Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

Modeling Player Experience with the N-Tuple Bandit Evolutionary Algorithm

Kamolwan Kunanusont
University of Essex
Wivenhoe Park
Colchester, CO4 3SQ
United Kingdom
kamolwan.k11@gmail.com

Simon Mark Lucas
Queen Mary University of London
Mile End Road
London, E1 4NS
United Kingdom
simon.lucas@qmul.ac.uk

Diego Pérez-Lébana
Queen Mary University of London
Mile End Road
London, E1 4NS
United Kingdom
diego.perez@qmul.ac.uk

Abstract

Automatic game design is an increasingly popular area of research that consists of devising systems that create content or complete games autonomously. The interest in such systems is two-fold: games can be highly stochastic environments that allow presenting this task as a complex optimization problem and automatic play-testing, becoming benchmarks to advance the state of the art on AI methods. In this paper, we propose a general approach that employs the N-Tuple Bandit Evolutionary Algorithm (NTBEA) to tune parameters of three different games of the General Video Game AI (GVGAI) framework. The objective is to adjust the game experience of the players so the distribution of score events through the game approximates certain pre-defined target curves. We report satisfactory results for different target score trends and games, paving the path for future research in the area of automatically tuning player experience.

Introduction

Automatic Game Design (Togelius and Schmidhuber 2008) is a subfield of Game AI that aims to apply AI techniques in assisting game design tasks. A game may consist of game rules, game maps (levels) or certain in-game parameters that can be tweaked. Game design tasks include deciding the look-and-feel and functionality of these components, which usually requires a number of playtests to determine the best composition. Manually playing games repeatedly for this purpose is usually time consuming and inefficient for a few reasons. Firstly, human testers tend to be inconsistent and possibly unable to capture many possible playing behaviours. Secondly, most of the human testing sessions are significantly slow compared to automatic testing. This is adequate to point out that an automatic tool to repeatedly play the games for all settings and evaluate them similarly with human testers is necessary. Attempts to develop such systems have been proposed using Evolutionary Algorithms (EA) (Browne and Maire 2010) (Ashlock 2010) (Sorenson and Pasquier 2010), mainly because of their generality and their suitability in optimization problems.

Automatic Game Design has been employed in the literature focusing on different objectives: from proof-of-concept studies (Togelius and Schmidhuber 2008) (Nelson and Mateas 2007), to focus on certain game components such as maps (Ashlock 2010)(Sorenson and Pasquier 2010) or even complete games, such as Ludi (Browne and Maire 2010) or ANGELINA (Cook and Colton 2011).

Another game design related task that would benefit from an automatic approach is parameter tuning. Isaksen et. al. (Isaksen et al. 2015) work for Flappy Bird, shows that EAs are applicable for this task and underlines its importance in the game development process.

Recently, the General Video Game AI competition (GVGAI) (Perez-Liebana et al. 2016) has just introduced a new *Game Design* feature in the framework that does not focus on game playing algorithm development, but on providing a "general" parameterization task instead.

The objective of the research work presented here is to propose a general approach for GVGAI game parameter tuning to evolve games that provide a specific *player experience*, particularly focused on the game's score trend. A set of 4 functions has been selected as targets, so that the score progression achieved by an agent that plays this game *approximates* such curves. Since player experience is difficult to measure due to the uncertainty of player's behaviour and skill, we used 2 GVGAI agents that employ score-based heuristic functions in action selection. Therefore it is fair to assume that these AI players would always look to gain score when the opportunities present, unless such score-able option leads to losing the game.

The remaining of this paper is organized as follows. A related work review is carried out in section 2. This is followed by essential background knowledge (section 3). The approach and experiments are then described in section 4 and 5 respectively. Then we conclude the paper with some final remarks and outlining potential future works.

Related Works

Nelson and Mateas (Nelson and Mateas 2007) defined automatic game design as a problem-solving task by declaring 4 design factors needed to create a game. This includes *game mechanics*, *game representation*, *game thematic content* and *control mapping*. They also developed a method to auto generate a game by designing all of these components using a set of common sense composition rules. Togelius and Schmidhuber (Togelius and Schmidhuber 2008) used an EA to evolve a 15×15 2D grid game rules from scratch.

