Illuminating Game Space Using MAP-Elites for Assisting Video Game Design

Martin Balla [1] and **Adrián Barahona-Ríos** [2] and **Adam Katona** [3] and **Nuria Peña Pérez**[4] and **Ryan Spick** [5]

Abstract. In this paper we demonstrate the use of Multi-dimensional Archive of Phenotypic Elites (MAP-Elites), a divergent search algorithm, as a game design assisting tool. The MAP-Elites algorithm allows illumination in the game space instead of just determining a single game setting via objective based optimization.

We showed how the game space can be explored by generating a diverse set of game settings, allowing the designers to explore what range of behaviours are possible in their games.

The proposed method was applied to the 2D game Cave Swing. We discovered different settings of the game where a Rolling Horizon Evolutionary Algorithm (RHEA) agent behaved differently depending on the selected game parameters. The agent's performance was plotted against its behaviour for further exploration, which allowed visualizing how the agent performed with selected behaviour traits.

1 Introduction

In the context of video games, the search space can be seen as the game space, defined by all the possible combination of game parameters (such as gravity, distance between objects, force applied when jumping, etc.). Search algorithms aim to find optimal game parameters for a particular game evaluation function (fitness function) by exploring the game space. The solution to the search problem is therefore a parameter combination that gets the highest score from the fitness function. However, while search algorithms traditionally focus on finding the best combination of game parameters, finding a set of diverse solutions that lead to "good" games can also be interesting, especially from a design point of view.

Recent literature has explored the game space to find different game variants. Isaksen et. al. used Monte Carlo Tree Search (MCTS) to automatically test the difficulty of various points in the game space [7]. The authors used a player model based on human motor skills (precision, reaction time and actions per second), allowing to retrieve a certain point in the game space linked to the desired difficulty level. Few other studies have focused on finding game variants associated to different agent behaviours instead of level difficulties. Tremblay et. al. compared MCTS with different search algorithms (A* and Rapidly-exploring Random Trees) to explore the player trajectories in platform games [16]. Different game levels were hand crafted and several game solutions were found for each single level. Game space has also been explored to find unique game variants by clustering the behaviours found for an agent playing different game levels [6]. However, these methods do not allow for automatic exploration of different playing behaviours by searching the game space.

Procedural Content Generation (PCG), in the context of video games, refers to the use of software to automatically generate content [14]. Content can be in the form of, for instance, game assets (such as textures, 3D models, etc.) sound, dialog trees or mechanics. Given its programmatic nature, this technique can accelerate the video game development cycle, reducing the development cost. Considering the increasing cost of game development [15], PCG is a very attractive solution, especially for bigger open world games. PCG can also be used to assist creativity, helping developers to generate new ideas faster.

This work proposes the use of a search algorithm to automatically look for multiple, high performing game variants based on different desired user-defined playing behaviour. The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm [11] can be used to illuminate the search space. In this way, different variants of the same video game (with different parameters) can be played, resulting in diverse ways of playing the game (different behaviours). In this work a Rolling Horizon Evolutionary Algorithm agent (RHEA) was used to evaluate each point in the game space, obtaining its performance and behaviour with that particular set of parameters. This allowed to map the game space to the desired set of features in the behaviour space. By using MAP-Elites to search the game space, designers could define the behaviour characterization, which results in a diverse set of games, where different play styles are required to achieve the game's objective. In this work we tested MAP-Elites on the game Cave Swing, where we found various levels, requiring many different play styles.

2 Background

2.1 Optimization algorithms

Optimization algorithms have been traditionally used to find the best solution of a given parameter set. To determine the best solution, a fitness function is required, which can be the score in the game or some designed heuristics. As most games have a high number of parameters, it is usually impractical to find the best combination by manual tuning. Parameters can be discrete, where a pre-defined set of values are used, or continuous, where the values are arbitrary within a defined range. Most optimization algorithms require the user to define discrete values, such as Grid Search or NTBEA [10]. Continuous parameter optimization is more complex as more combinations arise. Popular continuous optimization algorithms are Random Search and

[1] Queen Mary University of London, United Kingdom, email: m.balla@qmul.ac.uk
[2] University of York, United Kingdom, email: ajbr501@york.ac.uk
[3] University of York, United Kingdom, email: ak1774@york.ac.uk
[4] Queen Mary University of London, United Kingdom, email: n.penaperez@qmul.ac.uk
[5] University of York, United Kingdom, email: rjs623@york.ac.uk

CMA-ES [5]. One advantage of MAP-Elites is that it can be used for both discrete and continuous optimization problems.

2.2 MAP-Elites

MAP-Elites is a divergent search algorithm which belongs to a family of algorithms called Quality Diversity (QD) [13]. The goal of Quality Diversity algorithms is to find a diverse set of high quality solutions, instead of a single best solution. By optimizing for both performance and diversity, these algorithms often outperform purely objective based optimization by avoiding getting stuck in local optima. Quality Diversity algorithms rely on a user defined Behaviour Characterization (BC). This BC is used to measure similarity between solutions, which allows the algorithms to directly search for diversity. The BC function is domain dependent. In the maze solving domain [9], where a robot have to navigate in a maze, the BC can be the trajectory taken by the robot, or the final position of the robot. In a six legged robot walking domain [2] the BC can be defined by how much each leg touches the ground. While some QD algorithms like Novelty Search with Local Competition [9] focuses more on quality, MAP-Elites puts more emphasis on diversity. This is achieved by considering multiple dimensions of behaviours separately, instead of just calculating the distance between two behaviours. This property makes MAP-Elites especially useful for exploring the search space, by discovering all kind of different behaviours.

A recent work by Gravina et al. [4] surveys QD algorithms in the context of game design. MAP-Elites have been used for designing levels in Super Mario Bros [17], Dungeon levels [1], Bullet Hell Simulation [8] and to balance player decks in Hearthstone [3]. Our work differs from these methods by tuning the game's parameters directly, not just the parameters of the level generator or using MAP-Elites to output raw levels. We also visualize the behaviour map in the form of heatmaps and agent trajectories to get a better insight of the resulting behaviours.

2.3 Cave Swing

Cave Swing is a tap timing game that consists of travelling along a cave of certain width and height by shooting a rope that can anchor to specific locations. A run of the game is successful when an agent manages to travel all the way along the cave in a certain amount of time while avoiding accidentally hitting any of the borders of the map (see Figure 1).

Only two actions are available for this game, a "null" action and "shooting" action, which throws the rope so that it attaches to the nearest available anchor location. Once a rope is anchored, the agent remains hanging from it until another shooting action takes place. The physics of the game are relatively simple, with the movement of the hanging agent depends on both the pulling force exerted by the rope and an external force. The rope is modelled as an elastic of zero natural length, so its pulling force is determined by its stiffness k. This force is multiplied by a loss factor (see Table 1 for information regarding all game parameters). The external force acting on the agent can be picture as a gravity (G) that pulls the agent on both the horizontal (G_x) and vertical directions(G_y). The rope can only attach to the anchor locations, which are placed at a certain height depending on the map dimensions.

The score of this game is calculated for each time frame, and is therefore available at all times during a run of the game (see Figure 1). The calculation of the score is based on how much the agent has progressed in the x direction, how high it is on the vertical direction and how fast it is completing the level. This is defined in equation 1, for each given time t.

$$Score = xP_x + yP_y - tP_t \tag{1}$$

Where x and y are respectively the horizontal and vertical position at any given time t (in game ticks). Px, Py and Pt are the points for x and y positions and the cost per time spent. If the agent succeeds in the run, it will add to this score a positive bonus, but if it fails a penalty will be subtracted from its score. The values of all the costs are defined in Table 1.

2.4 Rolling Horizon Evolutionary Algorithm

To evaluate the different levels, an agent was required, which played reasonably well. We chose a Statistical Forward Planning agent called Rolling Horizon Evolutionary Algorithm (RHEA)[12].

At each step RHEA constructs a random sequence of actions, which is executed in the forward model (given the current state and an action returns the next state) and a score from the reached state is calculated. The action sequence gets mutated and evaluated again. This process is repeated until a time or an iteration budget is elapsed, and then the first action of the highest-scoring action sequence gets executed. A shift buffer is used, which allows the agent to keep the previously evolved sequence, execute only the first action, shift every element by one position to the left and fill the last position by a random element. We avoid total random mutation, which means that every element of the list cannot be mutated more than once per mutation. The parameters used for the RHEA agent in this experiment can be found in Table 2. The fitness function in our case is the numerical score that Cave Swing provides.

Table 1: Game Parameters. Fixed parameters are those that were kept as a default value. These includes the score-related parameters and parameters that could affect the selected behaviour features. Explored parameters are those that were tuned by the MAP-Elites.

Fixed Parameters	**Default Value**	
Map Width	2500	
Map Height	250	
Anchors Height	100	
Maximum Ticks	500	
Points per x	1000	
Points per y	1000	
Cost per Tick	10	
Success Bonus	-10	
Failure Penalty	1	
Explored Parameters	**Min. Value**	**Max. Value**
Number of Anchors	5	20
Gravity in x	-1	1
Gravity in y	-1	2
Rope Stiffness	0.005	0.1
Loss Factor	0.99	0.99999

3 Methodology

3.1 Experimental Procedure

The code used in this paper is available on Github at: https://github.com/martinballa/MAPElitesCaveSwing. Our implementation has two main parts: The MAP-Elites algorithm and the

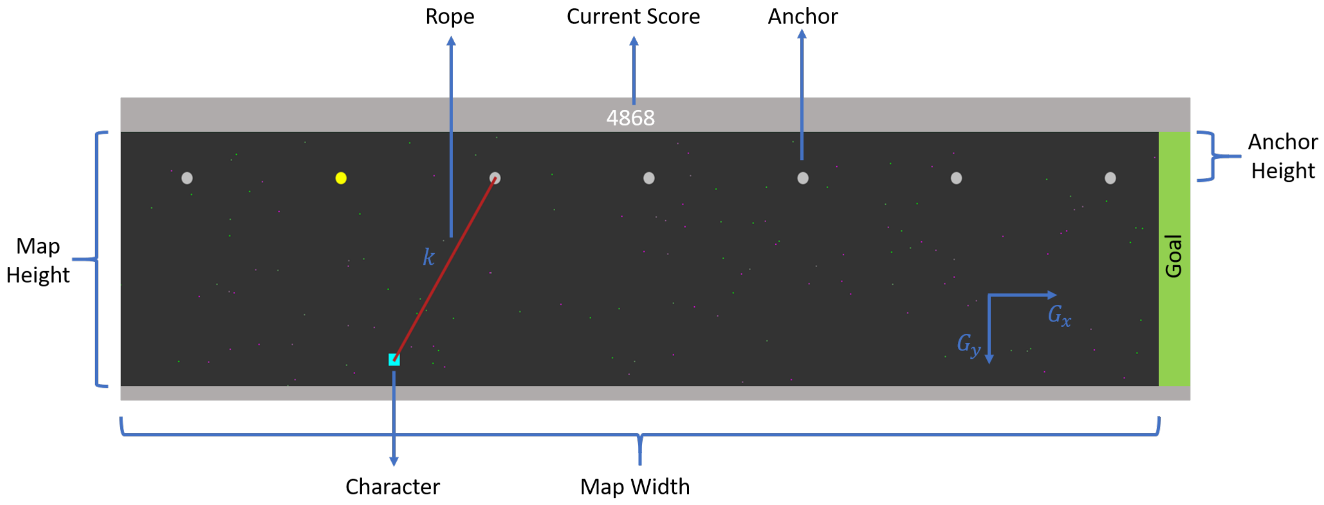


Figure 1: Schematics of the game *Cave Swing*. The agent controlling the character travels along the cave hanging from a rope of stiffness k, which can only attach to the anchors. G_x and G_y are the horizontal and vertical components of the external gravity force acting on the agent. The game finishes if the agent hits any of the grey borders of the map or exceeds a certain amount of time, giving a failure result; or if the agents reaches the goal, giving a successful result.

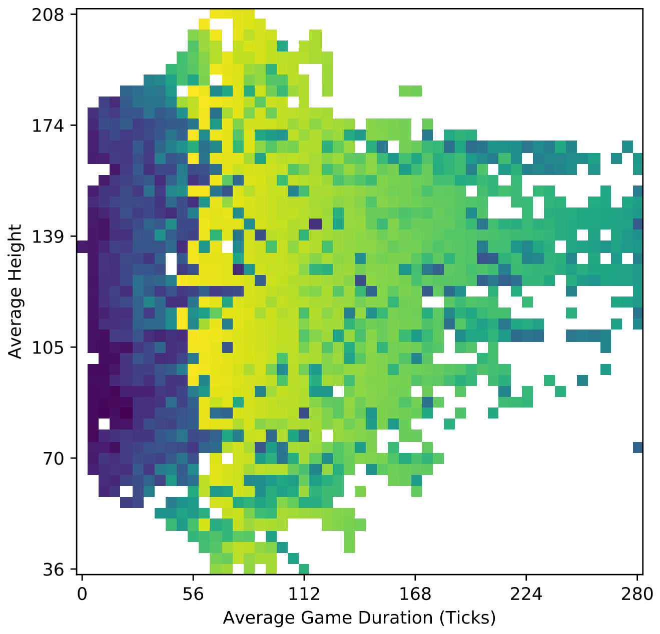


Figure 2: Average game ticks (X axis) against average game height (Y axis). The intensity of the heatmap represents the score achieved by the agent, with the given parameter set (the brighter, the higher scores were achieved by the agent). White regions correspond to areas where no behaviours were found (i.e. either impossible to achieve or not explored by MAP-Elites in 10,000 iterations).

