblivion

Joan Casas-Roma, Mark J. Nelson, Joan Arnedo-Moreno,
Swen E. Gaudl and Rob Saunders

Abstract. Morality systems are one of the key features that most computer role-playing games (CRPGs) include as a way of allowing players to build their own characters, as well as capturing how the virtual world reacts to their choices. In some of those games, non-playable characters (NPCs) follow their own virtual lives and schedules beyond the players’ actions, which contributes to simulating a more believable virtual world. However, the moral dimension of those NPCs is often very limited, and their morally-relevant deeds usually depend on scripted narratives; this prevents NPCs from showing believable moral autonomy in their actions, beyond what they have been hard-wired to do. In this paper, we analyze the case of The Elder Scrolls IV: Oblivion as a particularly detailed case in terms of its NPCs’ moral profiles, and we argue how, by reusing mechanics that already exist in the game, NPCs could be furnished with a much deeper moral profile and autonomy.

1 INTRODUCTION

Video games allow players to take active part in a story and, sometimes, to make all sort of choices on how to enact it. Some of those choices, specially in computer role-playing games (CRPG), have a clear moral dimension that, in turn, reflect on the way the player character (PC) is seen by the non-player characters (NPCs) inhabiting the game’s world. Even though there are studies on the relationship between video games and morality, most of these works focus on understanding how the human player behind the player character engages in the moral dimension of such choices, or even on whether video games are, after all, suitable platforms for players to engage in genuine moral reflection.

On that regard, works such as [9], [10] or [12] argue that video games allow for genuine moral reflection, while others, such as [6] or [11], challenge that claim. Authors like [5] argue that explicit morality systems are not suitable for that purpose, and defend that only implicit moral choices require the player to actually reflect on their actions. Other works explore the social dimension that the virtual worlds from these video games depict, and focus on topics such as law and power through moral choices, as in [1].

Beyond the effects that moral gameplay may or may not have on players, video games can also be looked at as complex virtual worlds able to account for the moral persona that the player builds through in-game actions, and which affect the way the video game world and its inhabitants react to the player character. [2] argue for the integration of multi-agent systems, artificial societies and complex CRPGs as simulations of virtual worlds as a cross-disciplinary study of morality systems. [8], for instance, focuses on the creation of the player’s social persona in the virtual world of Fable, and [6] examines some of the techniques used in video games’ morality systems, although the paper still ends up focusing on the players’ experience behind those. With respect to building up on tools to design video games’ morality systems, works such as [4] focus on how NPCs could be furnished with a more life-like moral dimension; in particular, the aforementioned paper provides references to alternative ways of encoding moral values and modeling characters accordingly. Those alternative ways, nevertheless, do not belong to existing morality systems in the video games’ industry, and they would need to be adapted and integrated at early design stages of a game.

Instead of focusing on models that could potentially be used in games, in our work we choose to focus on the study of how an existing CRPG already allows to account for NPCs with a certain degree of moral autonomy: The Elder Scrolls IV: Oblivion. Furthermore, we argue how the mechanisms already included in the game could be rearranged and adapted to model NPCs with much more detailed moral profiles.

2 MORALITY SYSTEM IN OBLIVION

The Elder Scrolls IV: Oblivion (called just Oblivion henceforth) is a computer RPG that takes place within a richly simulated social and cultural world [3]. Its social world simulation combined with frequent references to moral aspects of actions presents one of the more complex existing cases of a game morality system with a strong role for virtual agents in forming and enacting moral judgments. Figure 1 summarizes key elements of the game’s morality system, which we’ve produced as part of a larger research project analyzing how different RPGs implement morality systems.

Even though the game is still mainly focused on the players’ experience, and so the PC takes a more relevant role in the model, we can see how the NPCs are still related through all other agents via a Disposition attribute that accounts for how their relationship is. This disposition, in case of the relationship between the PC and the NPCs, is affected by the overall measurement of the PC’s “good” and “bad”

---

1 The Metamakers Institute, Falmouth University, Cornwall, UK, email: joan.casasroma@falmouth.ac.uk
2 Department of Computer Science, American University, Washington, DC, USA, email: mnelson@american.edu
3 Estudis d’Informática, Multimedia i Telecomunicació, Universitat Oberta de Catalunya, Barcelona, SPAIN, email: jarnedo@uoc.edu
4 School of Computing, Electronics and Mathematics, Plymouth University, Devon, UK, email: swen.gaudl@falmouth.ac.uk
5 The Metamakers Institute, Falmouth University, Cornwall, UK, email: rob.saunders@falmouth.ac.uk

---

We’ve gathered some details of how Oblivion is implemented from the UESP wiki [7].
deeds, which are represented through the Famé and Infamy properties. Aside from those, both the NPCs and the PC have different attributes detailing not only their physical and psychical strengths and weaknesses, but also detailing a set of skills in which they are proficient. This, in turn, determines what kind of activities each NPC can carry out in the game, and which it cannot. Furthermore, we can also see in the model how allegiance to certain factions or social groups is also taken into account when determining the affinity between agents.

One of the more unique features of the way Oblivion models NPCs, and which is particularly relevant when considering morality systems, is the attribute of Responsibility. In short, this attribute represents how the NPC feels towards the existing law in the virtual world. Unlike many other RPG games with complex NPCs, Oblivion goes one step beyond and allows NPCs with low responsibility to (non-scriptedly) choose goal achievement over lawfulness. For example, if an NPC has a low responsibility score, needs food, and currently lacks anything to eat, it may steal it from a market stall. This differs from many games, in which only the player and specifically scripted “evil” characters have the possibility to violate norms in a way that the in-game morality system would judge as a violation, which it a step closer towards a furnishing NPCs with a certain degree of moral autonomy.

Nevertheless, and despite this layer of added detail, the game is player-centered: therefore, most NPCs’ properties are only dynamic in terms of their relationship towards the player. In other words, although each NPC has a disposition value towards each other, or a responsibility value on their own, those properties do not change over time; the disposition of an NPC only varies in its relationship towards the PC, and the NPC’s responsibility is set right from the beginning, meaning that the moral behavior of the NPC is constant throughout the game, without the possibility of changing. Similarly, and even though the PC accounts for the Famé and Infamy derived from performing morally good or morally bad deeds, NPCs do not have such scales.

However, and as we argue in the next section, the model could be easily adapted to account for that in the same way it already does for the PC’s case. In particular, the amount of detail that Oblivion’s model has with respect to the moral dimension of the game and the NPCs opens up to the possibility of having NPCs that exhibit a higher degree of moral autonomy than that modeled in many CRPGs.

![Figure 1. Diagram summarizing the operation of the morality systems in The Elder Scrolls IV: Oblivion, viewed from an agent-centric perspective.](image)

---

7 Although The Elder Scrolls V: Skyrim, published also by Bethesda Softworks in 2011, follows Oblivion in many way, it does not implement the Responsibility mechanism for NPCs.

8 Although morality and law are not necessarily the same thing, unlawful actions are the ones that are most clearly reflected by the game’s morality system. Therefore, and for the sake of sticking to the existing game’s mechanics, we restrict ourselves to those existing actions.
Figure 2. Adapting *The Elder Scrolls IV: Oblivion*’s model to enhance the moral profile of NPCs.

On the one hand, their responsibility does not change; namely, if they are caught committing a crime, and thus punished in any way, they cannot “learn” from this action. Their responsibility does not change in any way, neither through reward, nor through punishment, thus resulting in having NPCs whose moral values are set right from the beginning, and left static throughout the course of the game. Just as a human player may decide to stop stealing horses once they get caught by a passing guard, so should NPCs get the same chance to revising their moral responsibility.