They claimed that this was the first attempt to do single-player game rule evolution and to apply EA into non-board game design. Browne and Maire introduced *Ludi* (Browne and Maire 2010), which was the first combinatorial automatic game generator framework with a commercially published game (Yavalath), that has an interesting winning condition beyond human common sense (create a four-in-a-row without three-in-a-row). This points out that a computer program can find a rule set that humans are interested in but failed to find. Cook and Colton proposed an automatic arcade game generator named ANGELINA (Cook and Colton 2011). They implemented generators for three game components: rule sets, character layouts and maps. An EA was applied to evolve these components separately, while share information of the fittest individuals with others.

For game parameter tuning, Isaksen et. al (Isaksen et al. 2015) defined a parameter space of *Flappy Bird* and tweaked those parameters using EAs to see variants of games while keeping the rules fixed. Four unique settings were discovered, all are different in pipe lengths, gaps, player size and gravity, which leaded to a unique gameplay experience for each. This shows that only changing parameters can lead to new game sets, even when the same rules. Liu et. al. (Liu et al. 2017) evolve *Space Battle* game parameters using *Random Mutation Hill Climber (RMHC)* evolutionary algorithm and its improved version called *Multi-Armed Bandit Mutation RMHC (MABM-RMHC)*, applied UCB equation to select a next mutated parameter and value. MABM-RMHC performed better as it explored more and converged faster. Kunanusont et. al. (Kunanusont et al. 2017) proposed a novel EA named *N-Tuple Bandit EA (NTBEA)* for Game AI optimization and applied it to a modified version of Space Battle called *Space Battle Evolved* to tune 30 in-game parameters, aiming to evolve games that favour skillful players to weak players. The fitness value of a game was calculated after three playtests with different General Video Game AI (Perez-Liebana et al. 2016) controllers: *MCTS* as a skillful player, *RAS* as an middle-skill player, and a *One-Step-Look-Ahead (1SLA)* as a weak player. The fitness function was chosen following the Relative Algorithm Performance Profiles (RAPP) principles: it should maximize the gaps between MCTS-RAS score and RAS-1SLA score, with assumption that good games should separate players with different skills.

The automated parameterization research reviewed earlier are all done based on player experience, but did not consider a non-subjective evaluating matrix for the whole gameplay like score trends. Instead, our work aims to evolve games that provide specific scoring experience throughout the game, regardless of the players.

Background

General Video Game AI

General Video Game Artificial Intelligence (GVGAI; (Perez-Liebana et al. 2016)) is a framework and competition for General Video Game Playing (GVGP), which aims to advance research on Artificial General Intelligence (AGI; (Goertzel and Pennachin 2007)), particularly focusing on the video games domain. GVGAI was firstly introduced in 2014 and has since become widely spread among the researchers in the Game AI field.

Research in AGI involves developing AI that is capable of solving problems with different levels of difficulty and characteristics with very little domain knowledge. In GVGAI, the agents are not given the game rules in advance, but they can use the environment information and a forward model to provide an action to the game. Since video games usually require real-time interact ions with the game, the agents are asked to return an action in just a few milliseconds.

Selected Games

Three GVGAI games: Defender, Waves and Seaquest were chosen to do parameterization from pre-defined score trend function. These games have been selected because, firstly, they all have *accumulative* score-system, in which players obtain score at any time during the game. Secondly, the in-game events in which the players gain score are similar, as they are all shooting games. Given these characteristics, they are good testbeds for our experiment as the score trends can be varied from the pre-defined game parameters.

1. **Defender** is inspired by the game Defender in the Atari 2600 framework. In this game the player plays as an armed aircraft trying to protect a city from alien assault that occasionally drop bombs to destroy the city. The aircraft can shoot missiles at aliens to kill them. In the GVGAI version, aliens move from their spawn points horizontally in one direction and are harmless to the avatar. The avatar missiles is limited and can collect more from supply packs that constantly fall from the sky.
2. **Waves** is an alien fighting GVGAI game that the player controls a spaceship trying to survive from a big alien attacking wave. Aliens are spawned from their spawn points and moving considerably fast towards the avatar's starting position. Aliens also shoot harmful missiles and lasers to the avatar. The avatar can regain health by collecting shields that dropped from destroyed lasers. The game ends if the player can survive during a number of time steps, unless it dies earlier by losing all health.
3. **Seaquest** is inspired by Seaquest in the Atari 2600 framework. In this game, the player controls a submarine aiming to rescue divers while defending itself from the aggressive underwater creatures by shooting them with torpedoes. The submarine has an oxygen level bar that will continuously decrease with the time it is underwater, and can be refilled once it moves to the surface. In GVGAI version, the player wins if they survived until a maximum number of game steps is reached.