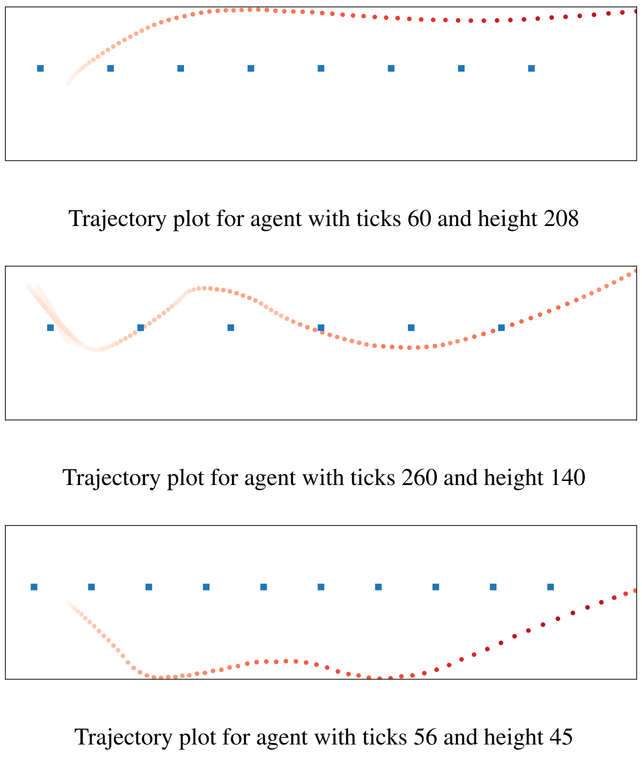
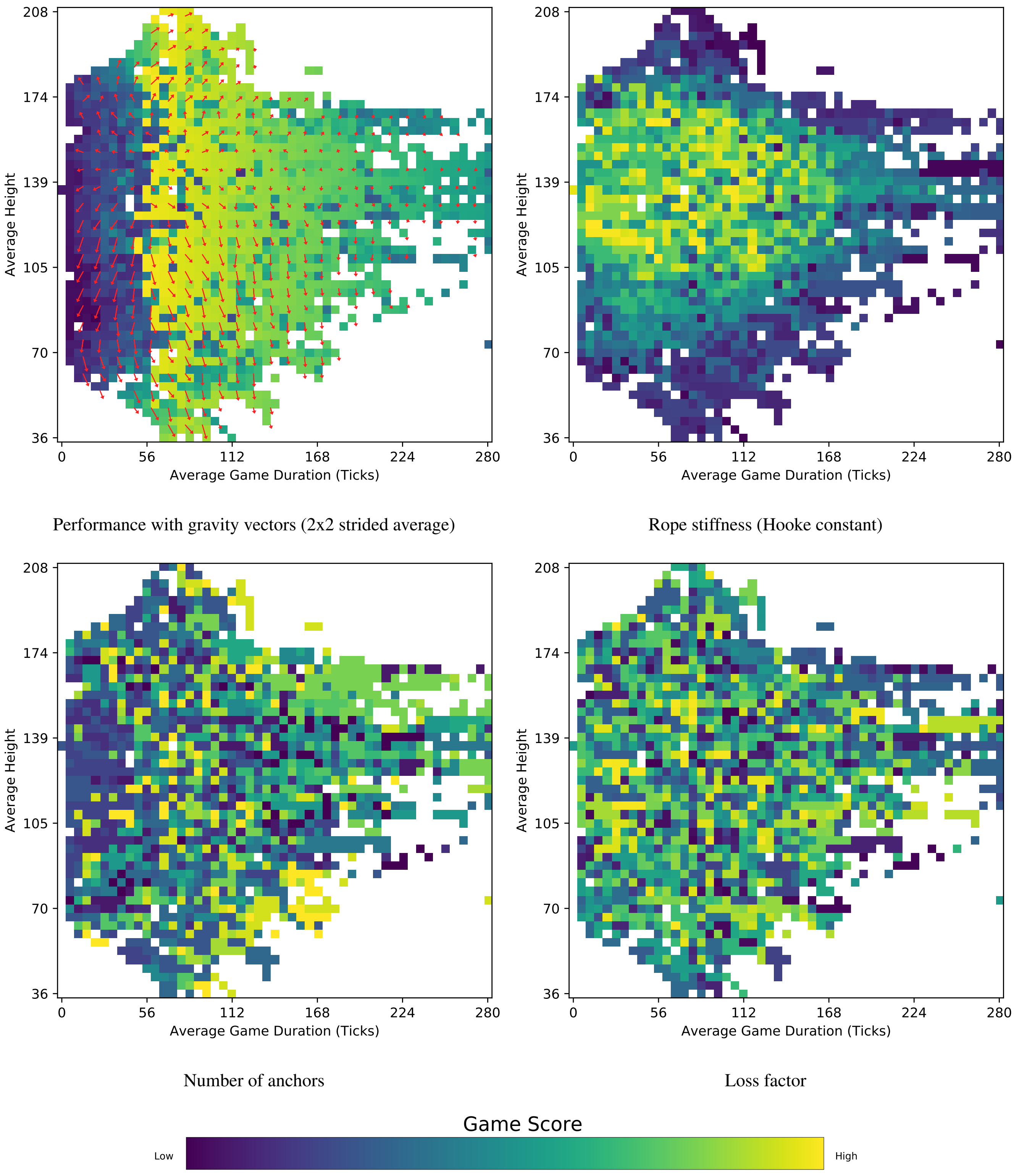


Figure 3: Agent behaviour plots for parameters from specific cells chosen from the behaviour map. Intensity of scatter plot signifies the speed at which the agent was traveling.

Cave Swing with the RHEA agent. For each iteration, the MAP-Elites algorithm picks a set of game parameters, which gets submitted to the game where it gets evaluated. Cave Swing is a fully deterministic game, but the agent relies on random mutations and random initial actions, which make its performance different from run to run. To get a better estimate of the agent's performance the game is played 20 times in our experiments and the collected statistics are averaged. From the collected statistics the MAP-Elites algorithm determines the position of the given game parameters in the map and updates it. We used $10,000$ iterations for the algorithm and a 50 x 50 grid for



Performance with gravity vectors (2x2 strided average)

Rope stiffness (Hooke constant)

Number of anchors

Loss factor

Figure 4: Average game ticks (X axis) against average game height (Y axis). The heatmap represents the performance of the agent with the gravity gradient represented by arrows visualizing the intensity and direction of gravity (3a), the rope stiffness (given by the Hooke constant)(3b), the number of anchors (3c) and loss factor (3d)

the map in this paper.

Each cell in the map produced by the algorithm belonged to a different behaviour. The horizontal and the vertical dimensions were determined by the value of the chosen BC. For demonstrating the effectiveness of our approach, we selected the following behaviours

Table 2: RHEA Parameters. All of these parameters were fixed for the experiment.

RHEA Parameters	Default Value
Sequence Length	200
Number of Evaluations	20
Mutation Rate	10
Shift Buffer	true
Total Random Mutation	false

for further analysis:

1. **Game duration**, which was calculated as the average number of ticks per game run.
2. **Average Height** (vertical position averaged over the run) of the agent in pixels throughout the evaluation runs.

Some parameters of Cave Swing would modify the score function used for evaluation, to avoid this, we fixed those parameters. We decided to also fix the agent's parameters as it would result in additional stochasticity in the evaluations. Please refer to table 1 regarding which parameters were explored.

Table 3: parameters used for the MAP-Elites algorithm

MAP-Elites parameters	
Mutation rate	4
Initial random evaluations	1000
Total evaluations	10000
Grid size	50 x 50

3.2 Data Analysis

The resulting output of $10,000$ iterations of the MAP-Elite algorithm (with 1000 of these being initial random permutations) was computed and the corresponding data collected.

The behaviour maps obtained were explored visually using heatmap style graphs. Each cell of the heatmap contains the parameter set for that agent's evaluation in the game. If a behaviour could be found for that set of parameters, the cell contains a coloured value, which shade was changed depending on the performance value being represented. Those cells where a behaviour could not be found remained in white. This allowed to easily identify regions of high versus low performance within the behaviour space with a contrasting gradient of colours.

Moreover, in order to facilitate the exploration of the found behaviours, a graphical interface was designed by making each of these heatmaps interactive. In this way, each of the cells in the map return its associated parameters by clicking them and the agent playing that particular game variant is shown.

4 Results

The proposed MAP-Elites algorithm successfully found relationships between the parameter space and several points in the behaviour space, with a 57.6% of the cells containing a solution. Note that this percentage depends on the selected ranges of the BC and the number of bins.

Figure 2 shows that the agent's performance distribution depends on both of the behaviour features selected. In this map, the horizontal axis represents the average duration of the game and the vertical axis represents the average height. The brighter the value, the better the performance.

By visually inspecting Figure 2, it can be observed that for short game duration, the performance tended to be very poor. In general, this correspond to runs when the agent did not manage to succeed in the game and the failure penalty was applied. Interestingly, the region determined by medium game duration seems to show the best performance values with longer runs of the game being more detrimental to the agent's performance. This region of medium game duration and high performance values is also related to a larger range of average height values for the agent. Specially for longer games, the variability of the average height reduces considerably.

Figure 3 shows the game simulations corresponding to the sets of parameters associated to 3 different cells in the behaviour map. As it can be observed, the trajectory of the agent in Figure 2b shows an oscillating behaviour as the agent travelled up and down around the anchors and therefore achieved a medium average height. This behaviour led to a long game duration. Figure 2a shows a very different behaviour. The trajectory of the agent was restricted to the top of the map and the duration of the game was relatively short. The agent seemed to have gained momentum, propelling itself towards the top of the map and being able to travel at fast speed. Figure 2c shows a similar behaviour, however the agent propelled itself to the bottom of the map. In order to understand how each of the five explored parameters affected the behaviour space four different heatmaps were built (see Figure 4).

Figure 3a adds to the previously described behaviour-performance relationship the corresponding direction and magnitude of the gravity force. This figure suggests that the average height has a dependency on the gravity direction, as high average height values correspond to gravity values that would have pulled the agent upwards and low values correspond to gravity values pulling the agent downwards. This could explain why the agent propelled itself downwards or upwards in Figures 2a and 2c. Visual inspection of Figure 3a also suggests that long games correspond to points in which the magnitude of the gravity was very low, which would have difficulted the agent's horizontal movement and decreased its height range.

Figures 3b to 3d use the color intensity to represent the value of the parameter selected instead of the performance.

Figure 3b suggests that the stiffness of the rope also played a relevant role in determining the agent's behaviour. Elite solutions with high rope stiffness were only found for low to medium game duration and medium average height, but more diverse solutions were found for more elastic ropes.

Figure 3c displays the relationship between the number of anchors and the behaviour map and Figure 3d presents the relationship between the different values of the loss factor and the behaviour space. It can be observed that the agent's behaviour does not seem to depend on the value of these parameters.

5 Discussion

In general, the obtained results show that it is possible to explore the behaviour space by using MAP-Elites. Visualization of the data seems to be quite useful to explore which parameters are giving place to interesting behaviours and how well a certain agent is able to perform in these conditions. In this way, using MAP-Elites can be useful for game designers to find many variants of a game/level. Instead of just highlighting the best solution, where the agent scores the highest, this method illuminates many different variants. The variants can be visualized, depending on their behaviour features, which would allow game designers to explore all the possible behaviours their game could achieve.

The game selected for this work (Cave Swing) is relatively simple and is governed by non-complex physics. Therefore, not many behaviour emerge from the AI. This game was therefore ideal for a proof-of-concept but the proposed method could be more interesting with more complex games, where the relationship between the game parameters and the agent's behaviour is less clear. For this behaviour characterization only 2 features were used at a time, but MAP-Elites is not limited to having a maximum number of features. As the number of features increase the visualization becomes more complex, but it is still possible to represent it as done by Cully et al [2].

The used agent for the evaluation had a fixed goal determined by the game's score, so the agent did not have any reason to follow a particular play style, that would not result in a high-score. A better evaluation would be to try either more agents or change the objective of the agent. Our method could be used to tune the agent and fix the level, or tune both simultaneously.

6 Conclusion and Future Work

This work presents an exploration of the use of MAP-Elites to a relatively low-parameter game space in order to discover interesting combinations of game parameters. The parameters were evaluated by a RHEA agent by trying to achieve the highest score in the produced game. The visualization of the data collected through the use of the MAP-Elites algorithms shows a high level of detail when comparing the features of the game space, which we chose to optimize for the grid of the chosen behaviour values for both average height and game duration. Furthermore, a plot of the agent playing a game with a chosen set of behaviours (speed/ticks) was also presented to understand how the game parameters affect the agent's movement. In this work we have shown the MAP-Elites can be effectively used to explore the parameters of a game. In this paper we used a simplistic game as a proof-of-concept and a Statistical Forward Planning agent, but any parameterised game with a reasonable agent could be used for this purpose.

As future work, the parameter tuning could be extended to tune more parameters in the game and also the agent to play the game. With MAP-Elites we were able to find parameters for many diverse levels, which could be used to optimize agents to play well a large set of different levels. As Cave Swing is fairly simple, applying MAP-Elites to more complex games would be more interesting, which could result in more complex behaviours.

ACKNOWLEDGEMENTS

This work was supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1].

REFERENCES

[1] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, 'Empowering quality diversity in dungeon design with interactive constrained map-elites', in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, (Aug 2019).

[2] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret, 'Robots that can adapt like animals', *Nature*, **521**(7553), 503, (2015).

[3] Matthew C. Fontaine, Scott Lee, L. B. Soros, Fernando De Mesentier Silva, Julian Togelius, and Amy K. Hoover, 'Mapping hearthstone deck spaces through map-elites with sliding boundaries', in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO 19, p. 161169, New York, NY, USA, (2019). Association for Computing Machinery.

[4] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, 'Procedural content generation through quality diversity', in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, (Aug 2019).

[5] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos, 'Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)', *Evolutionary computation*, **11**(1), 1–18, (2003).

[6] Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen, 'Discovering unique game variants', in *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*, (2015).

[7] Aaron Isaksen, Daniel Gopstein, and Andrew Nealen, 'Exploring game space using survival analysis.', in *FDG*, (2015).

[8] Ahmed Khalifa, Scott Lee, Andy Nealen, and Julian Togelius, 'Talakat: Bullet hell generation through constrained map-elites', in *Proceedings of The Genetic and Evolutionary Computation Conference*, pp. 1047–1054, (2018).

[9] Joel Lehman and Kenneth O Stanley, 'Evolving a diversity of virtual creatures through novelty search and local competition', in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218. ACM, (2011).

[10] Simon M Lucas, Jialin Liu, and Diego Perez-Liebana, 'The n-tuple bandit evolutionary algorithm for game agent optimisation', in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–9. IEEE, (2018).

[11] Jean-Baptiste Mouret and Jeff Clune, 'Illuminating search spaces by mapping elites', *arXiv preprint arXiv:1504.04909*, (2015).

[12] Diego Perez, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen, '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*, GECCO '13, pp. 351–358, New York, NY, USA, (2013). ACM.

[13] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley, 'Quality diversity: A new frontier for evolutionary computation', *Frontiers in Robotics and AI*, **3**, 40, (2016).

[14] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N Yannakakis, 'What is procedural content generation?: Mario on the borderline', in *Proceedings of the 2nd international workshop on procedural content generation in games*, p. 3. ACM, (2011).

[15] Julian Togelius, Noor Shaker, and Mark J Nelson, 'Procedural content generation in games: A textbook and an overview of current research', *Togelius, N. Shaker, M. NelsonBerlin: Springer*, (2014).

[16] Jonathan Tremblay, Alexander Borodovski, and Clark Verbrugge, 'I can jump! exploring search algorithms for simulating platformer players', in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, (2014).

[17] Vivek R Warriar, Carmen Ugarte, John R Woodward, and Laurissa Tokarchuk, 'Playmapper: Illuminating design spaces of platform games', in *2019 IEEE Conference on Games (CoG)*, pp. 1–4. IEEE, (2019).

Demo: Bonobo QA: Using Adversarial Agents to Aid Game AI Development

Nathan John
School of Electrical Engineering and Computer Science
Queen Mary University of London
London, UK
n.m.john-mcdougall@qmul.ac.uk

***Abstract*—This is a proposal describing a demo of Bonobo QA, which is a work in progress tool and method for debugging game AI using neuroevolution.**

I. Introduction

When considering the problem with the testing of game AI systems, the tools available to QA testers, designers and programmers are fairly limited. The fact that for modern games, an AI agent must gracefully handle a wide variety of situations. To ensure that the agents handle these situations correctly, QA testers need to invest a lot of time into checking as many scenarios as they can.

When looking at traditional automated test types such as unit tests, input-output tests and integration tests, they are not appropriate for two reasons. Firstly, the rapid iterations and changes that happen when developing games can mean that the tests will break often, slowing down development time. Secondly, in order to cover all of the situations and scenarios needed a very large number of tests would be necessary.

Ideally, tools for testing would be robust enough to survive the shifting development landscape, and flexible enough to find abnormal behaviours without requiring a large number of specifically written tests. Inspiration for such a tool could be drawn from monkey testing, where randomly generated inputs are used to check the behaviour of a software system. However, for most games, the action space is so large that randomly generated inputs will not explore the areas of interesting behaviour.

Bonobo QA could be considered an extension of monkey testing, using neuroevolution to guide the random inputs towards exploring actions that are interesting in the context of the game being tested. With a designer, programmer or QA tester observing the playthroughs created by the system, they will be able to make inferences about how their AI agents are performing, and potentially spot exploits or weaknesses in their implementations. The benefits of this are twofold. Firstly, the exploration is not constrained to testing thing that the tester has thought of. Secondly, being able to watch playthroughs while not dealing with playing the game frees up the mental load of the tester to just observe behaviours.

II. Current Design

In order to use Bonobo QA, there are a few requirements for the game being tested. Firstly, the game needs to support multiple instances of the game being run at the same time. In the current implementation, this is done through the use of each game instance being a self-contained Unity prefab.