On the other hand, NPCs lack a “record” of moral actions. Unlike the PC, who has two separate values of fame and infamy to keep track of their doings, NPCs’ morally-relevant actions are not recorded anywhere. Just as it makes sense for the player to build their moral persona and have the world react accordingly, so should NPCs get the same chance to revising their moral responsibility.

As it can be seen in Figure 1, relationships between the game’s characters is accounted by the disposition attribute, which determines how willing or reluctant a character will be to interact with another one, as well as determining the nature of such interaction – friendly, neutral or hostile. Although NPCs do already have a disposition attribute towards each other, this disposition does not change, and only does so with respect to the PC. The rigidity of such relationships is also an obstacle to what would be desirable, in terms of social interactions reflecting moral judgment of NPCs’ actions. Even though disposition may be affected by different factors, such as belonging to certain factions, we can see how, with respect to the PC, fame and infamy play a very important role in the way relationships evolve. Therefore, and related to what has just been said in a previous paragraph, depriving NPCs of fame and infamy also prevents their relationships from evolve as a result of their moral or immoral doings.

In order to furnish Oblivion’s NPCs with a higher degree of moral autonomy, we propose the following adaptations on the current Oblivion model:

1. **Dynamic moral values**: NPCs should not be permanently stuck in an initial set of moral values, represented in the game by their responsibility. Just as a human player could, NPCs’ responsibility should be allowed to evolve throughout the game. In order to achieve this, actions carried out by NPCs (be them morally positive or morally negative) could potentially modify their responsibility attribute. Following a reward and punishment schema, a pretty straightforward way to achieve this effect would be to increase an NPC’s responsibility whenever it carries out an action increasing its fame, while decreasing its responsibility in the opposite case.

2. **Moral record**: NPCs should have their own fame and infamy scores in order to build a record of their moral doings. As a result of this, their fame and infamy should be reflected on their interactions with other characters in the world in the same way they already do with respect to the PC.

3. **Dynamic relationships**: Disposition between NPCs should change accordingly to their moral doings. In particular, fame and infamy should have an effect on the way NPC relate to each other. In this case, responsibility should not directly modify the disposition value, as responsibility is meant to account for the “private” moral values that the NPC holds, and thus should not be accessible by other NPCs; nevertheless, if an NPC’s responsibility is low enough, the way infamy would affect its disposition should be lower than it would be, if it had a higher responsibility value.

A preliminary adaptation of Oblivion’s model, according to the previous guidelines, is shown in Figure 2. Note that, even though the diagram no longer draws a distinction between the PC and the NPCs (precisely in order to pull NPCs towards the same status as the PC), the PC would not need to have a responsibility value, as the human player behind it would already account for that.

4 CONCLUSIONS

*The Elder Scrolls IV: Oblivion* provides a detailed morality system with NPCs that show an interesting degree of moral autonomy. Through our analysis we see how, despite its strong points, the model cannot yet furnish NPCs with the desired degree of moral autonomy, as NPCs’ moral profile is still shallow and static, and the strong points of the game’s morality system are reserved only for the PC. We identify what a desired model of NPCs’ moral profiles lacks and, furthermore, we point out how those features can already be obtained by rearranging mechanics existing in Oblivion’s morality system. We
argue how the game’s model could be modified to connect existing elements that would lead to NPCs showing a much deeper and dynamic moral profile. This could not only lead to more engaging and interesting NPCs in Oblivion itself, but it would open up to achieving morally complex NPCs in CRPGs using similar mechanics.

As future work, the changes identified in Oblivion’s model could be implemented as a mod for the game to provide an initial prototype. Additionally, the relevant mechanisms of this model could be taken as guidelines to design more engaging NPCs with a higher degree of moral autonomy in other CRPGs.

ACKNOWLEDGEMENTS
This work is funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities) and the Spanish Government through projects CO-PRIVACY (TIN2011-27076-C03-02) and SMART-GLACIS (TIN2014-57364-C2-2-R).

REFERENCES
Evolving Source Code: Object Oriented Genetic Programming in .NET Core

John Speakman

Abstract. Object Oriented Genetic Programming (OOGP) is a method of Genetic Programming (GP) which gives access to standard language libraries, iteration and object-oriented method calls. The implementation of OOGP in this paper shows the automatic generation of retrievable C# files, following standard C# coding conventions with potential access to the entire C# library, derived from a genetic sequence. This new implementation utilises .net Core Roslyn, using reflection, which allows for retrievable, runtime execution and unloading of dynamically generated C# files with scope control in a modern server environment. Experiments were performed on unit tests to validate the algorithms ability to solve simple programming tasks and generate functional, plain text code. This is a new prototype designed to eventually act as the main Artificial Intelligence controller for a novel, behaviourally adaptive, Artificial-Life simulation. The design taken in the development of this algorithm stems from a requirement for a high potential variation in behaviour, processing efficiency in a server environment per iteration through generated code and low a minimal number of generations.

1 INTRODUCTION

The ability for an AI to generate, execute and evolve source code allows the potential search space of an AI to be as broad as that of a human programmer. With a broad search space comes the potential for highly dynamic, adaptive behaviour, through evolution, which may be applied directly as behaviour controllers for agents in games and Artificial Life environments.

This paper proposes a Template-Based Genetic Programming prototype, a novel solution to evolve, execute and output plain text code following standard C# coding conventions [1] with dynamic variables and scope handling. This also opens the plausibility of integrating automatic solutions to simple coding problems into future programming paradigms.

A group of genetically varied agents with the ability to reproduce against a fitness function (a quantified assessment of individual agents) will, given the right parameters, trend towards a solution which optimises their fitness score. If the agents are the body of a source code file and the fitness of agents is derived from unit tests (where the output from a method, when given a pre-defined input, is compared to a pre-defined, expected output), we can generate code which automatically solves simple coding problems. These automatically generated files can be used directly in other C# projects or re-used, through reflection, in the same project in which they were generated.

Koza [2]‘s Genetic Programming introduced the first model for the use of a genetic sequence to construct a computer program. These early forms of GP used simple expression trees, though further exploration into Object Oriented program space indicated that an Object-Oriented approach can provide significant benefits in comparison with grammar-based systems [3]–[6]: improved performance, direct use of class libraries, iteration, object state, generation of reusable, callable classes, sub-classing existing classes.

To expand potential functionality, this algorithm also permits dynamic variable creation and re-use, using a push/pop Stack, roughly translating to the indentation level in source code, allowing more modular, in-method code. Alternative approaches have been taken for stack-based GP [7], [8], which demonstrated the benefits from the efficiency, simplicity and manipulation of modular architectures introduced using this approach, though had not been used for object oriented scope control or source code generation.

This project is built for ASP.NET Core 3.0, a modern, lightweight, cross-platform framework, compatible with multithreaded server environments, which fully support C#.

Some aspects of the architecture in this approach are taken from Template Based Evolution (TBE) [9], [10], an artificial life algorithm for rapid evolution of subsumption architectures, using a genetic algorithm. Of particular interest from this method is the use of evolving variables through a genome which execute into a template. This simulation varies from TBE, as it dynamically constructs new templates, using smaller templates, into source code from a genetic sequence at run time, where TBE builds into a pre-existing template.