Play-testing Controllers

In the experiments shown in this paper, the games were played and evolved using a GVGAI controller called *Rolling-Horizon Evolutionary Algorithm* (RHEA). To verify that the evolved games do not only provide the targeted score trends for this agent, a *Monte Carlo Tree Search* (MCTS) agent is also employed to validate the results. This section describes the methods behind these controllers.

Rolling Horizon Evolutionary Algorithm (RHEA) RHEA was first introduced by Perez et. al. (Perez et al. 2013) as an online planning game player and then implemented as a sample controller for the GVGAI framework. RHEA has achieved promising performance in average and multiple variants and enhancements have been proposed in the recent literature (Gaina, Lucas, and Pérez-Liébana 2017a; 2017b). The general RHEA starts with randomly initialized a population of individuals (sequence of to-be-applied in-game action). Then evaluate each individual's fitness by applying the sequence of actions and computing a value for the state reached at the end. After that regular operators (crossover, mutation and elitism) are applied to generate the next population. Next, the new individuals are evaluated and the best N individuals are kept. The last two steps would be repeated until exhausting the decision time budget, and the first action of the best individual is returned.

Monte Carlo Tree Search (MCTS) MCTS (Browne et al. 2014) is a tree search algorithm designed to tackle problems with large branching factors, such as Go. Its main strength is the ability to explore mainly those sections of the space that are most likely to give promising outcomes, while neglecting other non (or almost non) informative branches. The MCTS algorithm can be divided into 4 phases: **Selection**: Repeatedly selecting a known child node from root using Upper Confidence for Bounds (i.e. with the UCB1 equation 1) until a node with some unexpanded children is found.

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \tag{1}$$

The best action ($a*$) is selected as the one that maximizes the UCB1 equation. $Q(s,a)$ is the estimated reward of taking action a from state s. $N(s)$ is the number of times s has been visited and $N(s,a)$ represents how many times action a has been chosen from s. C is a constant that balances between exploration and exploitation, with a value typically set as$\sqrt{2}$ when rewards are bounded in $[0,1]$. **Expansion**: A new unexplored child node is added to the tree, typically selected at random. **Simulation**: A Monte Carlo Simulation (rollout) is done from the newly added child node until the termination condition is reached, by picking random actions at each decision point, either uniformly or informed by some heuristic knowledge if available. **Back-propagation**: The outcome of the rollout is repeatedly back-up through the visited nodes from the new one till the root node, updating the values $Q(s,a)$, $N(s)$ and $N(s,a)$ on each node.

N-Tuple Bandit Evolutionary Algorithm (NTBEA)

The N-Tuple Bandit Evolutionary Algorithm (NTBEA), firstly introduced in (Kunanusont et al. 2017), is an EA that was designed for a noisy domain task which one fitness evaluation may not provide an accurate value. In many cases instead it is approximated by averaging multiple evaluations (*resampling*). All of our selected GVGAI games, and both controllers, are stochastic, which makes the evaluation of the individuals extremely noisy. Therefore, a fast and noise-robust EA is necessary in parameterization.