In principle, this could be accomplished with a machine farm with instances of the game streaming video and performance data back to the Bonobo QA master.

Secondly, an interface needs to be written that allows an agent to be controlled by an evolved neural network. The inputs to the network, and how the network's outputs are converted into game actions will vary from game to game. The final piece of game-specific setup is needed in the design of a fitness function to evaluate the quality of the games.

For this demo there are two games that are shown, Ping - an air hockey-like game - and DropFeet - an implementation of the cult two-button fighting game Divekick. For Ping, the inputs are the normalised direction to the ball, the normalised direction to the opponent, and the ball's X and Y speed. The outputs are the desired X and Y movement direction. For DropFeet, the inputs consist of normalised direction to the opponent, and the opponent's foot hitbox, the opponent's normalised velocity, and switches for both ourselves and our opponent being on the floor, or being in a kick state. The outputs used as boolean values for the jump and kick buttons.

With this setup complete, Bonobo QA can instantiate a set of game instances of the game, and evaluate their state after a period of time has passed, using NEAT as the underlying neuroevolution algorithm. For example, in the current implementation there is a 10 second evaluation period before the fitness function is applied to the game's states. Figure 1 shows the visualisation of Bonobo QA. You can observe the entire population of game instances, and can also see a visualisation of the neural network being used in the background. The ability to observe the entire population at once allows the user to quickly get a feel for the entire population of strategies that are active. This overview is important for getting a feel for potential problems.

While observing the overall population, the user can click on any individual instance to focus on it. This makes the selected instance full screen, allowing the user to focus their attention on one particular behaviour. The user can also pause the evaluation process, to allow time to run debugging tools to further interrogate an abnormal behaviour. Additionally, the user can override the automatic evaluation to manually select

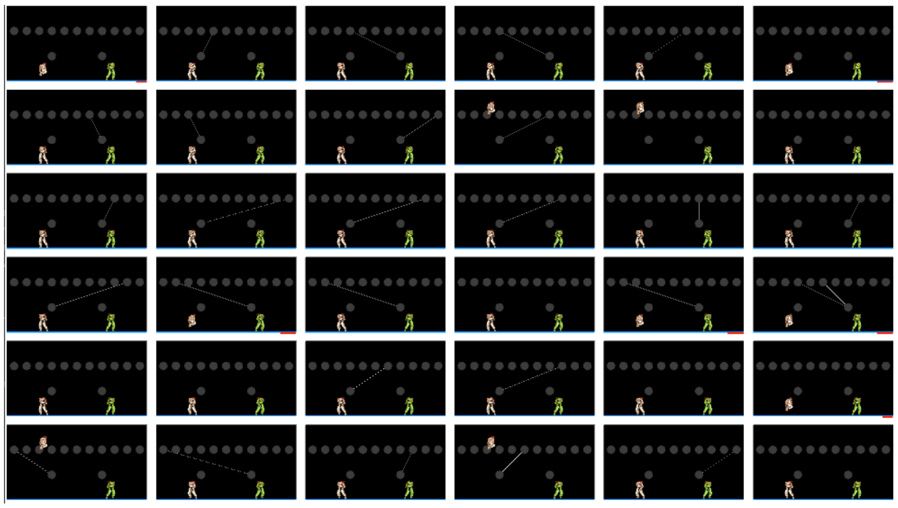


Fig. 1. Bonbobo QA running the simple fighting game DropFeet

potentially interesting behaviours, and guide the evolutionary process.

During the demo, players will be able to play against an authored AI for the two described games, and then can watch a Bonobo QA evolutionary process begin to exploit the AIs.

Weighting NTBEA for Game AI Optimisation

James Goodman and **Simon Lucas** [1]

Abstract. The N-Tuple Bandit Evolutionary Algorithm (NTBEA) has proven very effective in optimising algorithm parameters in Game AI. A potential weakness is the use of a simple average of all component Tuples in the model. This study investigates a refinement to the N-Tuple model used in NTBEA by weighting these component Tuples by their level of information and specificity of match. We introduce weighting functions to the model to obtain Weighted-NTBEA and test this on four benchmark functions and two game environments. These tests show that vanilla NTBEA is the most reliable and performant of the algorithms tested. Furthermore we show that given an iteration budget it is better to execute several independent NTBEA runs, and use part of the budget to find the best recommendation from these runs.

1 Introduction

In Game AI, as in many other fields, algorithms usually have several parameters that need to be specified. For any given problem some parameter settings may give good results, while other settings give very poor results. For any new problem (a new game for example) we need to decide on which parameter values to use. In many cases a set of 'standard' parameter settings are available based on previous work, but these may not be ideal for the new domain. An exhaustive search of all possible parameter settings is usually unfeasible - it may take days of processing time on a large parallel cluster to train a complex neural network using Reinforcement Learning (RL). If the RL algorithm has four parameters, each of which can have five values, then training a policy under each possible setting will take 5^4 = 625 cluster-days, or about 2 cluster-years to evaluate each. The problem is considerably worse if the outcome of any one evaluation (or experiment) is stochastic, so that a good estimate of the value of a given parameter setting requires many independent evaluations.

The field of parameter (and hyper-parameter) optimisation seeks fast methods for deciding on parameter settings in a new domain with an available computational budget. This generally involves constructing a predictive model for the result of a future untried evaluation. After each time-consuming real-world evaluation has been run, the computationally cheap predictive model is updated with the result and interrogated to suggest the next set of parameter values to try. By reducing the number of expensive full evaluations to find a good (if not necessarily optimal) set of parameters, we save significant time and money.

The N-Tuple Bandit Evolutionary Algorithm (NTBEA) was introduced in [8, 11]. It has been benchmarked against several other optimisation algorithms in stochastic game environments and proven to be more effective at finding a good set of parameter settings than other algorithms within a fixed computational budget [10]. Similarly [13] find NTBEA is the best optimiser of a number tried modify MCTS parameters during algorithm execution for a number of games.

NTBEA in [10, 11] estimates the value of a set of parameter values using the simple average of all matching Tuples in the model (see Background for a detailed explanation). The current work extends this to weight the matching Tuples using the amount of data (i.e number of real-world experiments) that inform a given Tuple, and the degree of specificity of the Tuple match. We hypothesise that this approach will allow us to converge to a good parameter setting faster and more robustly than vanilla NTBEA.

In addition to introducing Weighted-NTBEA in this work, we also modify four benchmark tests from the function optimisation literature to incorporate noise. These enable optimisation algorithms to be compared cheaply (in terms of computational budget) and also provide greater confidence in conclusions because the true underlying value is known exactly, and is not an estimate over multiple expensive evaluations.

2 Background

2.1 Black-box optimisation

Black-box function optimisation addresses the problem of finding the optimal value of some $f(\theta)$

$$y = \max_{\theta} f(\theta) \quad \theta \in \mathbb{R}^d \tag{1}$$

where $f(\theta)$ can be evaluated at any θ, but not differentiated. When $f(\theta)$ is expensive to evaluate we wish to minimise the number of evaluations we make and can use the real evaluations made so far to model the result of $f(\theta)$ (the 'response surface') to decide what value of x should be evaluated next. A common approach is to use Bayesian optimisation techniques with a prior over the response surface, and update a posterior model after each evaluation. To pick the next point a trade-off is made between exploitation and exploration; for example the point with the largest expected improvement (EI), or the highest 95% confidence bound (UCB) [3, 7, 12]. Bayesian methods require either a model to be specified, or a decision on the kernel functions to use in a (non-parametric) Gaussian Process. They are sensitive to stochastic noise, especially noise that is highly non-Gaussian [3]. Approaches exist to integrate different types of noise into the model, but these add complexity to the model [12].

Most Bayesian methods and libraries assume that θ is continuous in all dimensions d, and do not work in discrete spaces. This is not true for all, for example BOCS [2] uses Bayesian Linear Regression with semi-definite programming to optimise a discrete combinatorial problem. However, BOCS does assume uniform Gaussian noise. Other approaches have been used to model the response surface in

[1] Queen Mary University of London, Game AI Research group, email: james.goodman@qmul.ac.uk

black-box optimisation: Random Forests are used in the SMAC algorithm [6].

In a bandit-based approach, each setting of the parameters is one 'arm' of the bandit, and we seek to find out which 'arm' gives us the highest reward in a limited number of pulls. This is a natural fit if each θ_i can take a small number of discrete values, but it cannot cope easily with continuous dimensions.

NTBEA combines a bandit-based approach with an N-Tuple model [8] and an evolutionary algorithm to select the next point to be evaluated. The UCB1 (Upper Confidence Bound) algorithm is used to balance exploration and exploitation [1]. NTBEA is described in detail in the next section.

2.2 NTBEA

This explanation of NTBEA closely follows [11]. During each iteration of NTBEA we:

1. Run a full game (or experiment, or other expensive function evaluation) using the current test setting $\boldsymbol{\theta}$. For the first iteration $\boldsymbol{\theta}$ is selected at random.
2. Update the N-Tuple Model with the evaluation result.
3. Generate a neighbourhood of points by applying a mutation operator to $\boldsymbol{\theta}$ (repeat X times to get a neighbourhood of size X).
4. Evaluate the Upper Confidence Bound (UCB) for each of the N points using the N-Tuple Model. Select the one with the highest UCB as the new $\boldsymbol{\theta}$, and repeat from 1.

In this study, as in [8, 10, 11] we set X=50, and the mutation operator used is to randomly mutate each θ_i to a random setting with probability $\frac{1}{d}$, always mutating at least one θ_i.

2.2.1 *N-Tuple Model*

An 1-Tuple model breaks down the modelled $f(\boldsymbol{\theta})$ into d components, where $\boldsymbol{\theta} \in \mathbb{R}^d$ using Equation(2). Each component i is the expected value of f assuming that only θ_i affects the value. If $\theta_i = x$, this is the mean of all evaluation results so far where $\theta_i = x$. In (2), $\mathbf{1}(\theta_i = \phi_i)$ is a delta-function that is 1 when a previously evaluated $\boldsymbol{\phi}$ matches with the current $\boldsymbol{\theta}$ setting in the ith dimension, N is the total number of previous evaluations, and $f_{obs}(\boldsymbol{\phi}_m)$ is the mth of these. M_i is the number of evaluations that match with tuple i.

$$\hat{f}(\boldsymbol{\theta}) = \frac{1}{d}\sum_{i=1}^{d}\frac{1}{M_i}\sum_{m=1}^{N}\mathbf{1}(\theta_i = \phi_i) f_{obs}(\boldsymbol{\phi}_m) \tag{2}$$

$$\text{where } M_i = \sum_{m=1}^{N}\mathbf{1}(\theta_i = \phi_i) \tag{3}$$

In other words, our prediction $\hat{f}(\boldsymbol{\theta})$ is the average of all the d matching 1-Tuple predictions based on past observations. There are no interactions between different parameters, and there are no assumptions about relationships between different values of a given parameter. For example, if one parameter has discrete values 1, 2 or 3 then the result of evaluations where this was 1 or 3 will have no impact at all on predictions for the intermediate 2. This is a very conservative non-parametric model. In the case of 5 dimensions with 10 possible values for each, we need to maintain just 50 sets of statistics for a 1-Tuple model (M, the number of times each tuple-setting has been tried, and $\bar{f}$, the mean of these evaluations). Any $\boldsymbol{\theta}$ will match with exactly five of these, and $\hat{f}(\boldsymbol{\theta})$ is the mean of these five.

A 2-Tuple model extends this to consider interactions between two parameter settings. We replace $\mathbf{1}(\theta_i = \phi_i)$ in (2) with $\mathbf{1}(\theta_i = \phi_i, \theta_j = \phi_j)$, and now consider all evaluations that were a match on two different parameters. In the case of 5 dimensions with 10 possible values for each this gives a total of $\binom{5}{2} \times 10 \times 10 = 1000$ distinct 2-Tuples for which N and $\bar{f}$ are maintained. Any $\boldsymbol{\theta}$ will match with exactly $\binom{5}{2} = 10$.

In all the experiments in this study, as in [8, 10, 11] we use 1-Tuples, 2-Tuples and d-Tuples in the model. A d-Tuple matches on all parameters, so is unique for each $\boldsymbol{\theta}$. The predicted value $\hat{f}$ of the model for any new $\boldsymbol{\theta}$ is the arithmetic mean across *all* matching tuples.

2.2.2 *UCB*

The UCB1 algorithm [1] calculates a probable upper bound on the true value J of the 'arm' of a bandit $\boldsymbol{\theta}$, given the data observed so far using (4). N is the total number of trials of the bandit, and $n(\boldsymbol{\theta})$ is the number of times this 'arm' has been pulled (i.e. the number of times that $\boldsymbol{\theta}$ has been evaluated).

$$J(\boldsymbol{\theta}) = \hat{f}(\boldsymbol{\theta}) + k\sqrt{\frac{\log N}{n(\boldsymbol{\theta})}} \tag{4}$$

The N-Tuple model uses equation (2) to calculate $\hat{f}(\boldsymbol{\theta})$, but we still have the second term of equation (4) that controls exploration. We can calculate this for each individual tuple, with N equal to the total number of NTBEA iterations, and $n(\boldsymbol{\theta})$ equal to the number of these for which the tuple matches $\boldsymbol{\theta}$ in (2). NTBEA calculates the second term for each matching tuple, and then takes the arithmetic average. There is one additional nuance that some tuples will never have been evaluated, and formally (4) will return ∞ in this case. To avoid this an additional hyper-parameter ϵ is added, so that

$$J(\boldsymbol{\theta}) = \hat{f}(\boldsymbol{\theta}) + k\sqrt{\frac{\log N}{n(\boldsymbol{\theta}) + \epsilon}} \tag{5}$$

In this study, as in [8, 10, 11] we set $\epsilon = 0.5$. The value of k needs to be scaled to the range of $f(\boldsymbol{\theta})$, and is set for each domain (see Method section).

3 Hypothesis

Vanilla NTBEA estimates the value of a parameter setting $\boldsymbol{\theta}$ as the simple arithmetic mean of all the matching Tuples in the model that match. For example if we have five parameters and are using 1-, 2- and N-Tuples then any $\boldsymbol{\theta}$ will have one matching d-Tuple (where $d = 5$), five matching 1-Tuples and $\binom{5}{2} = 10$ matching 2-Tuples. The statistics gathered for each of these 16 Tuples is then averaged. The same approach applies to calculating the exploration estimate using (4). Even if we have evaluated a specific $\boldsymbol{\theta}$ multiple times, the results from those evaluations still only comprise $\frac{1}{16}$ of the NTBEA estimate; $\frac{5}{16}$ always comes from the matching 1-Tuples. Our hypothesis is that NTBEA will better estimate the value of a parameter setting if it applies greater weight to the more specific tuples as the number of evaluations increases. In the limit of a large number of evaluations of a specific $\boldsymbol{\theta}$, then only the statistics from the fully-matching d-Tuple should be relevant.