2 AUTOMATIC CODE CONSTRUCTION & EXECUTION

The approach to code generation taken in this paper uses pre-defined templates, which build single lines of code from a list of numeric values. Each line of code takes 3 inputs: the first input is used as a reference to a table of single line templates, which constitute the functionality and body of the code. The second and third inputs are used as values in those lines of code and may be used, non-exhaustively, as a value, variable, or function call.

Generation of reusable variables, scope and code indentation are handled by a push/pop stack: the code constructor accounts for lines which increment and decrement scope, where variables whose associated scope is pushed out of the current stack get removed from the available list of variables. If the genetic sequence terminates without resolving scope, scope is then
automatically resolved. This allows the use of more complex code structures, such as loops and if statements.

Code Excerpt 1 demonstrates the use of the push/pop stack to dictate which automatically generated variables (output, A, B, C etc.) are eligible for use by the next line of code. The colour and indentation depth indicate the depth within the stack which each line of source code applies to.

<table>
<thead>
<tr>
<th>Output Code</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>output = output * 2;</td>
<td>output</td>
</tr>
<tr>
<td>double A = 0;</td>
<td>output, A</td>
</tr>
<tr>
<td>if (output &gt; 5)</td>
<td>output, A</td>
</tr>
<tr>
<td>double B = 2;</td>
<td>output, A, B</td>
</tr>
<tr>
<td>if (output &lt; 27)</td>
<td>output, A</td>
</tr>
<tr>
<td>double C = 36 * output;</td>
<td>output, A, B, C</td>
</tr>
<tr>
<td>B == C;</td>
<td>output, A, B</td>
</tr>
<tr>
<td>}</td>
<td>output, A</td>
</tr>
<tr>
<td>double D = B;</td>
<td>output, A, B, D</td>
</tr>
<tr>
<td>double E = 0;</td>
<td>output, A, B, D, E</td>
</tr>
<tr>
<td>if (E &gt; D)</td>
<td>output, A, B, D, E</td>
</tr>
<tr>
<td>double F = 21;</td>
<td>output, A, B, D, E, F</td>
</tr>
<tr>
<td>D == F;</td>
<td>output, A, B, D, E, F</td>
</tr>
<tr>
<td>h</td>
<td>output, A, B, D, E, F</td>
</tr>
<tr>
<td>output = D;</td>
<td>output, A, B, D, E</td>
</tr>
<tr>
<td>}</td>
<td>output, A</td>
</tr>
<tr>
<td>return output;</td>
<td>output, A</td>
</tr>
</tbody>
</table>

Excerpt 1. Scope controller stack displaying accessible variables per line of output code

This generated code is then wrapped with the applicable namespaces. At this point, the code may be returned as a syntactically correct .cs source code file or run through a compiler.

In order to execute this file in the same application it was generated, the code is compiled, at runtime. By using the C# .NET Compiler Platform “Roslyn”[11] code analysis package, an intermediate compilation object, “an immutable representation of a single invocation of the compiler”[12], may be generated from the source code. This compilation object is then emitted using reflection, which builds the object as a collectible[13], in-memory Dynamically Linked Library (DLL) directly into a memory stream, which allows the DLL to be unloaded directly, freeing memory and removing the need to store a physical DLL on the drive. Reflection is used to obtain the MethodInfo[14], a class which provides access to a methods metadata, used here to call the method, for any method in the generated assembly. This may then be stored in an array of delegates, so the method may be called without needing to implement reflection on any future call to the method, significantly reducing call time[15].

3 GENETIC ALGORITHM

Multiple variables per codon are necessary to handle automatically assigned variables properly: building a single line of code using this code constructor takes up to 3 arguments, giving a requirement for the genetic sequence to fit a 3 x N matrix.

The first value, used to determine the main template, selects against a weighting matrix for the likelihood of each template being relevant (for example, an addition call may be much higher frequency than a cosine call). This value has a lower mutation likelihood than the other two values in the codon, as its impact on mutation is significantly greater.

The two other values in the codon are numeric and assume the frequency in code to favour low integers, especially 2 and 3, though with a lower probability of calling 1. The formula used in this model is:

\[ y = \frac{1000}{(1+x/10)} - 9 \]

The algorithm proposed in this paper assumes that the varying depth of scope has a similar effect to positional depth in grammar trees. Applying crossover and mutation at greater depth in grammar trees shows to have a higher likelihood, per mutation, of producing beneficial effects on fitness with a reduced likelihood of detrimental effect [16]. Replicating this, mutation may be applied proportionally to the push/pop stack depth. Crossover may be applied on a similar basis, increasing likelihood of crossover proportional to depth.

Due to the use of a genetic sequence, most standard genetic algorithm functions may be applied: mutation, injection, removal, etc.. Chromosomal block structures may also be implemented, using functions within a class as a chromosome with independently generated code, which can call other functions, even those within the same automatically generated file. While not yet tested, the introduction of these mechanisms is expected to, on average, significantly improve the diversity and fitness of agents over multiple generations.

As this system was intended to support an Artificial Life simulation, with an implicit fitness function, the algorithm can breed individuals on demand, rather than requiring distinct generational batch breeding (though batch breeding can still be applied). This would allow agents to breed based on their current state, independent of other agents or timeframes, where breeding becomes bound to the agent’s ability to survive and breed naturally within their environment.

4 IMPLEMENTATION

This project is currently early in development, being at the first stage capable of producing measurable results. As this is an early prototype, many of the proposed systems have not yet been implemented and the full potential of a complete solution is yet to be explored.

The tested solution runs on a .NET Core, multithreaded environment, with the intention of optimising the number of simultaneous calls to dynamically generated code in an asynchronous environment. This implementation was built to complete simple unit tests, where input(s) were automatically passed to the function, and the resultant outputs were compared against a pre-defined value. The fitness function assessed the number of unit tests which matched this value and, where there was no perfect match, the difference between outputs and that value, generating an associated score with an emphasis on perfect matches.

For these tests, only simple random mutation was implemented, constructing each new generation by duplicating the best agent from the previous generation with random mutation.

As a prototype, the number of defined templates available to the simulation is very low, currently only accessing simple maths
and mathematical comparisons. Similarly, a weighting matrix against the relevancy of templates is also still not yet implemented. The direct effect will have severely increased the likelihood of detrimental mutation and resulted in a generally lower fitness per generation. The fitness function may also be extended to reward through fitness score, reduced length of code and execution time, increasing performance over time.

Even with this simple implementation, we can achieve successful completion of simple unit tests, showing identifiable improvement per generation.

For the following simulations, all agents worked with a pre-set number of lines of code, though potential improvements are expected from injection and removal of code in later simulations. Results from simple tests, with simple problems, indicated a low number of lines tends to solve unit tests in a lower number of generations. Simple unit tests, for example, attempting to divide or multiply by 2, were often completed within the first generation and high complexity tests are yet to be applied.

The following results, shown in Figure 1, show 5 simulations, each with 100 agents over 20 generations, attempting to generate the value 1457 when given an input of 100. Agent fitness above 0.8 is within 0.01 of the correct output.

![Figure 1. Graph of Fitness / generation for 5 simulations](image)

Code Excerpt 2 displays the generated code output from the highest scoring agent from one of the simulations, with fitness >0.8. The sections in Grey represent the main body of the evolved file. The section in green is an automatically generated end of file scope termination. Sections in blue are a wrapper, with namespaces, to generate a syntactically correct .cs file. To verify the code’s validity, this code was exported into a separate C# project where it compiled and executed successfully.