Table 1: Defender's Parameter Space

Name	**Min—Max—Step**	**Size**
BSPEED	0.1—0.9—0.2	5
ASPEED	0.2—1.0—0.2	5
SUPSPEED	0.05—0.45—0.2	3
APROB	0.01—0.05—0.01	5
AMPROB	0.05—0.25—0.1	3
SLOWPPROB	0.05—0.25—0.05	5
FASTPPROB	0.3—0.5—0.2	2
ADDSUP	1—5—1	5
ACOOLDOWN	2—10—2	5
PCOOLDOWN	5—20—5	4
AMCOOLDOWN	5—20—5	4
LOSSCITY	-4—-1—1	4
BLIMIT	5—20—5	4
AREWARD	1—9—2	5
DELAY	0—300—50	7
CLOSE	350—500—50	4
Total search space size		1.08×10^{10}

NTBEA starts by approximating the fitness landscape of the problem by sampling points of the search space as suggested by a multi-armed bandits. An N-Tuple structure is used to store statistical information of each point (evaluated parameter set in this case) in the search space. Each time a point is evaluated, its fitness is added to the landscape model, as well as an estimation for a number of neighbours (two neighbouring solutions are two points in the search space which only differ in the value of one of the dimensions).

The landscape model is stored using N-Tuples, a structure used firstly by S. Lucas in (Lucas 2008). In our work, N-Tuples are a group of lookup tables that store statistics for each dimension. These statistics are used by multi-armed bandits to select which is the next point to sample from the search space. The reader is referred to (Kunanusont et al. 2017) for a full description of this algorithm, but the basic steps starts with building a population randomly. Then evaluate the fitness of each individual and update the statistics in the landscape model. Next, a set of new individuals (neighbours) are generated and their UCB1 (equation 1) are calculated. The neighbours with the highest UCB1 value are then selected as the next individual. The last 3 steps are repeated until the termination condition is reached.

Approach

Game Rules & Search Space

In the original games, benign and malicious sprites are spawned from portals and the portals spawns sprites at a constant rate during the whole game. In order to be able to define a rich search space where interesting solutions can be found, all original portals have been replaced with time-limited spawning portals for this study. Therefore, all portals can spawn sprites during a specific period of the game as determined by two parameters; the first and the last game step the portals are allowed to spawn. Note that the original games are still included in the search space, concretely when these limits are set to 0 and the last game tick, respectively. Tables 1, 2 and 3 show the parameter spaces for the three selected games with their description and possible values these may have. All games have a search space size of $E10$.

Table 2: Waves' Parameter Space

Name	**Min—Max—Step**	**Size**
RSPEED	0.45—2.45—0.5	5
SSPEED	0.5—2.0—0.5	4
LSPEED	0.1—0.5—0.1	5
PSPEED	0.5—1.5—0.5	3
ASPEED	0.05—0.3—0.05	6
ACOOLDOWN	2—14—4	4
RCOOLDOWN	2—14—4	4
APROB	0.01—0.05—0.04	2
RPROB	0.15—0.4—0.05	6
ASPROB	0.005—0.02—0.005	4
SLIMIT	2—10—2	5
SREWARD	1—9—2	5
SPLUS	1—5—1	5
LASERPEN	-4—-1—1	4
APEN	-4—-1—1	4
DELAY	0—300—50	7
CLOSE	350—500—50	4
Total search space size		7.741×10^{10}

Table 3: Seaquest's Parameter Space

Name	**Min—Max—Step**	**Size**
SSPEED	0.05—0.5—1.5	4
WSPEED	0.05—1.5	2
PSPEED	0.05—0.5—1.5	4
DSPEED	0.1—0.9—0.2	5
SHPROB	0.01—0.16—0.05	4
DHPROB	0.005—0.045—0.01	5
WSPROB	0.01—0.1—0.03	4
OFDHPROB	0.05—0.09—0.02	3
TIMERHPLOSS	5—20—5	4
WHPROB	0.005—0.085—0.02	5
WHALESCORE	5—20—5	4
HPPLUS	1—4—1	4
HP	9—33—8	4
MHP	10—40—10	4
DCONS	1—3—1	3
CRLIMIT	1—7—2	4
DELAY	0—200—50	5
CLOSE	200—400—50	5
Total search space size		5.892×10^{10}

Selected Target Functions

All of the selected target functions are positive-definite functions ($f(x) > 0$ for all $x > 0$). These include a linear function (equation 2 with $m \in \{0.2, 0.4, 1\}$), left and right shifted versions of the original sigmoid function (equation 3 with $K_1 = 150, K_2 = 30$ and $K_3 \in \{3, 12\}$), a logarithm (equation 4) and an exponential (equation 5) functions. These were included to see if the EA can react differently with the fast increasing rate in different areas.