We propose four distinct weighting schemes, which vary in the rate of decay in the influence of less-specific tuples. In all cases the

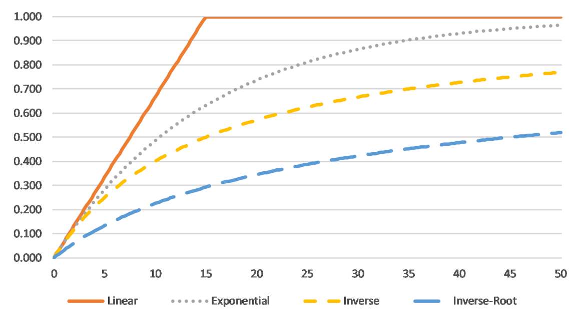


Figure 1: The four weighting functions used in Weighted-NTBEA. The x-axis is $n(\theta)$ the number of evaluations that match the tuple, and the y-axis is the weighting applied to the tuple between 0 and 1. The remainder of the value is calculated from the average of the next level of tuples. $T = 15$ in all cases.

value V of a parameter setting $\boldsymbol{\theta_N}$, with N different parameters is

$$V(\boldsymbol{\theta}_N) = wTP(\boldsymbol{\theta}_N) + (1-w)\frac{1}{|\boldsymbol{\theta}_{N-1}|}\sum V(\boldsymbol{\theta}_{N-1}) \tag{6}$$

where $TP(\boldsymbol{\theta}_N)$ is the average value from the N-Tuple statistics of $\boldsymbol{\theta}_N$ and $w \in [0,1]$ is the weight used for the N-Tuple statistics. The remaining 1-w weight is applied to the average of all (N-1)-Tuples, i.e. all Tuples on the next level down. In (6), $|\boldsymbol{\theta}_{N-1}|$ is a slight abuse of notation and refers to the number of such Tuples. In the case that no Tuples are held at the N-1 level, then this descends to the next level for which we do have Tuples in the NTBEA model. Note that (6) is recursive, and each of the $V(\boldsymbol{\theta}_{N-1})$ terms is calculated from weighting its Tuple statistics, $TP(\boldsymbol{\theta}_{N-1})$, with a sum over $V(\boldsymbol{\theta}_{N-2})$ at the next level down.

In our $N = 5$ example vanilla NTBEA always weights the 5-Tuple, the ten 2-Tuples and the five 1-Tuples at $\frac{1}{16}$ each. Using (6) this weighting will change as we gain more information. With no evaluations, the w for any Tuple will be 0.0, and as the number of evaluations for a Tuple increases we want w to increase towards a maximum of 1.0 so that asymptotically we ignore information from lower-level Tuples.

The four weighting schemes use linear, inverse, inverse square-root and exponential decay functions.

1. Linear
$$w = \min\left(\frac{n(\theta)}{T}, 1.0\right) \tag{7}$$
2. Inverse-root
$$w = 1.0 - \sqrt{\frac{T}{n(\theta)+T}} \tag{8}$$
3. Inverse
$$w = 1.0 - \frac{T}{n(\theta)+T} \tag{9}$$
4. Exponential
$$w = 1.0 - \exp\left(\frac{-n(\theta)}{T}\right) \tag{10}$$

These functions are sketched in Figure 1. They have the desired properties that $w = 0$ when $n(\theta) = 0$ (when no evaluations have been conducted that match the tuple), and $w \to 1$ as $n(\theta) \to \infty$. They differ in the rate at which this decay happens, which in all cases must be parameterised by some T. The Linear decay is most draconian, and will ignore any information from lower level tuples once $n \geq T$, while under Inverse-root decay lower-level tuples have a residual weight of 0.71 after T evaluations. For all experiments in this study we set $T = 15$. This is somewhat arbitrary, but scaled to be about 5% of the total iterations in the smallest experiments with a budget of about 300 NTBEA iterations.

4 Method

We apply each of the decay functions (7), (9), (8), 10) to a number of different optimisation problems to determine whether our hypothesis holds and the modified model does converge faster and more robustly than vanilla NTBEA. By using a number of different problems we seek to test that any improvement generalises, and is not specific to one domain. A secondary goal is exploratory, to see if the four different weighting functions have varying patterns of performance.

4.1 Benchmark functions

We test on four benchmark functions from the global optimisation literature [4,7]. These are interesting non-convex functions for which we can calculate the true value, and hence judge the performance for the NTBEA variants. Some amendments are needed to the original functions:

1. These are all deterministic functions with no noise. To convert them to a stochastic win/lose setting appropriate for a game benchmark we convert the function value to a probability p of a +1 score (a 'win'), and a 1-p probability of a -1 score (a 'loss').
2. They are continuous functions in all dimensions. We discretise by taking values at equally spaced intervals for each dimension.
3. Global optimisation seeks to minimise a function. To maximise we multiply by -1.

We outline the four functions below. A complete description is in [4].

- *Hartmann-3*. A three-dimensional function with four local optima. Two of these optima are close in value, with one slightly higher. In the original problem the output range is [0.0, 3.59], so we divide by 4.0 to get a p value between 0 and 1. We split all three dimensions into ten equally spaced discrete values, for a total parameter-space size of 1000 with a true value $p \in [0, 0.897]$.
- *Hartmann-6*. A six-dimensional function with a similar four optima to *Hartmann-3*. We apply the same modifications as with *Hartmann-3*, and discretize each dimension into five equally spaced values, for a parameter-space of size 15,625 with $p \in [0, 0.737]$.
- *Branin*. A two-dimensional function with three global maxima at 0.4. We split each dimension into 20 equally spaced intervals to get a parameter-space of 400. We add 10 to the result, divide by 12 with a floor at 0 to get to a valid range for $p \in [0, 0.795]$. In this case only 14.8% of the 400 points are non-zero.
- *Goldstein-Price*. A two-dimensional function with one global maximum, and several local ones. We split each dimension into 20 equally spaced intervals to get a parameter-space of 400. We add 400 to the result, divide by 500 with a floor at 0 to get to a valid range for $p \in [0, 0.794]$. 13.3% of the 400 points are non-zero.

In all cases we try each weighting function, plus vanilla NTBEA on each benchmark function with 300, 1000 and 3000 iterations. For each setting we run NTBEA 1000 times, and record the estimated p value (by NTBEA) of the finally selected θ and the actual p value. In NTBEA we use $k = 1$ for the exploration constant in (5).

Parameter	Planet Wars I	Asteroids I	Planet Wars II	Asteroids II
Sequence Length	5, 10, **15**, 20, 25, 30	5, 10, 15, 20, 50, **100**, 150	7, 10, 13, 16, 20, 25, 30	50, 75, 100, 125, 150, 200
Mutated Points	0, 1, 2, **3**	0, 1, **2, 3**	1, 2, 3, 5, 10, 15, 20	1, 2, 3, 5, 10, 20, 30, 50
Resample	**1**, 2, 3	1, 2, 3	**1**, 2, 3	1, 2, 3
Flip One Value	false, **true**	false, **true**	false, true	false, true
Use Shift Buffer	false, **true**	false, **true**	false, true	false, true
Mutation Transducer	false	false	false, true	false, true
Repeat Prob.	-	-	0.2, 0.4, 0.6, 0.8	0.2, 0.4, 0.6, 0.8
Discount Factor	1.0	1.0	1.0, 0.999, 0.99, 0.95, 0.9	1.0, 0.999, 0.99, 0.95, 0.9
Parameter Space size	228	336	23,520	23,040

Table 1: Parameter space for RHEA in Planet Wars and Asteroids game experiments. The first two columns for the I experiments are as in [10]. The optimal values found for the games in that paper and in [11] are in bold.

4.2 Game Parameters

Lucas et al. 2019 [10] compare NTBEA against several other popular optimisation algorithms in two games; Planet Wars and Asteroids. They optimise a Rolling Horizon Evolutionary Algorithm (RHEA) to find the best setting to win the 2-player Planet Wars (+1 for a win, and -1 for a loss), and also to obtain the highest score in 2000 game-ticks in the 1-player Asteroids. For comparable results we use exactly the same games and settings. In Planet Wars we use $k = 1$ for the exploration constant in (5), and $k = 5000$ for Asteroids.

In Planet Wars each player has a number of planets which generate ships at a constant rate. Players send ships from a planet to invade another, and to win the game they must conquer all planets. In Asteroids the player controls a ship which can rotate and shoot to destroy surrounding asteroids. Points are gained for shooting asteroids, and if one collides with the player then a life is lost; after three lost lives the game ends. The details of the gameplay are not central to this study, and more details can be found in [10, 11].

RHEA is optimised over five parameters in [10], which are listed in Table 1. Each optimisation algorithm was permitted 288 evaluations in Planet Wars, and 336 in Asteroids. This allowed Grid Search to run one game for each parameter setting. We repeat these experiments up to 100 times for each game and each weighting function. We record the parameter setting that is chosen each time. To get a good estimate of the actual value of the 288 and 336 possible settings it is feasible to run 1000 games for each setting of Planet Wars and 500 for Asteroids, although this takes 6 days to run for Asteroids, illustrating the value of a rapid optimiser.

These small parameter spaces of 228 and 336 have the advantage of permitting a good estimate of the 'best' setting to be found by brute force computation, but they are not representative of larger spaces in real problems. For example when optimising RHEA for a Game of Life variant [9] use NTBEA with 100 evaluations in a space of size 28,800. As a final experimental set we add further parameters to RHEA (discount factor, mutation transducer and repeat probability) from [9], and extend the other parameters to give a larger overall space as detailed in Table 1 in the 'II' columns. These extensions were fixed after seeing the results of the first set of experiments (the 'I' columns) to focus on areas with higher performance. For Planet Wars we increased the concentration of Sequence Length options around the optimal 10-15 range, and in Asteroids we did the same around the optimal 100 value. We also increased the upper range of Mutated Points significantly, especially for Asteroids where the optimal value of 3 was the highest possible.

For these larger parameter spaces we used a budget of about 20,000 total iterations to try different overall approaches:

- 10 runs of 2,000 iterations each
- 3 runs of 7,000 iterations each
- 2 runs of 10,000 iterations each
- 1 run of 20,000 iterations

Given the size of the parameters spaces it was not feasible to estimate an accurate value for all parameter settings. Instead we do this (by running 1000 or 500 games for Planet Wars and Asteroids respectively) for just the settings suggested by any of these runs. The purpose of these experiments is to understand how best to spend an available budget of iterations. Should we use them in a single NTBEA run, or spread them out and then pick the best of the suggestions. This is motivated by an observation from Deep Reinforcement Learning research, in which the random seed can have a major effect on the outcome of the algorithm, and results are often reported using 'best of N' runs [5].

5 Results

5.1 Benchmark functions

Table S1 in the Supplementary Material tabulates the numeric means and confidence intervals for the NTBEA experiments on the four benchmark functions with added noise. Figure 2 displays boxplots of the true value of the NTBEA recommended parameters for each benchmark function and weighting function (1000 NTBEA runs for each, at 300, 1000 and 300 iterations).

- Hartmann-3. The appears to be the easiest of the four functions for NTBEA to optimise, with 300 iterations getting a mean value of 0.862 of a maximum of 0.897 for both Vanilla NTBEA (STD), and the Linear and Inverse-root weighting functions. With 3000 iterations all of the variants obtain a mean score of between 0.88 and 0.89; in all cases 25% to 35% of all runs recommend one of the three top parameter settings with actual values between 0.895 and 0.897
- Hartmann-6. This is harder to optimise with a clear progression as iterations increase from 300 to 3000. Vanilla NTBEA is a clear winner at only 300 iterations, and the Inverse-root and Inverse weighting functions are joint top with the Vanilla version at 3000 iterations (in a parameter space of size 15,625). The Linear weighting function does very poorly in comparison.
- Branin. As with Hartmann-6, Vanilla NTBEA is a clear winner at 300 iterations, and is joint top with the Inverse-root and Inverse weighting functions at 3000 iterations. The parameter space is only 400.
- Goldstein-Price. The same pattern is repeated here. Vanilla NTBEA is best for a small number of iterations, and all except the Linear weighting function are equally good with 3000 iterations to explore a parameter space of size 400.

The key finding is that here vanilla NTBEA ('STD' in Figure 2) is always the best or joint best for any combination of benchmark function and number of iterations, and is particularly effective for smaller numbers of iterations.

5.2 Games

Table 2 shows the results for Planet Wars I and Asteroids I experiments, with 228 and 336 NTBEA iterations on similarly sized parameter spaces. Figures 3 and 4 have box plots for the data. These are averaged over 100 runs for each setting for Planet Wars, and between 62 and 69 runs for Asteroids (the number that completed in an 84 hour window). For Planet Wars vanilla NTBEA gives both the best and most reliable (i.e. lowest standard deviation) results. The Exponential decay variant is the only one to have a performance within the 95% confidence interval of vanilla NTBEA. The single highest parameter setting gives a score of 0.732, with 6 of the 288 settings having a score of 0.65 or higher averaged over 1000 games. Since we have run 1000 games for each of the 288 settings and *then* picked the highest result, the 0.732 will be an over-estimate. Apart from the Linear weighted variant, all algorithms pick one of the top 6 settings between 50% and 60% of the time.

For Asteroids the results are quite similar. Vanilla NTBEA gives the best result with the smallest standard deviation. One of the variants is within the 95% confidence interval, but in this case it is the Inverse weighting function. In both games is is clear, as in the Benchmark Function results, that vanilla NTBEA gives the best recommended parameter setting despite giving a very poor estimate of the absolute value that the recommendation will provide when used.

The 95% confidence intervals in Table 2 are calculated on the basis that the estimated values of each parameter setting are exact. This was true for the benchmark functions in Table S1, but is not true here due to noise in these estimates from averaging across 1000 or 500 independent games. We do not have an estimate of this additional uncertainty.

Encouragingly, we obtain exactly the same the optimal parameter settings for both games as those found in the original work (highlighted in Table 1) [10, 11]. However, we get rather higher values for these in game play. For Planet Wars the original work finds that 288 iterations of NTBEA achieves a score of 0.51 ± 0.01, while we obtain 0.65. In Asteroids the relevant values are $8,760 \pm 40$, against our 9600. The reason for this discrepancy is not clear, but we do not believe it affects the key conclusions of this study.

Table 3 shows the results from the Planet Wars II and Asteroids II experiments with larger, more realistic parameter spaces to explore. There were 142 unique parameter settings recommended by the 150 NTBEA runs for the Planet Wars II experiments and an estimated value for each of these was calculated from averaging 1000 runs of the game. The best estimated scores of the recommended parameter settings have increased to 0.77 compared to the best possible score of 0.73 for Planet Wars I, so the additional parameters enable RHEA to better play the game if we can efficiently explore the space.

For Planet Wars vanilla NTBEA gives the best mean result at 1k iterations, and does not give significantly different results at more iterations (within 95% error bounds). The same caveat applies to these error bounds as in Table 2 as they do not include the additional uncertainty from the average over 1000 runs used to estimate the value of the final parameter settings.

The Inverse-root weighting functions matches vanilla NTBEA at 1k, and at 3k all variants at least match vanilla performance, with the Exponential weighting being the best. These results make clear that there is a high level of uncertainty in any individual NTBEA run. The best of the 20 vanilla runs at 1k gives a parameter setting that scores 0.772 over 1000 games, and the worst scores a mere 0.616. This remains true at 10k and 20k iterations, with three of the 20k runs recommending parameters that score less than 0.7.

Even with a large number of iterations any single NTBEA run may give a relatively poor result. Given a fixed budget of games to optimise a parameter Table 3 suggests that it is not a good idea to put the whole budget into a single NTBEA run. Far better to execute several NTBEA runs with a small number of iterations, and then use the remaining game budget to estimate the true value of each of these and pick the best.

This is reinforced when we look at the Asteroids results in Table 3. Vanilla NTBEA does joint best with 1k iterations, and the mean score does not increase significantly for higher numbers of iterations. At higher iterations all variants except the Linear function are at least as good, but not necessarily reliably better. In the Asteroids case there is an effective maximum score of 10000 when we use 2000 game ticks as here, so with all the mean and best results in the 9700 to 9800 range the optimisation does not have much room to work, especially when we add noise.

6 Discussion

In all four of the benchmark functions, and in both games across small and large parameter spaces vanilla NTBEA is at least as good as the weighting variants tried for small numbers of iterations; and usually better with lower variance in results. As the number of iterations increases this effect shrinks, and for some cases one of the weighting variants can be significantly better. For example Inverse-root with 1000 iterations on the Hartmann-6 function, or the Exponential function with 3000 iterations in Asteroids II. However, this is cherry-picking. Furthermore the weighting variants introduce complexity with a new hyper-parameter T to be specified.