This simulation was set to output, per agent, every generation, a C# file with 15 lines of code in the function body. The output displayed above only utilised 7 of these lines, indicating the liability to create junk code and emphasizing the importance of genetic removal and dynamic genetic sequence lengths, as seen frequently in GP [17].

While subject to substantial change with further development, some simple, preliminary performance tests have been performed\(^2\). To generate, build and execute a MethodInfo class from a file with a single line of code in the function body took, on average, 16.5ms and accessing this class from a delegate took on average 7e-4ms. While solving a simple unit test, the application took 25 seconds to generate, build, execute, breed and display 10 generations of agents, with 100 agents per generation. No significant change to performance was noticed when varying the number of lines of code in the function body between 5 and 30 lines, per agent. All tests test resolved and executed correctly, including a larger experiment generating 10,000 agents, each building 300 lines of code.

```csharp
using System;
using System.IO;
namespace RoslynCore
{
    public static class AutoCode
    {
        public static double F1(double output)
        {
            output = Math.Sin(output);
            return output;
        }
        public static double F2(double output)
        {
            output = Math.Cos(output);
            return output;
        }
        public static double F3(double output)
        {
            output = Math.Tan(output);
            return output;
        }
        public static double F4(double output)
        {
            output = Math.Exp(output);
            return output;
        }
        public static double F5(double output)
        {
            output = Math.Log(output);
            return output;
        }
        public static double F6(double output)
        {
            output = Math.Sqrt(output);
            return output;
        }
        public static double F7(double output)
        {
            output = Math.Pow(output, 2);
            return output;
        }
        public static double F8(double output)
        {
            output = Math.Pow(output, 3);
            return output;
        }
        public static double F9(double output)
        {
            output = Math.Pow(output, 4);
            return output;
        }
        public static double F10(double output)
        {
            output = Math.Pow(output, 5);
            return output;
        }
        public static double FunctionA(double output)
        {
            if (output < 0.5)
            {
                output = output * 6;
            }
            else
            {
                output = output * 7;
            }
            output = output * 6;
            output = Math.Pow(output, 16);
            output += 51;
            return output;
        }
    }
}
```

**Excerpt 2. Example output code, output when input is 100: 1456.897**

5 CONCLUSIONS

The prototype implemented in this paper successfully generated, executed and returned syntactically correct C# files with potential access to the entire available C# library. These tests successfully implemented dynamic scope and variable control, with the ability to automatically generate new variables and restrict their application to within their local scope.

Through simple best-agent mutation over multiple generations, where each generation produces, compiles and executes a new group of C# files, this algorithm successfully completed multiple simple unit tests and returned the solution as a file automatically.

All generated files executed without runtime or compilation issues, both within the live server environment in which they were constructed and, using the output source code, independently in other C# environments. Efficiency of execution when calling generated files is also promising, as they can be called using delegates.

\(^2\) Testing on localhost IIS Express 10, i7-7700HQ
6 FUTURE WORK

The genetic algorithm and breeding functions for this system are still in progress with the anticipation of greatly improved performance per generation. Following this, a more robust benchmark is being carried out. Further research is also required to statistically determine the distribution of common lines of source code, in order to produce an optimal template selection weighting matrix.

This algorithm was designed with the intention of eventually acting as the behavioural controller for a server-based Artificial life simulation. This is intended for use by multiple simultaneous, geographically distributed users in a co-creative, modifiable virtual environment, bringing an emphasis on reducing the runtime processing requirements while maximising the quality of behavioural output on a server framework.

The client-side application for this model is intended for mobile and mixed reality devices, with an initial benchmark for the HoloLens. Continuing work in this direction will breed virtual agents in a virtual environment, using an implicit fitness function dictated by natural selection, rather than an explicit unit test. These agents will need to adapt to indirect human interaction, where users will modify the geometry and interactable objects within the virtual environment, directly impacting the survivability and implicit fitness function of agents. This introduces the need for further optimisations between software efficiency, speed of adaption and adaptive potential in development of the evolutionary algorithm.

When dealing with behaviour controllers for human interaction with this algorithm, a pre-defined genetic sequence which constructs a common behaviour may be implemented. For example, an initial BOID [18] template could be recreated using a manually entered genetic sequence, removing the need for an early, low functionality, high failure rate species. This initial functionality may then be mutated and expanded through adaptive evolution, where dynamic code generation permits absolute modification of behaviour from that point forward.

Alternative development outside of Artificial Life could see this algorithm being used as an alternative solution to common GP problems, particularly where the output is intended for human interpretation. It may also be used to approximate solutions or solve simple programming tasks in everyday programming, forming the basis of a form of pair programming between a human and an AI with integration into an IDE, where the human acts to guide the AI by outlining the required functionality.

REFERENCES

Why is debugging video game AI hard?

Nathan John\(^1\) and Jeremy Gow\(^2\) and Paul Cairns\(^3\)

Abstract. As the complexity of AI systems in video games increases, so too does the amount of time game programmers must spend debugging these systems. Debugging in general can be difficult, but there is a consensus among game programmers that debugging AI systems is uniquely so.

This paper explores the particular issues that debugging video game AI systems pose. We performed a thematic analysis on six interviews with industry AI programmers. This identified three core themes around debugging: difficulty identifying bugs, problems generating reliable reproductions, and conceptual complexity of AI systems which makes them difficult to reason about. We discuss how game AI research could better address these developer needs.

1 Introduction

As the quality and scope of commercial games increases, so too do the scope and ambitions of the AI systems within them. Programmers in industry must often develop specialized tools to assist in debugging the complex behaviours of these systems \cite{5, 8}. Even with the assistance of specialized tooling, debugging errors within AI systems can be incredibly time consuming \cite{1}. The need of programmers to develop these specialist tools seems to indicate that game AI presents unique debugging challenges compared to other game-play systems. After quoting Brian Kernighan’s comments on the difficulty of debugging software, Dill says this counts as double for game AI \cite{4}, which brings up the question, why is this the case?

This question remains mostly unanswered, with research into the specific issues relating to debugging and identifying game AI problems being scarce, as was mentioned by Wetzel as far back as 2004 \cite{10}. However, there is a wealth of interesting and unanswered research questions within this area that have great potential for industry applications, and novel applications of modern techniques. Importantly, to understand which of the questions provide the most benefit to industry, we must have a baseline understanding of the needs of game AI programmers, with respect to debugging game AI systems.

This paper intends to promote discussion among researchers, students and tool developers by examining the particular issues that debugging video game AI suffers from, from perspective of AI programmer’s needs. Importantly, we have been looking for themes that appear across a range of different projects, allowing the insights to apply to a variety of contexts, rather than specific to any particular AI technique or genre of game. Thus we hope the paper will be useful in informing further areas of research, and identifying areas where generalised debugging techniques and solutions could be developed.

2 Background

While there is a broad literature relating to creating AI agents, research focusing on the debugging of AI systems are few and far between \cite{10}. Comparing the availability of research into debugging game AI systems to that of debugging other AI software, such as expert systems, highlights this lack. There are, however some examples of developers and researchers addressing the particular needs relating to debugging AI for video games.

There have been multiple examples of developers looking into methodologies or tools to assist in debugging AI problems. Johnson et. al \cite{7} used logging visualisation in order to help identify or confirm errors in the AI behaviour. Young \cite{11} developed a tool that allowed a developer to scrub backwards in both the game state and the AI’s decision making. As often the root cause of AI bugs often occurred at some point in the past, this greatly assisted in the ability to debug issues.