$$f(x) = mx \tag{2}$$

$$f(x) = K_1(\frac{1}{1 + \exp(-\frac{x}{K_2} + K_3)}) \tag{3}$$

$$f(x) = 15 \log_2 x \tag{4}$$

$$f(x) = 2^{\frac{x}{70}} \tag{5}$$

Note that these are **ideal** functions for score progression. The objective is for evolution to tune games so actual gameplay by agents generate such score trends. In some cases, these progressions may simply not be possible, but obtaining an approximation to them can also provide the desired player experience in the games under test.

Fitness Calculation

We used a Normalized Root Mean Square Error (NRMSE) to calculate the total deviation between the obtained score and the target function. Suppose that $\hat{s}$ is a set of real score sequence from time step 1 to the last, and $\hat{y}$ is the set of $f(x)$ values for the selected target function with domain set = $1, 2, ..., n$ where n is the last time step, the RMSE value between $\hat{s}$ and $\hat{y}$ can be computed as in equation 6.

$$NRMSE(\hat{s}, \hat{y}) = \frac{\sqrt{\sum_{i=1}^{n} (\hat{y}_i - \hat{s}_i)^2}}{n(\hat{y}_{max} - \hat{y}_{min})} \tag{6}$$

Given a target function $\hat{t}$, an NRMSE $Loss(x)$ function (equation 6) and a game g, each point in the search space $\hat{p}$ parameterizes the game ($g(\hat{p})$). The RHEA agent plays $g(\hat{p})$ and the score observed at every time step is stored in $\hat{s}$. When the game is over, the fitness of $\hat{p}$ is calculated as $1 - Loss(\hat{t}, \hat{s})$, which should be maximized.

Experiments

We ran NTBEA to evolve game parameter sets for Defender, Waves and Seaquest, fitting each of 4 target functions (3 versions of a linear function, a logarithmic function, an exponential function and 2 versions of a shifted sigmoid function), giving $3 \times 7 = 21$ different experimental settings in total. 10 evolutionary runs were executed for each one of these settings, and the outcomes were then averaged. RHEA was the player controller during the parameter evolutionary runs. After that, the best evolved games were validated 10 times each using MCTS. RHEA population size was set at 20, while the individuals' length was set to 10. The mutation rate was set at 0.1, meaning that one gene is mutated on average on each individual. MCTS rollout depth was set at 10 to match the length of the RHEA individuals. Both controllers used the same heuristic function to evaluate states. The score is used as fitness (resp. reward in MCTS) unless the games end with a victory or loss, in which case the reward is 1000 or -1000 respectively. The C value for the UCB1 equation is $\sqrt{2}$ for both the MCTS agent and the NTBEA. The number of neighbours for NTBEA was set to 100.

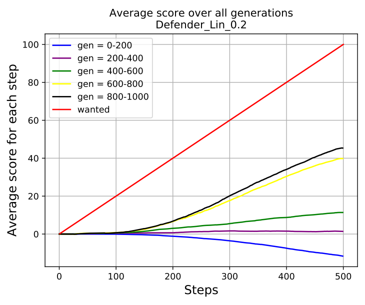
The analysis of the results is presented next, divided into two sections: evolution and validation.

Evolving Game Parameters

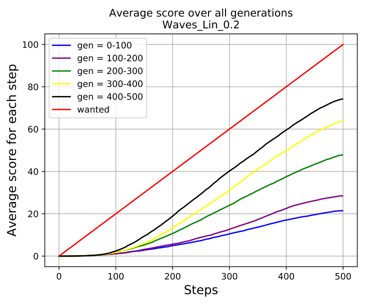
The performance of NTBEA for game parameter evolution is reported using average score for each time step in different generation ranges (as in Figure 1). The result analysis is done in between games and between target functions.

Different Games We compared the results between the three games to fit the same target function. Only $y = 0.2x$ is selected here since it was the easiest function to fit.