When we optimise an expensive function such as game performance over a parameter space we are deliberately trying to use a small number of iterations. Vanilla NTBEA works best in this situation, and we conclusively reject the hypothesis that improving the N-Tuple model with these weighting functions improves either reliability or performance.

We do not reject the hypothesis that the variants provide a better estimate of the true value of a parameter setting. Across all benchmark functions and game environments vanilla NTBEA provides very poor estimates of the actual value, under-estimating by a very large margin because it is averaging over all possible Tuple matches. The Inverse and Inverse-root weighting functions consistently do a much better job of estimating the value of their recommendation. However, this is not as important when our key objective is to get a good recommendation; we can always go on to get a good estimate of its value later.

Linear weighting is clearly worse than the other options that do not exclude all contributions from less-specific Tuples with more information after only T iterations. The appears to be because once it has T evaluations of a specific setting it ignores all other data, and uses the average of those evaluations. With a larger number of iterations what often happens is that sequential iterations focus on the current best estimate until the mean falls sufficiently and the focus shifts to another setting. With noisy function evaluations this often leads to a recommendation with a smaller number of trials (but more than T), that happens to currently have a high estimate. Hence the recommendation is optimistic because it picks the best (stochastic) estimate

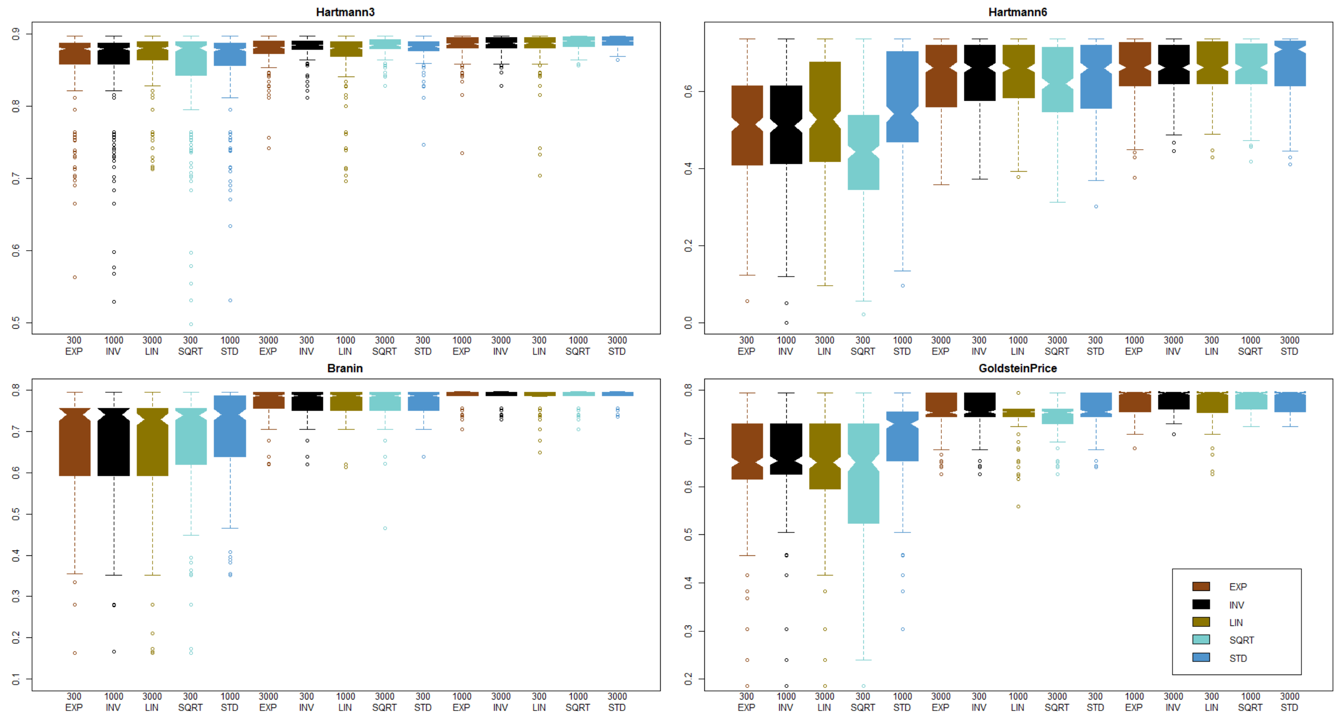


Figure 2: Boxplots for the true Score of settings recommended by NTBEA after 300, 1000 and 3000 iterations in each of the four benchmark functions.

NTBEA	Runs	Iterations	Game	Mean	S Dev	95% Interval		Delta	95% Interval		Top6
STD	100	288	Planet Wars	**0.655**	0.079	0.640	0.671	-0.185	-0.203	-0.167	60%
LIN	100	288	Planet Wars	0.615	0.111	0.593	0.636	0.187	0.164	0.211	44%
INV	100	288	Planet Wars	**0.630**	0.110	0.610	0.656	0.086	0.061	0.107	53%
SQRT	100	288	Planet Wars	0.633	0.097	0.616	0.653	**-0.017**	-0.036	0.001	51%
EXP	100	288	Planet Wars	**0.643**	0.091	0.625	0.663	0.130	0.107	0.151	58%
STD	62	336	Asteroids	**9596**	67	9580	9613	-709	-741	-680	94%
LIN	66	336	Asteroids	**9577**	87	9556	9598	129	104	155	88%
INV	68	336	Asteroids	**9584**	77	9567	9604	**-27**	-52	-3	87%
SQRT	69	336	Asteroids	9563	118	9536	9591	-248	-274	-222	81%
EXP	67	336	Asteroids	9570	82	9552	9590	104	80	125	87%

Table 2: Results for Weighted NTBEA variants with Planet Wars and Asteroids. Mean is the *estimated* value of the final recommended parameter setting from 1000 offline games, with a 95% confidence interval. Delta is the average difference to the NTBEA-estimated value of this point in the N-Tuple model, with a 95% confidence interval. All confidence intervals are calculated with a basic bootstrap. Bold entries indicate the best performing variants (within confidence bounds) for each game. LIN is the Linear weighting function; INV is Inverse, SQRT is the Inverse-root and EXP is the Exponential. STD is vanilla NTBEA. Top6 is the percentage runs that recommended one of the Top 6 parameter settings as estimated from the 1000 games run for each.

across all options with more than T evaluations, and we can see this reflected in the general over-estimate of the value of its recommendation (a version of the 'winner's curse'). This effect is less evident for the other weighting functions, as they never let the weighting of other Tuples fall to zero.

7 Conclusion and Future Work

We hypothesised that adding a recursive weighting function to apply to Tuples in NTBEA would improve performance in parameter optimisation in terms of quality and reliability of a recommended (optimised) parameter setting and in providing a more accurate estimate of the value of this. We tried four different weighting functions with different decay characteristics (linear, inverse, inverse-root and exponential) across four benchmark functions from the function optimisation literature, and two games with two distinct sizes of parameter space.

Across all ten experiments we found no evidence that the proposed weighting functions improved NTBEA except in the least important one of providing a better estimate of the true value of the parameter setting recommended by the optimising process. On the contrary, we found strong evidence that vanilla NTBEA is better able than the weighting function variants to reliably find a higher quality recommendation. This is especially true for the smaller number of iterations that would tend to be used in real world applications.

Finally we investigated how best to use a fixed budget of NTBEA iterations in the Planet Wars and Asteroids games. These showed than any individual NTBEA run may give a poor recommendation,

Game	NTBEA	Iterations	Runs	Best score	Mean	SD	95% Bounds	
Planet Wars	STD	1000	20	0.772	**0.707**	0.045	0.688	0.727
Planet Wars	LIN	1000	20	0.752	**0.679**	0.067	0.652	0.711
Planet Wars	INV	1000	20	0.788	**0.694**	0.070	0.665	0.728
Planet Wars	SQRT	1000	20	0.762	**0.712**	0.035	0.697	0.728
Planet Wars	EXP	1000	20	0.774	**0.681**	0.061	0.656	0.708
Planet Wars	STD	3000	7	0.762	0.709			
Planet Wars	LIN	3000	7	0.762	0.718			
Planet Wars	INV	3000	7	0.762	0.708			
Planet Wars	SQRT	3000	7	0.760	0.714			
Planet Wars	EXP	3000	7	0.774	**0.735**			
Planet Wars	STD	10000	2	0.756	0.717			
Planet Wars	LIN	10000	2	0.748	0.736			
Planet Wars	INV	10000	2	0.756	0.747			
Planet Wars	SQRT	10000	2	0.756	0.740			
Planet Wars	EXP	10000	2	0.770	0.748			
Planet Wars	STD	20000	1	0.708				
Planet Wars	LIN	20000	1	0.640				
Planet Wars	INV	20000	1	0.674				
Planet Wars	SQRT	20000	1	0.732				
Planet Wars	EXP	20000	1	0.632				
Asteroids	STD	1000	20	9815	**9701**	63	9675	9728
Asteroids	LIN	1000	20	9776	9655	89	9617	9694
Asteroids	INV	1000	20	9803	**9706**	70	9690	9722
Asteroids	SQRT	1000	20	9811	**9702**	68	9676	9736
Asteroids	EXP	1000	20	9819	9620	125	9569	9673
Asteroids	STD	3000	7	9804	9707			
Asteroids	LIN	3000	7	9804	9764			
Asteroids	INV	3000	7	9835	**9778**			
Asteroids	SQRT	3000	7	9817	9736			
Asteroids	EXP	3000	7	9818	9758			
Asteroids	STD	10000	2	9705	9705			
Asteroids	LIN	10000	2	9709	9612			
Asteroids	INV	10000	2	9804	9801			
Asteroids	SQRT	10000	2	9817	9814			
Asteroids	EXP	10000	2	9779	9762			
Asteroids	STD	20000	1	9735				
Asteroids	LIN	20000	1	9783				
Asteroids	INV	20000	1	9783				
Asteroids	SQRT	20000	1	9815				
Asteroids	EXP	20000	1	9815				

Table 3: Results for Weighted NTBEA variants with Planet Wars and Asteroids over larger parameter spaces. Mean is the *estimated* value of the final recommended parameter setting from 1000/500 offline games for Planet Wars/Asteroids, with 95% confidence intervals calculated with a basic bootstrap. Bold entries indicate the best performing variants (within confidence bounds) for each game. Best score is the best individual result for any of the runs for that line.

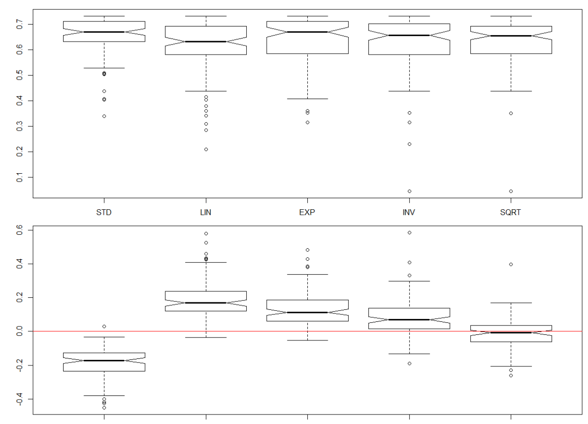


Figure 3: Boxplots for the estimated true Score of settings recommended by NTBEA after 288 iterations in Planet Wars (top), and the Delta of the NTBEA predicted value to this (bottom). The red horizontal line marks a Delta of 0.0, indicating perfect prediction by NTBEA.

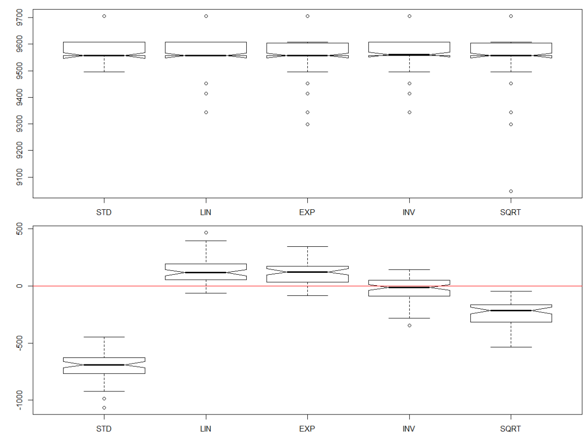


Figure 4: Boxplots settings recommended by NTBEA after 336 iterations in Asteroids. Key as in Figure 3.

and it is better to run several NTBEA runs with a smaller number of iterations, and then use the remaining budget to estimate more accurately the value of these, and then pick the best.

We have not explored different values of T, the hyper-parameter introduced to determine how the weighting function is used, and it is possible that other values may perform better. There are other more adventurous options to improve the N-Tuple model, such as regression across the tuples to determine which ones are important. The updated model in this paper still assumes that each Tuple at a given level is equally important. If we have no data for the full d-Tuple then we average across all matching 2-Tuples, when in practise some of these may be more important than others. One approach to try would be to construct a regression model across the tuples to up-weight the ones that better predict the observed results. We have also not changed the exploration model, which averages across all matching tuples as in vanilla NTBEA. It could be worthwhile to experiment with different noise models, for example using a square root instead of a log function in Equation (4), which has been found useful in other areas where exploration is more important than exploitation [14].

8 Acknowledgments

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/S022325/1.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer, 'Finite-time analysis of the multiarmed bandit problem', *Machine learning*, **47**(2–3), 235–256, (2002).

[2] Ricardo Baptista and Matthias Poloczek, 'Bayesian optimization of combinatorial structures', in *International Conference on Machine Learning*, p. 462–471, (Jul 2018).

[3] Eric Brochu, Vlad M. Cora, and Nando de Freitas, 'A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning', *arXiv:1012.2599 [cs]*, (Dec 2010). arXiv: 1012.2599.

[4] L Dixon and G Szego, 'The global optimization problem: An introduction. vol.2, 1-15', *Amsterdam, Holland*, (1978).

[5] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger, 'Deep reinforcement learning that matters', in *Thirty-Second AAAI Conference on Artificial Intelligence*, (2018).

[6] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *International Conference on Learning and Intelligent Optimization*, p. 507–523. Springer, (2011).

[7] Donald R. Jones, Matthias Schonlau, and William J. Welch, 'Efficient global optimization of expensive black-box functions', *Journal of Global optimization*, **13**(4), 455–492, (1998).

[8] Kamolwan Kunanusont, Raluca D. Gaina, Jialin Liu, Diego Perez-Liebana, and Simon M. Lucas, 'The n-tuple bandit evolutionary algorithm for automatic game improvement', in *2017 IEEE Congress on Evolutionary Computation (CEC)*, p. 2201–2208. IEEE, (2017).

[9] Simon M Lucas, Alexander Dockhorn, Vanessa Volz, Chris Bamford, Raluca D Gaina, Ivan Bravi, Diego Perez-Liebana, Sanaz Mostaghim, and Rudolf Kruse, 'A local approach to forward model learning: Results on the game of life game', in *2019 IEEE Conference on Games (CoG)*. IEEE, (2019).

[10] Simon M. Lucas, Jialin Liu, Ivan Bravi, Raluca D. Gaina, John Woodward, Vanessa Volz, and Diego Perez-Liebana, 'Efficient evolutionary methods for game agent optimisation: Model-based is best', *arXiv preprint arXiv:1901.00723*, (2019).

[11] Simon M. Lucas, Jialin Liu, and Diego Perez-Liebana, 'The n-tuple bandit evolutionary algorithm for game agent optimisation', *arXiv:1802.05991 [cs]*, (Feb 2018). arXiv: 1802.05991.

[12] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas, 'Taking the human out of the loop: A review of bayesian optimization', *Proceedings of the IEEE*, **104**(1), 148–175, (Jan 2016).