Additionally, game AI programmers often consider the design of their system’s architecture to reduce the complexity of the AI systems. Dill \cite{3} provides an overview of multiple common architecture tricks that are used to reduce the impact of AI systems complexity, and hence make it easier to debug. Ilsa \cite{6} used their experience working with Halo 2’s complex AI systems to write a case study containing a set of design principles to once again reduce the impact of the complexity of AI systems.

These examples show that despite the lack of direct research, there is an intrinsic awareness among game developers that debugging game AI systems poses its own difficulties, that need to be considered.

3 Methods

We conducted 6 interviews with game AI programmers of different experience levels working on a variety of different projects. The logs and transcripts of these interviews were used as the basis of a thematic analysis to identify common themes. When the interview was framed, the participants were made aware that we were interested in debugging experiences that were unique to debugging game AI systems, rather than debugging games or software generally.

3.1 Interview Methodology

We conducted our interviews either by phone, or via instant message (IM) platforms such as Skype and Slack. We started each interview by asking the participant to describe a particular project that they’d worked on the AI systems for. The reason for this was to establish context for the rest of the interview, and to be able to ask intelligent and relevant questions.
3.2 Thematic Analysis

When analyzing the collected interview data, we chose to perform a thematic analysis, as we were primarily searching to understand what was happening in the data, rather than understanding the mechanisms of why, which would have suited a grounded theory. Due to the level of analysis applied to each data item in this form of qualitative study, generally smaller sample sizes are used than would be for questionnaire data.

Referencing the framework for performing thematic analysis described by Braun and Clarke [2], the exact method would be described as an inductive thematic analysis, identifying the latent themes across the interview data. This meant that the interview was coded without attempting to fit it into a preconceived coding frame, and during the coding we looked beyond the surface meaning of the interview responses.

4 Results

After the initial coding and analysis of the interview data, we formed a set of 15 themes, shown in Table 2. All of the participants acknowledged that the fact that the design of the game was tightly coupled to the requirements of the AI system affected how they were able to identify bugs in the system in some way. For some this was reflected by classifying bugs according to how much they impacted the end users, and when talking about automatically finding AI bugs Frank considered finding a balanced solution to be “extremely tough especially if the game itself is changing too.”

Additionally, when looking at the themes this level, an interesting combination was the desire for as much information as possible from the system to be available for debugging purposes, but also highlighting the importance of tools to filter the potentially overwhelming amount of information that would emerge.

With further analysis, the original 15 themes could be grouped into three overarching issues facing AI programmers when debugging. These are that bugs are difficult to identify, when they are identified, they are often difficult to reproduce, and even once they can be reproduced reliably, the systems that are being debugged are often conceptually complex.

4.1 AI bugs are difficult to identify

Compared to other pieces of software, defining a bug within game AI can be extremely difficult. As previously mentioned, the fact that the AI is tightly coupled to the game’s design means that in many cases the definition of “expected behaviour” can often change. In fact some issues relate more to game-feel rather than anything else, which Eva referred to as “not quite a bug but still something you need to tweek, and get right”. In fact in Danny’s case, on multiple occasions things that were technically bugs in the AI system resulted in desirable behaviour. This fuzziness around the definition of an AI bug can make it difficult to know what to look for, and identify bugs in the first place.
Difficult to identify Bugs

Coupled to design

Danny

“there was another thing that happened from a bug, which we ended up leaving in game [because it] was quite a nice effect”

Brian

“it is so closely tied to game design that it sort of has one foot in design and the other in programming.”

Development environment

Eva

“It’s probably possible to solve if you put your mind to it but we put our energy elsewhere”

Reproduction issues

Reproduction issues

Brian

“[…] make sure that randomized values would be identical. It is not a guarantee, but it increases the reproducibility quite a bit.”

Live examples valuable

Frank

“My favorite trick is replay data. Even if your game does not have full replay support, having a log of the last few seconds of AI data can be huge for the dev who has to fix a reported bug”

Conceptually complex

Many system components

Eva

“these days, what I like to do separate the movement or traversal behaviour from the logic behaviour.”

Many states

Charlie

“Game AI debugging has the additional problem compared to some other kinds of debugging […] in that it is very stateful”

Ada

“the AI has to be really flexible and be able to get itself out of any number of random situations”

Information dense

Charlie

“some of the AI systems involve a huge amount of iteration and so tracking down problems using logging and conditional breakpoints in the debugger is very time consuming.”

Table 3. Some example quotes that show how the themes cropped up in the interview data

4.2 Reliably reproducing bugs can be difficult

A key point that appeared across all the participants was the importance of being able to observe an issue as it was happening. This provided the value of being able to use debugging tools to investigate the state of the system, but also for but also simply for getting a sense of the surrounding context of the bug to inform where to look. This means that although not every bug was difficult to reproduce, reproducibility issues had a large impact on the debugging process.

A common theme across the interviews was, reproduction coming down to a lot of trail and error testing. Eva giving the example “we would spawn this AI in various situations, trying to illicit the behaviours to see when it failed and I think that was... that would be the main solution”, when talking about reproducing bugs.

Charlie and Frank tackled the reproducibility issue in similar ways. Charlie took advantage of a deterministic simulation to be able to replay the game state, and noted “[it’s] hard to over-emphasise how great a tool replays are for debugging AI problems due to the ability to go back and try to find the cause of the state that then caused the actual manifestation of the bug.” In Frank’s case, rather than replaying the entire game state, he supported the ability to replay the AI’s decision making process, allowing for similar benefits in investigation. While there’s likely to always be some issues in reliably reproducing bugs in complicated AI systems, tools such as these point in a potential direction to assist.

4.3 AI systems are conceptually complex
The final primary theme related to the conceptual complexity of game AI systems. This complexity means that even if it’s possible to identify and reproduce a bug reliably, a lot of work is needed to then understand what is causing it to occur. In Frank’s words “it takes a lot of experimenting to really feel why something happens, instead of something else.”

Part of the game AI system’s complexity comes from the fact that even though we refer to game AI as if it were one system, usually it’s comprised of multiple potentially complex components. Even looking at Danny’s relatively simple AI system, it’s comprised of the navigation component which is in charge of path-finding and the movement state machine, which controls targeting and what buttons to press on the virtual controller. These multiple components mean a debugging programmer will often need to consider the interactions between these systems when attempting to fix a problem.

Furthermore, in most modern games the state space that game AI systems operate in are massive. In RTS games, the state complexity comes from the sheer amount of units, potential targets, and tactical decisions that are available to the AI system. However even an MMORPG NPC has a wide range of choices between which abilities to use, as well as needing to react to player actions. Frank captured the essence of this state space problem perfectly saying that “thinking about AI in terms of all the possible situations [...] is really tough, because there are a lot of possible situations!”. An additional problem that game AI systems have compared to most other software or AI systems is that temporal state can play a significant role in the state of the system’s behaviour. Charlie mentions encountering bugs where “you might be able to see that it’s caused, for example, by a variable being set to some value, but have no idea when the variable got into that state.”

Finally, game AI systems in modern games tend to be incredibly information dense, with many components constantly passing large amounts of data between each other. This multiplies the effect of AI systems being real-time systems that are operating in the aforementioned large state space, and can make it difficult to make sense of the cause-and-effect relationships between the inputs and outputs of the systems. The most common tool to assist in this is writing to the console, however the sheer amount and rate of data can be a problem. Charlie alluded to this, saying “log too much and it makes it difficult to find relevant information (and it also slows the game down).”