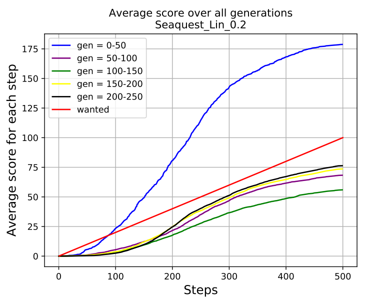
The score trends (Figure 1) shows the averaged values for score per game tick on different generation ranges. All target lines are plotted in red. The final generations shown for each game are different just for visualization purposes as different games converged at different speeds. The first 50 generations for Seaquest produced a score trend (blue line) above the target score line by a significant gap, and managed to find



(a) Defender score

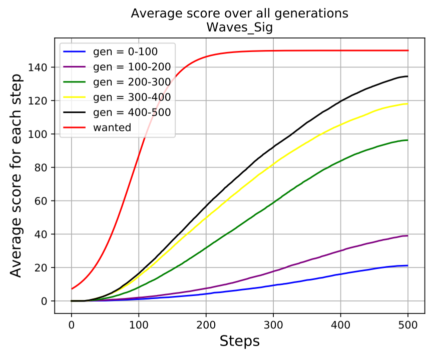


(b) Waves score

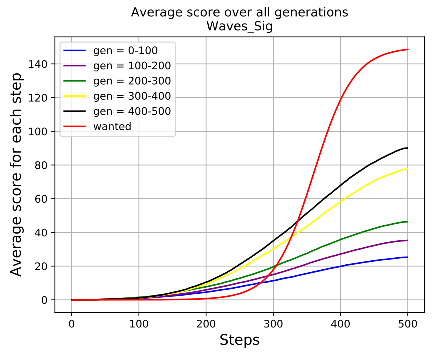


(c) Seaquest score

Figure 1: Average score trend throughout evolution for the linear function $y = 0.2x$ on the games tested in this study. Each plot shows different trends, averages taken at different generation ranges through evolution. Trends for later generations tend to be more fitted to the target function.



(a) Score trends for left-sigmoid



(b) Score trends for right-sigmoid

Figure 2: Average score trends for the shifted sigmoid function in Waves.

the parameter sets that fit the target line more (black line). In contrast, the initial (random) parameter set for Waves provided a score trend below the target line, but it was able to progress during evolution to fit the linear target (Figure 1b). Finally, the starting score trend in Defender (Figure 1a) was negative (i.e. losing points), but NTBEA was able to find parameters for a positive linear-like trend after generation 800.

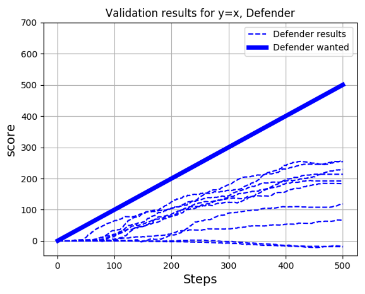
These results show that NTBEA is able to find game parameters that fit a linear score trend in the three games tested. It's worthwhile highlighting that the final score trends never fit perfectly the targets, but in many cases these targets are practically impossible to achieve: they serve as mere guidelines for evolution to make progress in the desired direction.

Functions and Parameters We compared the results from fitting different target functions, by pairing the same types with opposite trends. For the sake of space, only results from the shifted sigmoid functions in Waves are included in this paper. The average score in later generations for the left sigmoid (Figure 2a) case is higher than the right sigmoid (Figure 2b) one at step 300. As it would be expected, the blue curves in both cases are very similar, showing that the initial (random) parameter sets in both scenarios provide a similar score trend. However, NTBEA managed to evolve games with different score trends in the end, in **both** directions (more score events at the beginning versus the end). This suggest that NTBEA and the approach we are proposing is general and adaptable to different target functions from a single starting point.

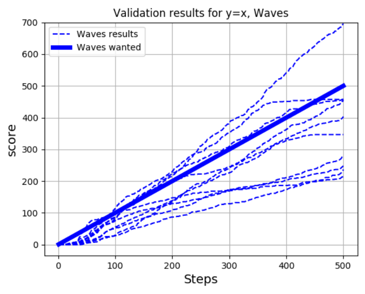
Validating Evolved Games

To validate the results obtained by NTBEA, the best individual of each run was selected and played for 10 times using MCTS. We analyze here the results using the same comparison criteria as in the previous subsection.