[13] Chiara F. Sironi and Mark HM Winands, 'Comparing randomization strategies for search-control parameters in monte-carlo tree search', in *2019 IEEE Conference on Games (CoG)*, p. 1–8. IEEE, (2019).

[14] David Tolpin and Solomon Eyal Shimony, 'Mcts based on simple regret.', in *AAAI*, (2012).

NTBEA	Runs	Iteration	Function	Mean	SD	95% Bounds		Delta	95% Bounds	
STD	1000	300	Hartmann-3	**0.859**	0.051	0.856	0.862	-0.056	-0.061	-0.052
LIN	1000	300	Hartmann-3	**0.862**	0.104	0.855	0.868	0.080	0.074	0.087
INV	1000	300	Hartmann-3	0.855	0.057	0.852	0.859	-0.075	-0.084	-0.066
SQRT	1000	300	Hartmann-3	0.848	0.066	0.844	0.853	-0.235	-0.241	-0.230
EXP	1000	300	Hartmann-3	**0.862**	0.044	0.859	0.865	**-0.003**	-0.012	0.006
STD	1000	1000	Hartmann-3	0.881	0.016	0.880	0.882	-0.075	-0.078	-0.072
LIN	1000	1000	Hartmann-3	0.872	0.032	0.870	0.874	0.087	0.084	0.091
INV	1000	1000	Hartmann-3	0.881	0.016	0.879	0.882	0.046	0.044	0.049
SQRT	1000	1000	Hartmann-3	**0.883**	0.013	0.882	0.883	**0.007**	0.004	0.010
EXP	1000	1000	Hartmann-3	0.879	0.018	0.878	0.880	0.067	0.065	0.070
STD	1000	3000	Hartmann-3	**0.888**	0.009	0.887	0.888	**-0.021**	-0.022	-0.020
LIN	1000	3000	Hartmann-3	0.882	0.022	0.881	0.884	0.042	0.040	0.044
INV	1000	3000	Hartmann-3	0.886	0.010	0.885	0.886	0.031	0.029	0.032
SQRT	1000	3000	Hartmann-3	**0.888**	0.009	0.887	0.888	**0.020**	0.018	0.021
EXP	1000	3000	Hartmann-3	0.885	0.012	0.884	0.886	0.037	0.035	0.039
STD	1000	300	Hartmann-6	**0.551**	0.142	0.542	0.560	-0.127	-0.135	-0.118
LIN	1000	300	Hartmann-6	0.535	0.142	0.526	0.544	0.119	0.107	0.130
INV	1000	300	Hartmann-6	0.516	0.149	0.506	0.525	**0.026**	0.015	0.038
SQRT	1000	300	Hartmann-6	0.461	0.155	0.451	0.471	-0.120	-0.133	-0.108
EXP	1000	300	Hartmann-6	0.526	0.149	0.516	0.535	0.065	0.053	0.077
STD	1000	1000	Hartmann-6	0.633	0.095	0.627	0.639	-0.178	-0.188	-0.168
LIN	1000	1000	Hartmann-6	0.633	0.092	0.628	0.639	0.108	0.100	0.115
INV	1000	1000	Hartmann-6	**0.639**	0.085	0.634	0.645	0.046	0.040	0.052
SQRT	1000	1000	Hartmann-6	0.610	0.101	0.604	0.617	**-0.031**	-0.041	-0.020
EXP	1000	1000	Hartmann-6	0.633	0.092	0.627	0.639	0.082	0.076	0.088
STD	1000	3000	Hartmann-6	**0.666**	0.085	0.661	0.672	-0.202	-0.220	-0.184
LIN	1000	3000	Hartmann-6	0.635	0.104	0.628	0.641	0.100	0.092	0.107
INV	1000	3000	Hartmann-6	**0.666**	0.066	0.661	0.670	0.031	0.027	0.036
SQRT	1000	3000	Hartmann-6	**0.668**	0.057	0.664	0.671	**-0.006**	-0.012	0.000
EXP	1000	3000	Hartmann-6	0.658	0.079	0.653	0.663	0.058	0.054	0.063
STD	1000	300	Branin	**0.705**	0.098	0.699	0.712	**-0.020**	-0.027	-0.013
LIN	1000	300	Branin	0.676	0.142	0.667	0.685	-0.389	-0.398	-0.380
INV	1000	300	Branin	0.670	0.136	0.661	0.679	-0.389	-0.398	-0.380
SQRT	1000	300	Branin	0.460	1.327	0.376	0.544	-0.204	-0.288	-0.120
EXP	1000	300	Branin	0.678	0.126	0.670	0.686	-0.397	-0.405	-0.388
STD	1000	1000	Branin	**0.773**	0.027	0.771	0.775	-0.031	-0.035	-0.028
LIN	1000	1000	Branin	0.768	0.035	0.766	0.771	0.055	0.050	0.060
INV	1000	1000	Branin	**0.772**	0.026	0.771	0.774	**0.015**	0.011	0.019
SQRT	1000	1000	Branin	**0.773**	0.028	0.771	0.775	-0.047	-0.052	-0.042
EXP	1000	1000	Branin	**0.773**	0.029	0.771	0.775	0.041	0.037	0.044
STD	1000	3000	Branin	**0.789**	0.012	0.788	0.790	-0.020	-0.021	-0.018
LIN	1000	3000	Branin	0.781	0.024	0.780	0.783	0.024	0.021	0.026
INV	1000	3000	Branin	**0.789**	0.013	0.788	0.789	0.011	0.009	0.012
SQRT	1000	3000	Branin	**0.788**	0.015	0.787	0.789	**-0.002**	-0.004	0.000
EXP	1000	3000	Branin	0.784	0.020	0.783	0.785	0.019	0.017	0.022
STD	1000	300	GoldsteinPrice	**0.700**	0.076	0.695	0.705	**-0.005**	-0.011	0.001
LIN	1000	300	GoldsteinPrice	0.621	0.145	0.612	0.630	-0.318	-0.328	-0.308
INV	1000	300	GoldsteinPrice	0.622	0.142	0.613	0.631	-0.323	-0.333	-0.313
SQRT	1000	300	GoldsteinPrice	0.622	0.142	0.613	0.631	-0.350	-0.360	-0.340
EXP	1000	300	GoldsteinPrice	0.614	0.142	0.605	0.623	-0.313	-0.323	-0.303
STD	1000	1000	GoldsteinPrice	**0.759**	0.029	0.757	0.761	**-0.017**	-0.021	-0.014
LIN	1000	1000	GoldsteinPrice	0.755	0.035	0.753	0.758	0.071	0.066	0.075
INV	1000	1000	GoldsteinPrice	0.756	0.029	0.754	0.758	0.024	0.020	0.028
SQRT	1000	1000	GoldsteinPrice	0.750	0.030	0.748	0.752	-0.042	-0.046	-0.037
EXP	1000	1000	GoldsteinPrice	0.756	0.032	0.754	0.758	0.057	0.053	0.060
STD	1000	3000	GoldsteinPrice	**0.779**	0.020	0.778	0.781	-0.016	-0.018	-0.014
LIN	1000	3000	GoldsteinPrice	0.775	0.026	0.774	0.777	0.024	0.021	0.026
INV	1000	3000	GoldsteinPrice	**0.780**	0.020	0.778	0.781	0.014	0.012	0.016
SQRT	1000	3000	GoldsteinPrice	**0.778**	0.021	0.777	0.779	**-0.001**	-0.003	0.001
EXP	1000	3000	GoldsteinPrice	**0.778**	0.022	0.777	0.780	0.020	0.018	0.022

Table S1: Results for Weighted NTBEA with benchmark functions. Mean is the *actual* value of the optimised parameters. Delta is the difference between actual and N-Tuple estimated value of these parameters. Bold entries are the best performing variants (within confidence bounds) for each combination of function and number of iterations. LIN is the Linear weighting function; INV is Inverse, SQRT is the Inverse-root and EXP is the Exponential. STD is vanilla NTBEA.

Rolling Horizon Co-evolution in Two-player General Video Game Playing

Charles Ringer and **Georgiana Cristina Dobre** [1] and **Cristiana Pacheco** and **Diego Perez-Liebana** [2]

Abstract. Artificial Intelligence for General Video Game Playing (GVGP) is challenging not only because agents must be generalisable to a range of games, but they must also make decisions within the time constraints of real-time video games. The General Video Game Artificial Intelligence framework (GVGAI) is one of the most popular frameworks for GVGP. Recently, a two-player track has become available where two agents play a game together, either competitively or cooperatively, which poses an additional challenge to agents because they have to adapt to the other player. Commonly, agents only consider their own moves in these two-player games. In this paper, we present a Rolling Horizon Co-evolutionary Planning, a modification to Rolling Horizon Evolutionary Algorithms which considers the actions the opponent may take. We present the results of experiments which compare its effectiveness against other agents playing a subset of GVGAI games and show that co-evolution can improve results by planning for the opponent and outperforms a RHEA agent.

1 Introduction

Two-player General Video Game Playing (GVGP) is an interesting challenge for AI, not only must agents be able to form effective plans for a wide range of games, but also be capable of developing plans expediently (given the real-time nature of these games) and consider the opposing agent in the environment. The unknown actions of this second agent add stochasticity to the environment and may impact the plans made by the player. This means two-player games can pose a more difficult challenge to state-of-the-art GVGP agents compared to single-player games.

The flagship competition/framework for developing AI for general video game playing is the General Video Game AI (GVGAI) competition [19]. GVGAI has a large library of more than 160 games that can be used to test agents in a wide range of single-player and multi-player games. Many games in the framework are re-implementations of classic games, such as *Frogger*, *Pac-Man* and *Sokoban*, although there are also many new and unique games. GVGAI holds yearly competitions for GVGP across a range of tracks, including the two-player games track from which the games and framework used in this study are taken.

The current state of the art in AI agents for two-player GVGAI often uses a random model for simulating which moves the opponent may take. However, it seems modeling the opponent could be useful when planning actions as the utility of certain action sequences may be highly dependent on the actions of the opponent. The main contribution of this paper is the proposition of Rolling Horizon Co-evolutionary Planning (RHCP) for GVGAI. This is a modification of Rolling Horizon Evolutionary Algorithms (RHEA) where the agent both evolves a plan for itself, which represents the moves it may take in the game, alongside a "best guess" plan for what the opponent may do. This method assumes that the opponent is playing rationally and is seeking to maximise its score because the RHCP opponent model is evolved to maximise its score when playing against the agent's best known action sequence. The experiments described in this paper compare these approach against other dominant methods in two-player GVGAI, and also analyze the prediction accuracy of RHCP over the actions taken by the opponent player.

2 GVGAI

Competitions have been used as a way of testing game playing AI algorithms and their ability to solve specific problems. These range from board games, such as *Go* [22], platformers, for example, *Super Mario Bros* [23], and real-time strategy games, e.g. *StarCraft* [14]. However, these have the major disadvantage that the AI developed will be tailored to one game only and will rarely be applicable to other games.

To tackle this problem, a General Game Playing competition was developed in 2005 [8] where submitted agents are evaluated on unknown games (to both the agents and the competitors). For this competition, the games used are variations of existing turn-based discrete games. Therefore, GVGAI was developed to test the adaptability of a single AI to many real-time games. Similar environments, such as the Arcade Learning Environment [1] and Open AI Gym [2], have been developed to tackle artificial general intelligence in games, although these are typically used, without a forward model, in iterative learning tasks.

2.1 Framework and Competition

The GVGAI [18] framework uses the Video Game Description Language (VGDL) [21] to describe games and levels, and to provide a Forward Model (FM) to the agents. This gives the possibility of rolling the state forward to simulate possible upcoming states upon providing an action. There is a great variety of games both within the arcade and puzzle genres, where all the interactions and objects are defined with an objected-oriented language - providing more generalised games. Sprites can be created and parameterized in levels within a 2D game using text files. Currently, GVGAI has 80 publicly available single player games and 50 two-player games, with further private sets of 20 single-player and 20 two-player games used

[1] Goldsmiths University of London, United Kingdom, email: {c.ringer,c.dobre}@gold.ac.uk
[2] Queen Mary University of London, Game AI Research group, email: {c.pacheco,diego.perez}@qmul.ac.uk

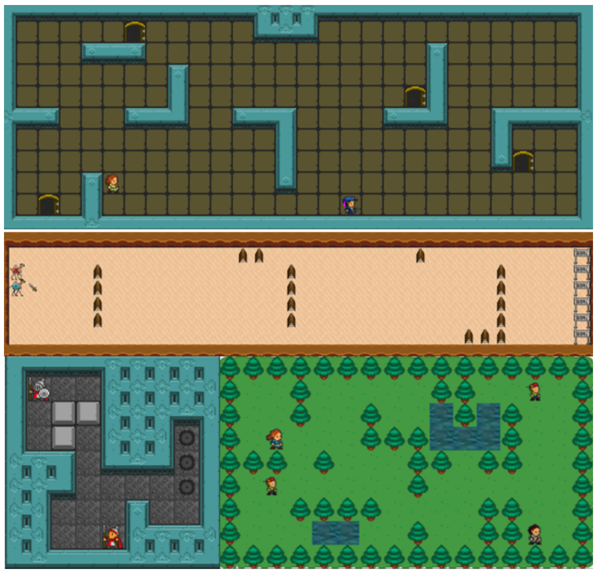


Figure 1. An example of two dimensional arcade style games from GVGAI framework: Samaritan (top), Steeplechase (middle), Sokoban (bottom left) and Gotcha (bottom right).

for evaluating competitions (examples of these games can be seen in Figure 1). In these games, agents receive data about their status in game state. This includes the score, the time step, current winner/loser (if any), sprite positions, orientations, resources and collisions with other sprites. Given this information, it is up to the agent(s) to decipher the games' mechanics and its rules. A complete description of the implementation of the framework and rules of the competition can be found at [19].

At present there are several tracks for this competition: single-player learning [24], single-player planning [19], level generation [11], rule generation [10] and the two-player planning [4] tracks.

2.2 Two-Player Track

The two-player track extends GVGAI to multiplayer games [4]. Interactions and other game state data must be given to both players as well as information on the number of players in the game. Agents can both win, both lose, or only one of them wins. For the state observation, it is possible to access all agents' current score, action, position, speed and orientation.

As previously indicated, the avatars have access to a Forward Model (FM) that allows them to simulate potential future states during their thinking time. This FM permits the agents to make copies of a game state and to roll the state forward by providing an action for every player on the game. Moves in two-player GVGAI are simultaneous in all games. The agents have no previous information about the game rules or mechanics, hence these can only be estimated given the interactions with the environment and other players. The games still range within different genres and games for this track feature both cooperative and competitive scenarios (which is not revealed to the playing agents). Since these new mechanics are introduced, there are many differences in the games: more interactions are available, scores' systems and end conditions - such as defence of objects and both players having to finish in a given location.

A recent survey in GVGAI research can be found in [17], which highlights the different approaches that have been tried in this domain. The two most popular are variants of Monte-Carlo Tree Search (MCTS) [3] and Rolling Horizon Evolutionary Algorithms (RHEA) [15].

2.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) based techniques are widely used in the GVGAI framework for both single and two-players games. MCTS combines Tree Search with Monte Carlo simulations by building the search tree incrementally and evaluating leaf nodes by using a Monte Carlo simulation. A multi-armed bandit algorithm is used to balance exploring tree nodes that have little information known about them with exploiting nodes that are known to be good. At each iteration, MCTS performs 4 steps: selection of a node using Upper Confidence Bounds [12] (UCB), expansion of its children, simulation using a random sequence of moves (rollout) and back-propagation of the reward back through to tree to update parent nodes. This process is repeated until a predefined computational budget is reached. An insight on this algorithm is outside of the remit of this work; for a comprehensive overview see [3].