5 Discussion
Through collecting and analyzing interview data from industry AI programmers, we could get an insight into issues that make debugging game AI a unique challenge. While the problem of conceptual complexity is common among other large or complicated software projects, the fuzziness of most game’s designs cause unique issues in identifying and reproducing bugs. Our interview data shows that game AI programmers acquire an understanding of these difficulties through experience, and often work on specialized tools to help lessen their impact. These tools are often are limited to their immediate development team, and not shared or reused outside of the current project.

Considering the three major themes, each has a different possible area for improvement, and level of game generalisability. The most case specific theme is the difficulty in identifying bugs. Due to the fact that this is often tightly coupled to the design of the game, we can assume that general methods of identifying bugs will be hard to come by. That being said, there are times where AI bugs present obviously bad behaviour, and tools that could identifying the cases which aren’t tightly coupled to design would free the mental load of programmers for the more nuanced issues. For example, the area of search-based software engineering contains solutions such as automatically generating test cases[9], which could be transferable to game AI systems. It would be interesting to explore what the effects of writing specifications for the behaviour of the AI systems would have on programmer’s ability to identify bugs in the systems. Also, as identifying bugs generally requires tailoring around the game’s design, this could be an interesting area to use machine learning techniques, either comparing to labelled game play data, or analysing sets of game play heuristics.

When looking at the difficulty of reproducing bugs, reducing the impact of this on an AI programmer generally relates to the architectural design of the game itself, and choices made early in the game’s development. Multiple interviewees mentioned that having access to deterministic simulations and/or the ability to replay the AI system’s decision making process makes reproducing issues far easier. Interestingly, replays and deterministic simulations are known solutions, but not considered by developers to be standard debugging tools. This is an area where the solution isn’t technical research, but is instead just attempting to highlight the importance of, and standardising existing techniques.

The theme that most immediately appears to have generalisable solutions would be the conceptual complexity of game AI systems. Interestingly, most of the tools that the interviewees mentioned that they developed themselves related to this theme, whether it be tools to filter through large amounts of logging data (assisting in the information density), or visualisations of the current and previous states of a particular decision-making AI (helping in navigate the statefulness of the system).

6 Limitations and Further Work
While we aimed to get a broad range of project genres and programmer experiences, there are some missing genres, meaning that this study may not capture the entire experience of game AI programmers across the industry.

In order to further encourage discussion, and aid in the communication relating to bugs in game AI systems, it might be wise to develop a bug typology, with examples.

For each of these themes that we have defined, there is potential for interesting further work. With regards to the difficulty of identifying bugs, it would be interesting to look into the types of bugs that AI programmers encounter, and how many of them are effected by the design of the game. Additionally, studying the cost versus the benefit of spending resources developing tools to assist in identifying bugs may be an interesting future piece of work.

When looking at issues in reproduction, there is potential in researching automatic or intelligent methods of reproducing a bug, when given a specification of the bad behaviour. Additionally, looking into methodologies of designing game systems to help reduce the difficulty of reproducing a bug once it’s been observed.

Finally, studies into how each of the sub-themes of conceptual complexity affect a game programmers ability to debug an issue may provide insight into which has the larger effect. Also, looking into ways of displaying large amounts of data, to reduce the impact of the information density issue would be useful work.
Acknowledgments

Nathan John is supported by the IGGI EPSRC Centre for Doctoral Training (EP/L015846/1).

REFERENCES


Understanding Perception of Game AI Through YouTube Critique

Tommy Thompson

Abstract. The AI and Games YouTube channel aims to present informative content on the inner workings of how artificial intelligence (AI) is employed within the video game industry. This results in a variety of responses to the information presented as consumers of games media are asked to engage with the technical and design underpinnings of characters and systems that employ AI techniques. This paper is a preliminary analysis of the forms in which users engage with these videos to better understand how this information informs their perspective, reaffirms biases or is outright rejected on various grounds. In addition, we discuss the benefits and drawbacks that could emerge from engaging with this medium for examining the perception of AI in games.

1 Introduction

In recent years online video and streaming sites such as YouTube and Twitch have transformed the discourse around video games and the level of agency that end-consumers (i.e. players) have within the market. The video content produced ranges from streaming gameplay footage of a given title to a live audience, to video essays discussing various aspects of game design and implementation. These videos accumulate thousands if not millions of views as audiences from around the world engage in discussion with their favourite creators and fellow followers.

While the most popular format for gaming-related content online is the ‘Let’s Play’ - whereby the creator of the video records their interactions with the game alongside footage of the game itself - or the ‘livestream’ - whereby users play a game and commentate on their experiences live without editing - there is a growing community of creators focused on essays built around discourse of games development and industry topics that has evolved from similar work in film and literature [2].

Video essays are an increasing popular format on YouTube which enables a creator to discuss not just their engagement or enjoyment of a video game or specifics of its game design and development, but provide an avenue through which subscribers and viewers can engage in conversation much akin to more traditional forums.

It is this particular format of video content that the author has engaged in with the formation of the YouTube channel ‘AI and Games’. The purpose of this channel is two-fold: to more broadly disseminate the technical underpinnings of artificial intelligence practices within the video game industry, but also how academic research utilises the technical underpinnings of characters and systems that employ AI techniques. This paper is a preliminary analysis of the forms in which users engage with these videos to better understand how this information informs their perspective, reaffirms biases or is outright rejected on various grounds. In addition, we discuss the benefits and drawbacks that could emerge from engaging with this medium for examining the perception of AI in games.

2 ‘AI and Games’

‘AI and Games’ is a YouTube channel founded in 2014 by the author with the intent to provide additional resources for students studying artificial intelligence and/or video games at university. Regardless of a students level of study, the project aims to provide an easily accessible format for engaging in the topics being discussed, whilst also pointing to relevant sources for further study. The series is researched from a variety of conference presentations, research papers, technical papers, developer blogs, magazine articles and in some cases interviews with the developers in question. The presentation format aims to condense what is often highly technical detail into a more accessible format for students to either take on board the more surface-level details for purposes of their own projects, or to then conduct a more scholarly examination of the subject matter through the relevant materials. Outside of the pedagogy motivations, the show also aims to provide an entertaining format for the wider video game community to better appreciate the work related to AI and games in both the video games industry and academic study.

At the time of writing, the author has published 38 videos in the main ‘case study’ series. In case studies, the author explores a particular video game or research project and explores the impact AI research or development has had upon that body of work (Figure 1). In the context of a commercial video game the video will focus on the design decisions and implementation of a given AI system within the title. Meanwhile for videos inspired by research projects, the emphasis is on understanding why a given game provides an interesting research test-bed and explores some of the work being conducted in this area.

1 AI and Games (UK), email: contact@aiandgames.com
3 Examining Comments

For the purposes of this initial study we focused upon the discourse written within the comment section of case study videos. These videos typically see significant number of users engage with the content of the video discussing the merits of the work presented and their feelings towards it. We readily concede that reading a YouTube video comment feed - much like most online discussion boards - is not necessarily a reliable format for rigorous discourse. However we have observed a significant amount of engagement in the comments section, largely as a result of the YouTube format and the possibility to engage - be it positively or negatively - with other users on the topic at hand. Furthermore, there is existing precedent within academic research to engage in the comment habits and behaviours of users on online forums such as YouTube to understand their behaviours and attitudes with regards to the subject at hand as well as one another [3, 1].