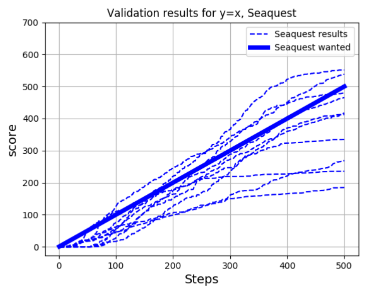
Different Games Figure 3 shows the score trends of the best individuals found in all evolutionary runs for the three games. Most of the parameter sets evolved for Defender (Figure 3a) provided an environment for positive score on average, which is a very relevant improvement considering that most of the initial parameter sets gave negative score. None of them, however, achieved a linear positive trend with $m = 1$, which seems to be difficult to achieve in this game. Waves and Seaquest evolved parameter sets gave better results, with points progression more similar to the target line. We can see that NTBEA was capable of tuning parameters for these games at least in for this function, with the final outcomes of the evolution still provide the game scoring environment in the similar trend as the target functions.



(a) Defender score



(b) Waves score



(c) Seaquest score

Figure 3: Average score trend on validation for $y = x$ on the games of this study.

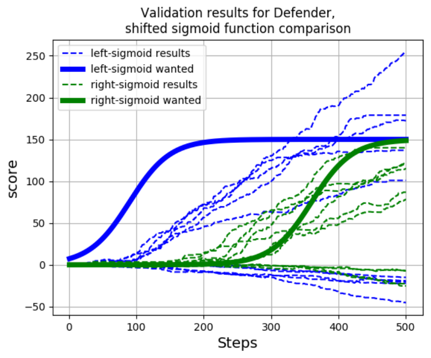


Figure 4: Average score trend on validation for Defender, shifted sigmoid target functions

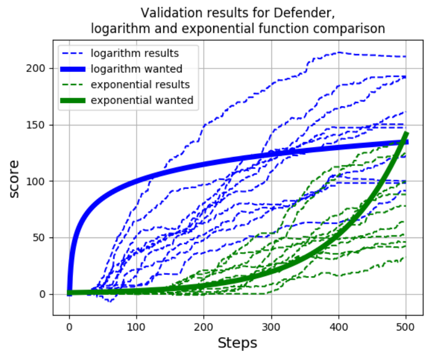


Figure 5: Average score trend on validations for the game Defender, logarithm and exponential target functions.

Functions and Parameters We center this discussion on the sigmoids and logarithmic-exponential function results from the game Defender, which NTBEA seems to struggle more to achieve the desired score trends. We expected that the games evolved to fit the left sigmoid function would provide more score in the beginning and then stop later on. Based on the blue dotted curves in Figure 4, the score seems to increase in a more linear trend for this game after step 100. In contrast, results from fitting right sigmoid function (green dotted curves) look more similar to the target curve. NTBEA is more successful in evolving games to fit right shifted sigmoid. It is possible that it is more challenging to stop providing the score than to suppress it in the beginning and start producing score events later.

Figure 5 shows the results of average score for Defender in logarithm-exponential function comparison. For logarithm fit (blue dotted curves), the final individuals struggle to achieve a quick increase on score at the beginning of the game. It is worth noticing that the target values of the few early steps are hardly achievable, as it requires to achieve 35 points after only 5 game steps. In contrast, green dotted curves which shows an exponential function seems easier to achieve, and results show that it is possible to evolve parameters so score events are rare at the beginning of the game but more common towards the end.

In general, we conclude that the evolved games show the desired trends up to certain degree, with some particular individuals deviating from the target occasionally.

Evolved Games

We observed the final parameters evolved by NTBEA in Defender and compared them within the same group of target functions. For linear functions, we compared $m = 0.2$ and 1 and found that the game with $m = 1$ tended to have slower bomb speed and alien spawn probability. In right-shifted sigmoid final parameter set, alien speed and spawning probability are higher, while the portal opened later and closed earlier compared with the left-shifted version. In logarithmic/exponential comparison, the portal opened and closed later in exponential version, with faster alien. All games evolved maximized the reward given when an alien is shot while minimized the penalty of a city being bombarded.