Of particular interest to this study, [9] investigated the effects of using an opponent model for MCTS in two-player GVGAI [9]. Some of the 9 models implemented are simplifications or variations of the sample MCTS algorithm, while others follow a probabilistic approach. The latter involves off-line learning to have a probability distribution over the possible player's actions on a game state. The results show that the probabilistic models perform the highest win rates. It is interesting to observe that this work is, to the knowledge of the authors, the only one in which an attempt to modelling the opponent is made - all the other approaches for two-player GVGAI assume a random move for the opponent in their use of the FM (or a similarly simple approach).

2.4 Rolling Horizon Evolutionary Algorithms

Rolling Horizon Evolutionary Algorithms (RHEA) are statistical planning algorithms which aim to evolve a sequence of actions (a plan), which maximises a reward function. RHEA was first proposed for real-time games in [15], where the authors applied it to the Physical Travelling Salesman Problem, showing better performance than MCTS in this scenario. Their applications in GVGP has recently been subject to attention, by first studying the vanilla algorithm and its parameters [4], seeding techniques [5] and overall improvements regarding shift buffer, statistical trees and added rollouts [7]. Initially, a population of uniformly random action sequences is generated. Each action is represented by an integer $[0, N-1]$, where N is the number of actions available to the agent. Once this initial population is generated, individuals are evaluated by stepping through the forward model and applying each action in the sequence when the AI is required to act. After this, an heuristic function is used to calculate the value of the game state, which then becomes the fitness for this individual. Once the initial population has been evaluated, genetic operators are applied to evolve a new population of action plans which are, in turn, evaluated and evolved until a set computational budget is exhausted. At this point, the first move in the best individual is returned as the action to take.

3 Rolling Horizon Co-evolutionary Planning

Rolling Horizon Co-evolutionary Planning (RHCP) is a modification of RHEA where the agents' plans are co-evolved with a population of opponent plans. It was first proposed in [13] although, in this work, experimentation was carried out on only one game and as such RHCP's viability for GVGAP was untested. Whilst the player's plan represents the moves the player will make, the opponent's plans represent "best guess" plans about what actions they may take. This is intended to give the agent some information about strong actions the opponent might take when evolving its own plan. This population of opponent plans is then used when calculating the fitness of individuals in the player's plan population and likewise the player plan population is used when assessing the fitness of the opponent's plan. Compared to [9], this approach does not require off-line learning of move probability distributions because plans are evolved online, alongside the agent's plan. However, it does assume that the opponent is playing rationally and its moves are similar to those that are discoverable by a RHEA agent.

In the work presented in this paper, two different evolutionary strategies are used depending on whether the agent's plan or the opponent's plan is being evolved. This is because part of the computational budget is dedicated to investigate which moves the opponent may take, but there needs to be a balance with the time spent on planning the own agent's move. It is possible that, for some games, the opponent's moves are not relevant, for example, a racing game where blocking is not possible and, ideally, little computational time should be employed in this case.

3.1 Agent move planning: μ + λ Evolutionary Strategy

The population of a player's plan is evolved using a μ + λ evolutionary strategy. First, the population is sorted based on fitness, and elitism is applied, selecting the μ most fit individuals. Then λ individuals are generated by first selecting some individuals from the previous population, using tournament selection of size t_size, and then applying uniform crossover and random mutation to these individuals. Uniform crossover is carried out by, for each gene in the new individual, selecting with equal probability either the gene from one of the parents. Random mutation is carried out by selecting a gene uniformly at random and then setting it to a new valid value, which in this case is an integer [0, $N-1$]. The fitness for this new population is then calculated and this process is repeated until a computational budget is exhausted at which point, the individual with the highest fitness is then returned as the best solution.

3.2 Opponent move planning: (1+1) Random Mutation Hill Climber

A (1+1) Random Mutation Hill Climber (RMHC) is an evolutionary algorithm whereby a population of $N = 1$ is evolved by first assessing the fitness of the single individual; followed by mutating its chromosomes using genetic operators and assessing the mutated fitness. If the mutated individual has the same or better fitness, it replaces the original individual as the single member of the population. This process is repeated until a computational budget is exhausted, at which point, the first move in the individual is returned. For this algorithm, the mutation operator used is the same as in the μ + λ ES described above, although it is applied M times to maximize exploration.

3.3 State Evaluation

Both the $\mu + \lambda$ ES and $(1 + 1)$ RMHC use the same scoring heuristic when determining the fitnesses of an individual. The fitness is defined

Algorithm 1 Rolling Horizon Co-evolutionary Planning

```
1:  Requires state: current game state.
2:  procedure ACT($state$)
3:      init()
4:      while $budgetremains$ do
5:          evaluate(population, opponent, state)
6:          evolve(population, opponent, state)
7:      return $population.best().first\_move()$
8:  procedure INIT
9:      $population \leftarrow randompopulation$
10:     $opponent \leftarrow randomindividual$
11: procedure EVALUATE($population$, $opp$, $state$)
12:     for indv in population do
13:         $indv.fit \leftarrow simulate\_game(indv, opp, state)$
14:     $population \leftarrow sort(population)$
15: procedure SIMULATE_GAME($indv$, $opponent$, $state$)
16:     for i in indv.length do
17:         $move_a \leftarrow indv[i]$
18:         $move_b \leftarrow opponent[i]$
19:         $state.advance(move_a, move_b)$
20:     return $score\_heuristic(indv, state)$
```

as the game score for the player in the state reached at the end of the sequence, plus a large negative value if the player lost (-1000) or a large positive if the player won (1000). This scoring heuristic is also used in our experiment as the scoring heuristic for the other agents which require a scoring function (RHEA and MCTS).

3.4 Co-Evolution

Co-Evolution refers to evolving chromosomes for two (or more) different populations in parallel by using one to influence the fitness of the other[20]. This study presents a methodology for co-evolution of the agent's plan with a "best guess" plan for the opponent. When evaluating the fitness of the agent's plan, we use the best guess opponent plan to simulate what actions the opponent may take. Likewise, when evaluating the opponent's plan, we simulate the actions the agent takes using its best known plan.

In this paper, we use a μ + λ evolutionary strategy to evolve the agent's plan and a (1+1) RMHC to evolve our "best-guess" opponent's plan. In each iteration of the algorithm, the population of plans is evaluated, the elite selected and the rest mutated. Then the opponent plan is mutated and both the original and mutated plans are evaluated against the best known player plan. The opponent plan with the highest fitness is then preserved and survives into the next iterations of the algorithm. This process is repeated until the budget is exhausted. Algorithm 1 details the core RHCP loop as well as initialisation and evaluation policies; algorithm 2 details the genetic operators for RHCP.

4 Experiment

For this experiment, 10 games from the GVGAI Two-Player track were used, those found in the GVGAI Two-Player Training Set 1. This training set contains a mixture of Competitive (8) - games where players are competing and there is at most one winner - and Cooperative (2) games - where players are working towards a shared goal and win or lose together - as well as a mixture of Symmetric (7) games - where each player has the same goal - and Asymmetric (3) games

Table 1. Games Set comprising of all games from the GVGAI Two-Player Training Set 1. The key attributes that each game can have are Cooperative (Co) or Competitive (Cp), Symmetric(Sym) or Asymmetric(Asym), and Stochastic (S) or Deterministic (D). The descriptions given here are abridged version of the descriptions found at http://www.gvgai.net.

ID	Game	Key Attributes	Description
0	Akka Arrh	Co, Sym, S	Two players defend a locked spaceship from aliens, while trying to open and enter in it. Both players win when they enter the spaceship but lose if one is hit by the aliens.
1	Asteroids	Cp, Sym, S	Space game where players shoot asteroids for points, which are then broken into smaller asteroids. The players can also shoot each other. Last player standing wins, or the one with more points at the end.
2	Capture the Flag	Cp, Sym, D	The level is divided into two areas, each with a flag. Players must capture the opponent's flag and bring it to their area. Players can capture each other. Capturing gives points, the player with most points wins at the end.
3	Cops N Robbers	Cp, Asym, D	One player is the robber and the other is a cop. The robber wins if all gems are collected and the cop wins if he catches the robber.
4	Gotcha	Cp, Asym, S	One player has to chase the other. There are safe places where the chased one can hide. If the chaser catches the other player, they win but loses if time is over before that.
5	Klax	Cp, Sym, S	Coloured objects fall from the sky. Players catch them for points (own colour is worth more points). Higher score at the end decides the winner.
6	Samaritan	Cp, Asym, D	One player tries to cross a portal to another world, while the other tries to avoid so. The first player wins the game by reaching it on time.
7	Sokoban	Co, Sym, D	Both players must push all boxes into determined locations. The game ends when all boxes are correctly placed.
8	Steeplechase	Cp, Sym, D	Racing game. Players win by reaching the end. There are many obstacles and a hidden gem is worth many score points.
9	Tron	Cp, Sym, D	A version of the classic game with the same name. Players race in a wall-encircled arena creating walls as they move. The first player that collides with a wall loses.

Algorithm 2 Rolling Horizon Coevolutionary Planning Genetic Operators

```
1:  Requires elitism ($\mu$)
2:  procedure EVOLVE($population, opponent, state$)
3:      $newPop \leftarrow emptypopulation$
4:      while $newPop.size < elitism$ do
5:          $newPop \leftarrow add(next\_fittest\_indiviudal)$
6:      while $newPop.size < population.size$ do
7:          $newPop \leftarrow selectandmutate(population)$
8:      $mutOp \leftarrow mutate(oppoenent)$
9:      $opFit \leftarrow simulate\_game(opponent, bestindv, state)$
10:     $opMFit \leftarrow simulate\_game(mutOp, bestindv, state)$
11:     if $opMFit <= opFit$ then
12:         $opponent \leftarrow mutationOpponent$
13: Requires $t\_size$: tournament size
14: procedure SELECTANDMUTATE($population$)
15:     while $tournament.size <$t_size do
16:         $tournament \leftarrow population.rand\_indv()$
17:     $parentA \leftarrow tournament.fittest()$
18:     $parentB \leftarrow tournament.secondFittest()$
19:     $new\_indv \leftarrow uniformcrossOver(parentA, parentB)$
20:     return $mutate(new\_indv, mutation)$
21: Requires $M$: mutation repetitions
22: procedure MUTATE($individual$)
23:     for $M$ do
24:         $gene_idx \leftarrow individual.new\_random\_gene()$
25:         $individual[gene_idx] \leftarrow new\_random\_gene()$
26:     return $individual$
```

- where each player has a different goal. There is also a mixture of Stochastic games, those with a random element, and Deterministic games, those that have no randomness from the environment. A list of games and their properties can be found in Table 1.

Four algorithms were tested; our RHCP agent and 3 control agents. All 3 control agents, RHEA, MCTS, and Random, were modifications of sample agents provided with the GVGAI framework. RHEA and RHCP use both the same length for their individuals ($l = 15$), tournament selection ($t\_size = 2$), uniform crossover and mutation. The only difference between the two algorithms is that RHEA used a population of $n = 10$ whereas RHCP used a population of $n = 8$. This was in order to allow computational budget for the secondary population for the opponent's plan, such that it also evaluates 10 individuals per generation.

In RHCP, the player's plan population has a μ of 1, selected through elitism, and $\lambda = n - \mu$. The population of λ individuals is generated by selecting past individuals through tournament selection, crossover and random mutation, where 1 gene is mutated, selected uniformly at random. When mutating the opponents plan in RHCP, M is set to 5.

The default MCTS, included in the GVGAI framework, is used for the experiments. This is a closed loop version [16] of the algorithm, using a standard UCB1 [12] function for its tree policy. Uniform random roll-outs, limited to a depth of 15, are used for the default policy. Finally, the random agent is also taken verbatim from the provided agents and it simply chooses one of the available moves in the current state uniformly at random.

RHEA, MCTS and RHCP substituted the time-based budget from the competition (40ms of real-time per decision) for a number of usages of the forward model's function that rolls the state forward. This ensures that all experiments are consistent independently from the machine used. As in [6], 900 FM calls were provided as the decision-making budget for each game tick.

For each game, all 5 levels were evaluated with 2 different orders, each agent playing either side, providing 10 different scenarios. Each scenario is then played 10 times, resulting in 100 trials per agent pairing on each game, and a total of 1000 matches per pair of controllers.

5 Results

This experiment compares our proposed RHCP algorithm against a suite of control algorithm as well as investigating if co-evolution help predict the opponent's next move. The overall win/loss results for all ten games are shown in Table 2 and the prediction accuracy against each opposing algorithm is in Table 3. These are discussed below.

5.1 Comparison of Agent vs Agent performance

5.1.1 RHCP vs RHEA

Comparing RHCP to RHEA is perhaps most interesting as RHCP operates similarly to RHEA, with the addition of co-evolving an opponent's plan at the expense of some computational time for evolving its own plan. Of the 1000 games played, RHCP won 360 compared to 309 games won by RHEA. These 51 games, or 5.1% increase is a significant ($p < 0.05$) improvement. A 2-sample Z test was carried out for significance testing with a Z-Score of 2.417 and a p-value of 0.01552.

Furthermore, RHCP performed at least as well as, if not better, in every game, indicating that spending time planning for the opponent's is worthwhile, even in environments such as GVGAI where there is a very limited computational budget. It is interesting to note that the two games where RHCP really performed well, Game 2 - Capture the Flag and Game 9 - Tron, are particularly adversarial; the actions the opponent takes have a huge impact on the performance of the agent.

5.1.2 RHCP/RHEA vs MCTS

MCTS is the most dominant two-player GVGAI approach and it is clear that whilst RHCP performs better than RHEA, the performance of both is considerably worse than MCTS. Neither RHCP nor RHEA is significantly better than one another when playing against MCTS. It does seem that RHCP is more competitive than RHEA against MCTS (winning more and losing less), although further testing would be required to confirm this, especially due to the stochastic nature of evolutionary algorithms. However, it is interesting to note that RHCP and RHEA performed similarly, within 4 wins of each other, for all games except Game 5 - Klax. Whilst this win ratio improvement is not significant, this is another example of RHCP performing better in games which are highly interactive and are reliant on your opponent's moves, exactly the sorts of games RHCP can give a competitive advantage.

5.1.3 RHCP/RHEA/MCTS vs Random

Understandably RHCP, RHEA, and MCTS all performed very well against a random agent. It is interesting to note that even against a random agent, no other agent was able to achieve a better than 50/50 win ratio in Game 4 - Gotcha. We believe this because not only is Gotcha asymmetric, but also unbalanced such that when playing one 'role' it is very challenging.

Against a random agent our implementations of RHEA and MCTS actually use an opponent model with exactly the same logic as the real opponent. It can be observed that RHCP performs similarly to the other agents, with no significant difference between RHCP and RHEA when playing against Random.

5.2 Prediction Accuracy

The opponent's predicted next move based on the first move of the RHCP evolved opponent's plan and the opponent's actual move were recorded for each tick in games involving RHCP. Table 3 shows the average accuracy of these predictions for each game played against each opponent.

The prediction accuracy was not significantly different amongst the various opponents. Given that one agent is a Random agent, this essentially means that across all games predicting the next move based on co-evolution is no better than random. That said Game 4 - Gotcha is a game where RHCP was able to predict the next move with a high average accuracy against the RHEA agent. The prediction accuracy was 0.418, compared to 0.199 against random, a significant improvement (Z Score = 3.36, $p < 0.05$).

Even though the average move prediction accuracy against RHEA/MCTS is not very high, co-evolution is clearly making a positive difference to decision making, evidenced by RHCP agent performing better than RHEA. One possible reason for the poor prediction accuracy reported is the stochastic nature of the evolutionary algorithm coupled with the small number of evolution iterations, approximately 6, the algorithm performs each game tick (as the FM budget each game tick is 900, the population is n=10, the simulation length is l=15, therefore $10 * 15 * 6 = 900$). It is possible that the prediction plan may find a strong move sequence, but not the exact move sequence found by the opponent.