For the purposes of this initial study, we have pulled comment data from the 10 most popular case study videos based on ‘AAA’ commercial video game titles\(^2\). This collection at the time of writing has a total of over 1.5 million views with approximately 4200 comments. These comments were scraped from each YouTube page as JSON and CSV files for later consumption and analysis. It is worth noting at this time that this is only a subset of the complete list of comments originally submitted to the page. The comment section of the channel is heavily moderated, with many comments that are deemed inflammatory or insensitive being held for review and often later removed before being published on the video page.

The focus of this initial examination was to attempt to classify the discussions that arise within a given videos comments section that relate to a users interpretation, understanding or appreciation of a particular AI system or design within a commercial game. What common threads arise across each video and the types of behaviour that arises within the viewing community in relation to AI in games. This resulted in a number of comments being filtered out or ignored given they focus on the opinions on the video itself, the author or discourse related to the games themselves.

4 Response Types

Upon examining the collection of responses received on videos and filtering irrelevant or inappropriate comments, We identified a number of consistent responses across each video in relation to the AI topics being discussed. What is interesting is examining how players relate to the content discussed and how it reflects upon their experiences with the game. Many of the comments submitted imply that the author has played the game(s) being discussed in the video. As such their responses are often a reflection of how they have reacted to the content presented in the video, which we have broken down into six categories or themes that will now be explored in detail.

To give readers an idea of the types of content being identified under each theme, we provide anonymous and censored versions of actual comments submitted to the channel in Table 1.

4.1 Confirmation

The first comment type observed is one of confirmation; that the content of the video - and the AI being discussed - aligns not just with the authors understanding of the game, but reaffirms their positive feelings towards this game. In this instance it seems evident that the author has played the game in question and has enjoyed their time playing it. Perhaps more critically, the video now acts as a source that reinforces the authors confirmation bias. Whereby it has only strengthened their positive opinion towards the game in question and often results in anecdotes related to their interaction with these AI systems in the games themselves.

4.2 Appreciation

The second positive comment type observed is one of appreciation: where the user expresses their respect towards the work involved, how it enhances the game in question and in some instances changes their opinion of the developers responsible. This is achieved courtesy of the viewer now having a stronger understanding of how the AI systems are built within the game. This particular comment type aligns closely with the previously established ‘confirmation’ type. However a critical difference is that they often lack the more positive comments regarding their experiences with the game, but instead focus on admiring what has been achieved in the game as an artefact.

Table 1. A collection of comments lifted from the sample set that reflect the six themes identified in this paper. These comments range from appreciation of the AI found within the game to outright dismissal of the content explored in the video.

<table>
<thead>
<tr>
<th>Confirmation</th>
<th>Appreciation</th>
<th>Intrigue</th>
<th>Dismissal</th>
<th>Correction</th>
<th>Denial</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;F%$&amp;, now I have to play halo 3 for the 400th time because it’s such a good game.&quot;</td>
<td>&quot;Great video! That explains a lot about what makes this game unique.&quot;</td>
<td>&quot;I hope in the next game we get more menacing vocalizations and breathing, more drool.&quot;</td>
<td>&quot;And yet [the AI subject of the video] is still useless&quot;</td>
<td>&quot;It is scripted I’m afraid. There is no AI at all.&quot;</td>
<td>&quot;Stop throwing the word ‘AI’ around as if any computer game actually has intelligence.&quot;</td>
</tr>
</tbody>
</table>
| "I love this game to death and knowing how complex the AI (and the inner system) is makes me love it even more" | "Wow. I knew making a game was complicated, but its hard to appreciate just how complicated this is. Great video, you got a subscribble" | "is HTN AI common in real robot as well, such as a Roomba?" | "This game was terrible bland with a stupid story" | "The AI can shoot through walls when synchronise shots" | "There was no AI. This was just marketing."
| "One of the best A.I.s every created! I literally fell in love with it, when it killed me for the first time." | "Really in deep. it was really interesting! I enjoyed a lot" | "Very interesting. I wonder what the biggest differences are between HTM and GOAP?" | "i don’t remember any of that from when i played this game" | "12:25 that isn’t entirely true, ive seen the slime dripping from the vents" | "This guy has no idea what hes ****** talking about"

It is subsequently more difficult to interpret whether the author has in fact played the game in question.

4.3 Intrigue
The final positive interaction acknowledged is one of intrigue. This comment provide evidence that the commenter has been given sufficient information to further consider the possibilities of the technologies and techniques discussed in the video. This results in them asking questions aimed either a promoting discussion within the comment section or to the author directly. Much like appreciative comments, it becomes increasingly more challenging to interpret whether the author has played the game that was the focus of the video.

4.4 Dismissal
Moving into more negative comment types, a common and often least toxic format is one of dismissal. This can be identified as a comment that largely ignores the merits of the work discussed throughout the video given there are larger issues surrounding the game itself that inhibit the authors ability to appreciate or enjoy the work described. This is in many respects the opposite of the appreciation theme, whereby the viewers issues with the game itself negatively influence their engagement with the discourse related to the AI implementation. These dismissive comments seldom ever discuss the subject matter of the video and instead base their discourse on personal experience or opinion related to other facets of the game in question.

4.5 Corrections
This is a rare instance whereby the viewer will seek to correct the information presented in the video. This often arises due to small technical details not being explored that the viewer wishes to clarify. This also occurs in circumstances whereby a game has changed or evolved since the initial implementation. However in some instances, this is actually driven by a desire to correct the information that is presented in despite it being accurate, with the commenter providing inaccurate information without supporting evidence. This carries strong similarities with the previously identified ‘dismissal’ themes, in that the commenter will correct statements made in the video to better align with their own interpretations. Even when this contradicts information provided in the video that was accurately researched.

4.6 Denial
The final theme found within the comments is that of denial and is the most negative form of comment. In this instance, the viewer is unwilling to accept the information presented within the video as accurate, regardless of its validity and source. These comments can often lead to a subsequent correction or dismissal comment as previously identified. However, what is interesting is the number of instances whereby outright denial occurs despite neither evidence to suggest alternative information nor any negative feelings towards the game itself. An argument could be raised that the commenter is deliberately posting controversial messages (i.e. ‘trolling’) in an effort to provoke the audience.

5 Discussion & Conclusion
Whilst this preliminary study is aimed at providing a formative evaluation of our research questions on the value of online comments in assessing the general public’s understanding of game AI, there are some interesting insights we can lean into. The range of different responses as identified in Table 1 indicates that even from a more manual review there is a range of perspectives that viewers can form.
There is potential for further work using the existing data set to observe the level of positive and negative engagement with videos, we feel that there are greater opportunities to be found in this space provided we know more about the users themselves. A follow-up project is currently being conducted on the existing data using sentiment analysis to identify the overall positive and negative engagement with each video to see how these attitudes persist across individual videos as well as the channel as a whole.

One interesting research avenue to consider is whether particular comment types established in this taxonomy appear more frequently on individual videos. Combined with the sentiment analysis, what is the overall ‘mood’ towards a given video that has been posted online. Can we understand what external influences may have driven that behaviour? This could be plausible if a given game was not as popular as others or is perceived negatively by game-playing audiences.

Furthermore, this preliminary taxonomy only considers whether a given comment is positive/negative comments and identifying the manner in which this behaviour is expressed. As mentioned in Section 3, the comments assessed were filtered to remove those that were deemed ‘off-topic’ and aimed more towards the author and the wider audience of the channel. A further examination of the data could enable an understanding of how many of the comments raised were relevant to the topic at hand and the level of engagement on a per-video basis. This could reflect not just the quality of content being delivered, but also the overall sense of interest or engagement in the particular game or topic being explored.