Examples of the evolved games in Defender can be found in an online video[1]. This video shows one game from each analyzed set (linear with $m = 2$ and 1, left/right shifted sigmoids and logarithm/exponential). In the videos, it is noticeable the differences in supply speed/spawning rate and numbers of enemies spawned, as well as the lack of enemies at the beginning of the exponential-function-evolved game.

[1]https://youtu.be/GADQLe2TiqI

Conclusions and Future Work

The work presented in this paper proposes the usage of N-Tuple Bandit Evolutionary Algorithm (NTBEA) as a way to tune game parameters. The objective is to tweak games so they produce different target scoring trends. This can be seen as one of the possible ways to tune player experience in a given game. Three games from the GVGAI framework are used (Defender, Seaquest and Waves) and their parameters are tuned to fit 5 different target functions. Two agents are employed for play-testing: Rolling Horizon EA (RHEA) during evolution, and Monte Carlo Tree Search (MCTS) to validate the evolved games. Results show that NTBEA has the ability to evolve in a highly noisy environment toward different target trends and for different games, and its effectiveness is shown consistently on each experiment. Additionally, these validation results also show that the games evolved did not overfit to RHEA, as a different agent was able to show a similar behaviour in most cases. The experiments described here suggest that NTBEA is sufficiently robust to the agent used in playtest, as both controllers reproduce similar score trends during evolution and validation.

In fact, Defender has shown to be the most challenging game to tune for all target functions. Random parameter sets for this game typically produced scenarios in which the agents finished with negative score, but NTBEA was able to modify the game experience in order to achieve most of the target functions. It's possible that longer evolution or different settings for NTBEA could have landed better results in Defender, which is left as future work in this area.

References

Ashlock, D. 2010. Automatic Generation of Game Elements via Evolution. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 289–296. IEEE.

Browne, C., and Maire, F. 2010. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games* 2(1):1–16.

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2014. A Survey of Monte Carlo Tree Search Methods. 4(1):1–43.

Cook, M., and Colton, S. 2011. Multi-faceted Evolution of Simple Arcade Games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 289–296. IEEE.

Gaina, R. D.; Lucas, S. M.; and Pérez-Liébana, D. 2017a. Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing. In *2017 IEEE Conference on Evolutionary Computation (CEC)*. IEEE.

Gaina, R. D.; Lucas, S. M.; and Pérez-Liébana, D. 2017b. Rolling Horizon Evolution Enhancements in General Video Game Playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE.

Goertzel, B., and Pennachin, C. 2007. *Artificial General Intelligence*, volume 2. Springer.

Isaksen, A.; Gopstein, D.; Togelius, J.; and Nealen, A. 2015. Discovering Unique Game Variants. In *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*.

Kunanusont, K.; Gaina, R. D.; Liu, J.; Perez-Liebana, D.; and Lucas, S. M. 2017. The N-Tuple Bandit Evolutionary Algorithm for Automatic Game Improvement. In *IEEE Proceedings of the Congress on Evolutionary Computation (CEC)*.

Liu, J.; Togelius, J.; Pérez-Liébana, D.; and Lucas, S. M. 2017. Evolving Game Skill-Depth using General Video Game AI Agents. In *IEEE Proceedings of the Congress on Evolutionary Computation (CEC)*.

Lucas, S. M. 2008. Learning to play Othello with N-tuple Systems. *Australian Journal of Intelligent Information Processing* 4:1–20.

Nelson, M. J., and Mateas, M. 2007. Towards Automated Game Design. In *Congress of the Italian Association for Artificial Intelligence*, 626–637. Springer.

Perez, D.; Samothrakis, S.; Lucas, S.; and Rohlfshagen, P. 2013. Rolling Horizon Evolution versus Tree Search for Navigation in Single-player Real-Time Games. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 351–358. ACM.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016. General Video Game AI: Competition, Challenges and Opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Sorenson, N., and Pasquier, P. 2010. Towards a Generic Framework for Automated Video Game Level Creation. *Applications of Evolutionary Computation* 131–140.

Togelius, J., and Schmidhuber, J. 2008. An Experiment in Automatic Game Design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, 111–118. IEEE.