Beyond the impact upon this individual YouTube channel, it would be of great pedagogical value to better understand the motivations and interests of the viewing audience. Are the viewers of video essay YouTube channels watching purely for entertainment or seeking educational value? Which channels do they openly subscribe to based on their interests? In positions of further and higher education can we find ways to align these popular forms of video content as means to reinforce topics and delivery in our classrooms rather than suppress or supplant it. This is an avenue that we have considered and would require a larger amount of data collection and analysis from the channel audience.

Often under the veil of anonymity, the internet enables users to express their opinions on a range of topics often free from further discourse - both for good or ill. While this can prove to be a difficult if not impractical field for the mining of user data, as researchers and educators we are in a unique position to better educate and inform the wider world of the realities of AI and other technical fields. If we can better understand their perspectives, it can help influence how we engage in public interaction and discourse.

REFERENCES

Approaching the Analysis of the Spectatorship of AI in Saltybet.com

Rory Summerley¹

EXTENDED ABSTRACT
The proposed submission is a paper-in-progress that seeks to examine the appeal of watching AI compete against one another. This paper takes, as its primary case study, Saltybet.com [1], a streaming site which uses the M.U.G.E.N. fighting game engine [2] and various player-made AI characters, and has them fight in exhibition and tournament matches. Spectators of these matches can bet fake money or ‘salty bucks’ on the outcome of a match and a small community has grown around Saltybet’s unusual entertainment prospect.

Many discussions of AI focus on their ability to be optimised for a specific task and only rarely is the appeal of AI as a form of entertainment, particularly one which involves so much blunder and imperfection, a focus of discussion. Even within the realm of games AI are often discussed for their capacity to compete with or best human players in games like Chess, Go or DOTA2 [3] or to learn how to perform a very specific task within a game-space defined by a fitness function [4] [5]. Although highly competent AI typically occupy the academic mainstream’s attention, Saltybet stands as an example of how inefficient or ‘bad’ AI can be a source of entertainment. This entertainment seems to stem from a mixture of the AI’s behaviour, the visual depiction of an AI character as well as the context in which the spectatorship happens. Saltybet is framed in a very similar way to many Twitch livestreams of fighting games between human opponents. The key difference is that no human players are present, something that is typically a core part and appeal of watching others play. In presenting this paper the author seeks to open a discussion on the place of AI designed to entertain as well as the use cases of AI that are sub-optimal or - to put it characteristically - ‘foolish’. A central question that this proposal seeks to answer is what is the appeal of AI vs AI spectatorship?

Saltybet streams typically involve a series of betting phases and phases where fights actually occur. During the betting phase, betting spectators can speculate on which character will win based on their visual appearance, gambling fallacies, character loyalties (e.g. characters from a series such as DragonBall Z), traits such as having a sword (which makes it likely that a character will have large disjointed hitboxes) or prior knowledge of a specific AI. Bettors can choose how many ‘salty bucks’ to wager based on their assessment of the character and whether or not they win or lose. Then the fight begins, usually with the best of three rounds determining the winner. When matches begin it often quickly becomes clear who will win or lose due to certain behaviours or traits of the AI such as extremely damaging attacks, stun-locking an opponent so they cannot act, inactivity on the part of a poor AI or many other imbalanced behaviours. Throughout the betting and fighting phases a live chat feed can be seen next to the broadcast where players participate in discussion of the matches and Saltybet itself (as illustrated in Figure 1).

Figure 1. A screenshot showing the layout of a Saltybet stream. (Bettor information [left], Match stream and match information [centre], Chat [right]).

Alongside Saltybet, this paper also briefly considers other examples of entertaining and foolish AI including: community streams of various Mario Party games (disparagingly dubbed ‘Mario Retardy’[6]); autonomous AI robot competitions (Robot Sumo [7], Robot Soccer [8]); mistakes by everyday AI companion applications (e.g. Alexa [9], Autocorrect [10]); and procedurally generated matchmaking between AI (BadCupid [11]). These examples will be examined, with video examples and a live demonstration of Saltybet.com itself, to help answer the central questions of this paper and supplement the discussion of the appeal of Saltybet. A distinction is made between those AI that are intentionally implemented to be incompetent (Mario Retardy, BadCupid), those AI that emergently develop behaviour perceived as ridiculous (digital evolution, artificial life, RobotSumo) and mixtures of emergent and intentional foolishness (Saltybet). Currently the study seeks to document typical instances of spectatorship with reference to the Saltybet stream’s live chat as well as surrounding community material including the twitch chat, comments on archived footage. The intentions of the people that stream and design these AI will be scrutinised.

The intention of making AI entertaining for spectatorship is worth discussing especially given the focus of AI researchers. Lehman et al.’s [12] collected anecdotes of various evolutionary computation and artificial intelligence projects that surprised their creators with bizarre, unusually inventive or extreme behaviours shows that there is a rich font of discussion, yet to be tapped in the field of artificial intelligence. The examples discussed in Lehman et al.’s case studies share much in common with Saltybet and other AI vs AI games for their apparent entertainment value. Media studies sources including Taylor’s [13] comprehensive discussion of the Twitch platform, Fagan’s [14] speculative analysis of the appeal of the Roman games, Geertz’s [15] ethnographic study of Balinese cockfighting and Klastrup’s [16] concept of shared ‘player stories’ will be used to

¹ Games Academy, Falmouth Univ, TR10 9FE, UK. Email: r.summerley@falmouth.ac.uk
help understand the situation and experience of watching AI play against each other.

The appeal of AI spectatorship appears to come from a mixture of othering (seeing the AI as separate and inferior), characterisation (typically as foolish or ridiculous) and the thrills associated with other traditional sports spectatorship (skilful play and climaxes). These aspects of the appeal of AI spectatorship are highlighted for discussion and their place in wider discussions of AI will be interrogated. In discussing the appeal of the spectatorship of AI many potential, related causes must be disentangled from one another in order that a clear understanding of the phenomenon of AI spectatorship can be fully understood. The pleasures of gambling, for example, might be a more prevalent factor than others given the inherent psychological appeal of betting on outcomes (even mundane outcomes). Does the appeal lie primarily with the AI themselves or elsewhere?

A proposed hypothesis of the paper is that AI vs AI spectatorship feeds on the desire to anthropomorphise AI and thus narrativise the spectated event. Scholars such as Bryson [17], Gunkel [18] and Floridi & Sanders [19] have discussed the risks of anthropomorphising AI in this way which makes for an unusual opportunity to discuss the moral patency of AI in a context which seems harmlessly entertaining but also superficially resembles activities such as dog-fighting or cock-fighting. However, this anthropomorphisation seems to fill in for the absence of the spectacle of human players while also serving as a satisfying and morally acceptable way of othering the focus of spectatorship (often by discussing the AI’s lack of intelligence, skill or sensibility). This ‘perceived foolishness’ appears to be the central appeal and this paper presentation intends to discuss this aspect of AI in depth.

Ultimately, the proposed submission seeks to open a discussion on the potential directions of this paper as well as the subject of AI spectatorship more generally. How does understanding the appeal of AI spectatorship inform its future and what developments could be made in this space? The author intends for the presentation to bridge discussions of how AI is understood in the fields of both computer science and the humanities as well as in a gaming context.

REFERENCES