6 Conclusion

In this paper, we present Rolling Horizon Co-evolutionary Planning (RHCP) for General Video Game Playing, which evolves plans of actions for itself and the opponent. We have experimented and compared this algorithm to three others, RHEA, MCTS and Random in a set of 10 two-player GVGAI games. We have found that spending computation time considering the potential opponent's moves shows improvement compared to the same algorithm without this feature (RHEA), but still performed worse than MCTS. Surprisingly, despite RHCP outperforming RHEA, it was not able to predict the next move of 'rational' agents (RHEA/MCTS) with a greater accuracy than against Random. Perhaps the opponent plan evolution is finding strong sequences over 15 moves but due to the stochastic nature of RHCP it often disagrees on one move in particular.

Further investigation into this is required to fully understand the positive effect of co-evolution, but it's reasonable to think that predicting the opponent's move more accurately would improve the performance of RHCP. Future work should additionally investigate how existing RHEA modifications impact RHCP, including probabilistic approaches to further improve the prediction of the opponent's actions. Also, submitting RHCP to the GVGAI competition would allow assessing its performance against other approaches.

REFERENCES

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling, 'The arcade learning environment: An evaluation platform for general agents.', *J. Artif. Intell. Res.(JAIR)*, **47**, 253–279, (2013).

Table 2. Results from all games played across the 6 pairings of agents. For each pair, first (second) column shows number of victories for the first (second, respectively) agent. The maximum possible wins for each pairing and game is 100 (ties possible).

GameID	RCHP vs RHEA		RCHP vs MCTS		RHEA vs MCTS		RCHP vs Rand.		RHEA vs Rand.		MCTS vs Rand.	
0	0	0	1	1	3	3	0	0	0	0	1	1
1	49	45	9	85	7	87	85	9	92	6	100	0
2	56	45	6	97	3	98	88	12	90	11	100	0
3	39	31	3	66	4	70	59	8	57	13	69	1
4	50	50	50	50	50	50	50	50	50	50	50	50
5	54	48	35	70	23	79	100	0	97	4	100	0
6	50	50	46	54	50	50	56	38	53	44	64	34
7	0	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	3	4	3	2	1	3	4	12	2
9	61	39	6	94	10	90	98	2	98	2	100	0
Total	**360**	**309**	**157**	**520**	**154**	**530**	**538**	**120**	**540**	**134**	**596**	**88**

[!t]

Table 3. Prediction accuracy (Acc $\in [0, 1]$) for the first move in the RHCP opponent plan compared to the actual move taken, together with the Standard Deviation (SD) across all trials per game against all opposing agents.

	RHEA		MCTS		Random	
Game	Acc	SD	Acc	SD	Acc	SD
0	0.185	0.016	0.165	0.016	0.166	0.02
1	0.169	0.107	0.160	0.167	0.173	0.15
2	0.223	0.01	0.201	0.010	0.2	0.01
3	0.2	0.032	0.176	0.031	0.18	0.033
4	0.418	0.013	0.202	0.041	0.199	0.025
5	0.375	0.013	0.332	0.013	0.418	0.013
6	0.201	0.034	0.180	0.044	0.181	0.026
7	0.223	0.009	0.199	0.009	0.199	0.009
8	0.186	0.008	0.166	0.009	0.168	0.009
9	0.245	0.06	0.227	0.045	0.197	0.102
Avg	**0.223**	**0.032**	**0.201**	**0.038**	**0.208**	**0.04**

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, 'Openai gym', *arXiv preprint arXiv:1606.01540*, (2016).

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton, 'A survey of monte carlo tree search methods', *IEEE Transactions on Computational Intelligence and AI in games*, **4**(1), 1–43, (2012).

[4] Raluca D Gaina, Adrien Couëtoux, Dennis JNJ Soemers, Mark HM Winands, Tom Vodopivec, Florian Kirchgeßner, Jialin Liu, Simon M Lucas, and Diego Perez-Liebana, 'The 2016 two-player gvgai competition', *IEEE Transactions on Computational Intelligence and AI in Games*, (2017).

[5] Raluca D. Gaina, Simon M. Lucas, and Diego Perez-Liebana, 'Population seeding techniques for Rolling Horizon Evolution in General Video Game Playing', *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, 1956–1963, (2017).

[6] Raluca D Gaina, Simon M Lucas, and Diego Pérez-Liébana, 'Population seeding techniques for rolling horizon evolution in general video game playing', in *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pp. 1956–1963. IEEE, (2017).

[7] Raluca D. Gaina, Simon M. Lucas, and Diego Perez-Liebana, 'Rolling horizon evolution enhancements in general video game playing', *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017*, 88–95, (2017).

[8] Michael Genesereth, Nathaniel Love, and Barney Pell, 'General game playing: Overview of the aaai competition', *AI magazine*, **26**(2), 62, (2005).

[9] Jose Manuel Gonzalez-Castro and Diego Perez-Liebana, 'Opponent models comparison for 2 players in gvgai competitions', in *Computer Science and Electronic Engineering Conference (CEEC)*, (2017).

[10] Ahmed Khalifa, Michael C Green, Diego Pérez-Liébana, and Julian Togelius, 'General Video Game Rule Generation', in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, (2017).

[11] Ahmed Khalifa, Diego Perez-Liebana, Simon M Lucas, and Julian Togelius, 'General video game level generation', in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, (2016).

[12] Levente Kocsis and Csaba Szepesvári, 'Bandit based monte-carlo planning', in *European conference on machine learning*, pp. 282–293. Springer, (2006).

[13] Jialin Liu, Diego Pérez-Liébana, and Simon M. Lucas, 'Rolling horizon coevolutionary planning for two-player video games', *2016 8th Computer Science and Electronic Engineering Conference, CEEC 2016 - Conference Proceedings*, 174–179, (2017).

[14] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss, 'A survey of real-time strategy game ai research and competition in starcraft', *IEEE Transactions on Computational Intelligence and AI in games*, **5**(4), 293–311, (2013).

[15] Diego Perez, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen, 'Rolling horizon evolution versus tree search for navigation in single-player real-time games', *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*, 351, (2013).

[16] Diego Perez Liebana, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas, 'Open loop search for general video game playing', in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 337–344. ACM, (2015).

[17] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas, 'General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms', (2018).

[18] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Simon M Lucas, and Tom Schaul, 'General video game ai: Competition, challenges and opportunities', in *Thirtieth AAAI Conference on Artificial Intelligence*, pp. 4335–4337, (2016).

[19] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson, 'The 2014 general video game playing competition', *IEEE Transactions on Computational Intelligence and AI in Games*, **8**(3), 229–243, (2016).

[20] Christopher D. Rosin and Richard K. Belew, 'New methods for competitive coevolution', *Evol. Comput.*, **5**(1), 1–29, (March 1997).

[21] Tom Schaul, 'A video game description language for model-based or interactive learning', in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pp. 1–8. IEEE, (2013).

[22] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., 'Mastering the game of go with deep neural networks and tree search', *nature*, **529**(7587), 484–489, (2016).

[23] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis, 'The mario ai championship 2009-2012', *AI Magazine*, **34**(3), 89–92, (2013).

[24] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana, 'Deep reinforcement learning for general video game ai', in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, (2018).

WordClicker: Tuning our approach for Language Resourcing

Chris Madge[1] and **Richard Bartle**[2] and **Jon Chamberlain**[3] and **Udo Kruschwitz**[4] and **Massimo Poesio**[5]

Abstract. Previous work has shown great promise for overcoming the challenges of embedding text annotation tasks in games through the application of incremental game mechanics. This demo describes some of the design choices that are being explored to tune the incremental game approach to perform well as a true GWAP (Game With A Purpose) for language resourcing.

1 Introduction

There is a growing number of Games-With-A-Purpose (GWAPs) for creating language resources, that have shown promise in terms of their ability to gather high quality annotations. However, player recruitment and retention remain a challenge with such games, that have yet to acquire or retain players at a scale comparable to the most successful GWAPs [8, 1]. To address this, previous work proposed the application of incremental game design mechanics.

The original GWAPs, such as The ESP Game, were effective in presenting their tasks, as per the original definition, in such a way that the labels gathered were a byproduct of play [8]. In contrast, it has been said that language resourcing games such as PhraseDetectives , are not really GWAPs as annotations are not a byproduct, but rather it is evident that the player is annotating text [5]. This can be said of the majority, if not all language resourcing GWAPs. Wordrobe for example, unlike PhraseDetectives, is a game which deliberately aims to hide the true nature and linguistic complexity of the tasks by presenting them as multiple choice questions and removing linguistic terminology [7]. However, it remains evident the player is annotating text. Similarly for other well-known gamelike approaches to NLP resource creation such as Jeux-de-Mots and Zombilingo [4, 2]. Proper GWAPs have been proposed, but never really used for resource creation [3] nor one that can be easily hidden. Ultimately this creates an inescapable problem of a core game mechanic that the majority of people find tedious and repetitive.

To address this, we chose to employ selective design concepts from incremental games, a game genre of entertaining games created out intrinsically repetitive activities. We developed a game that incorporated text annotation as the core mechanic. This has yielded very positive results for player recruitment and retention [6].

In this work we demonstrate some of the design concepts being explored to fine tune WordClicker [6] as a GWAP.

[1] Queen Mary Univ. Of London, email: c.j.madge@qmul.ac.uk
[2] University Of Essex, email: rabartle@essex.ac.uk
[3] University Of Essex, email: jchamb@essex.ac.uk
[4] University Of Essex, email: udo@essex.ac.uk
[5] Queen Mary Univ. Of London, email:m.poesio@qmul.ac.uk

2 WordClicker

WordClicker is a web-based, desktop and mobile friendly, one-player game in which a player learns the classes of words by playing a baker that gets her/his ingredients by clicking on words associated with those ingredients. The core game mechanics is simply classifying individual words into classes (associated with ingredient jars) by clicking on them, a mechanics that should be transferable to the majority of word-labelling tasks. If the player is correct, after clicking he/she gets ingredients that are used to make the cakes. The game is very simple, taking approximately two weeks for one person to develop. To begin with, the player is shown details of the task they will be performing with a short explanation. When they press play they are presented with an interactive tutorial that takes them through basics of the game. They can repeat this tutorial and view additional instructions regarding the classes at any point. During gameplay, the player is shown a single sentence at a time. They can advance to the next sentence by using the "Next sentence" button. As the action step of the game loop, they mark words by clicking on the appropriate ingredients jar, then selecting one or more words in the sentence that are of that category. When the player selects the correct item, the tag is shown with a shimmering effect and an animation shows the ingredients going into the appropriate jar. In the wait step of the loop, that jar empties over time as cakes are automatically created and sold, giving the player a reward. To encourage the player to explore all the categories, the cakes that are produced are worth more when the player has more ingredients available (depicted by the changing cake). When the player labels incorrectly, they are given feedback in the form of a text notification message that appears in the bottom left hand corner and a flashing red outline on the token. Incorrect labels damage the player's purchases. Purchases are acquired from the shop by the player investing their reward. They increase the number of cakes produced per second and the number of ingredients produced by selecting a correct token. Progress bars in the shop area show how far away they are from being able to purchase a new item, with item abilities being hidden until they have almost accumulated sufficient funds. The cost of items increments exponentially after each purchase, giving the game an infinite feel.

3 Tuning our Approach

In tuning our design for language resourcing, we want to raise accuracy as high as possible, without detracting too greatly from player retention or engagement.

3.1 Feedback

Before investigating mechanics that positively reinforce accurate annotation, we looked for methods of giving the player more feedback in relation to their accuracy that could be themed and incorporated into the game setting. We achieved this in the form of a "Company Report" that informs the player what their "Reputation" is (shown in Figure 1). They are told that the more accurately they label, the higher their reputation would be. Technically speaking, their reputation is linked to their F-measure based on each round they label. The company report is available after three rounds, and is opened from the navigation bar. Further achievements may be linked to "Reputation" performance in future to draw further attention to the player's accuracy.

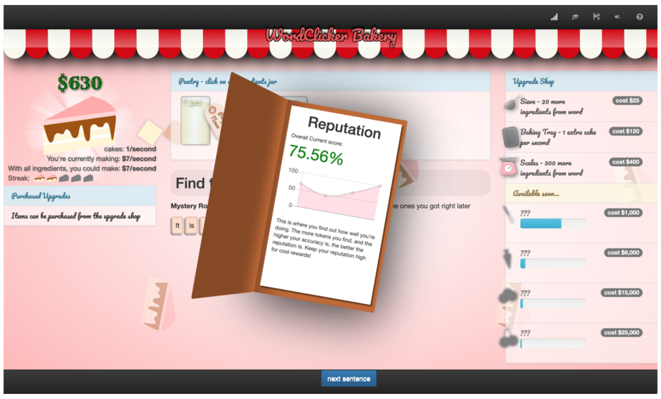

WordClicker Bakery
$630
Reputation
Overall Current score:
75.56%
This is where you find out how well you're doing. The more tokens you find, and the higher your accuracy is, the better the reputation is. Keep your reputation high for cool rewards!
Upgrade Shop
next sentence

Figure 1. Company Report - Provides themed ongoing in-game feedback to player about their accuracy

3.2 Precision

Next we looked at methods of increasing precision. To achieve this we reward a winning streak. We take a winning streak to be five consecutive correct judgements, without error. The player is rewarded with a "Sugar Rush" (shown in Figure 2). This gives the player double the points for thirty seconds or until they make a mistake. The current winning streak status is depicted by five cakes that are grey silhouettes prior to a win. In Sugar Rush mode a spinning radial burst attracts the players eye to the status panel of the game where they are told they are in "Sugar Rush" and will receive double points. This is accompanied by the further visual effect of the cakes in the game dancing back and forth. We hope that this will encourage players to label more carefully, in turn raising precision.

3.3 Recall

It is possible that any measure that rewards players for labelling with high precision may cause them to narrow in on a safe and familiar strategy, impacting learning of new labels and reducing recall. We next devised a method of attenuating that potential affect and increasing recall. We now require a player to find ingredients for all the jars currently available to them in the game in order to produce any cakes. While the virtual currency of ingredients is increasing as the player finds correct tokens, the virtual currency of in-game dollars remains stationary unless there is at least some quantity of every ingredient. A message beneath the jars informs the player which ingredients they

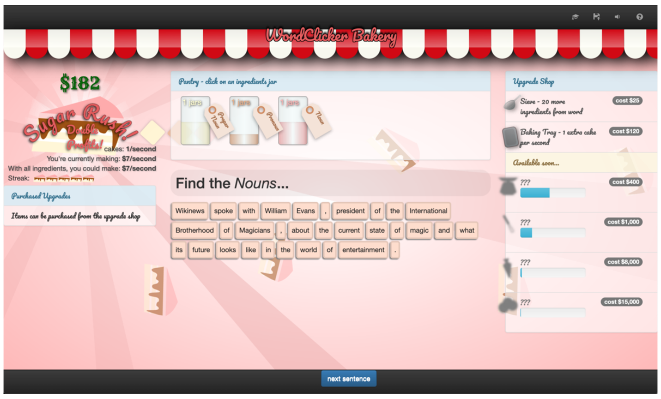

WordClicker Bakery
$182
Sugar Rush
Find the Nouns...
Wikinews spoke with William Evans , president of the International Brotherhood of Magicians , about the current state of magic and what its future looks like in the world of entertainment .
Upgrade Shop
next sentence

Figure 2. Complete Winning Streak resulting in "Sugar Rush"

are lacking (shown in Figure 3). We hope this encourage players to continue to explore all of the labels throughout their gameplay, rather than repeatedly return to labels they are more comfortable with.

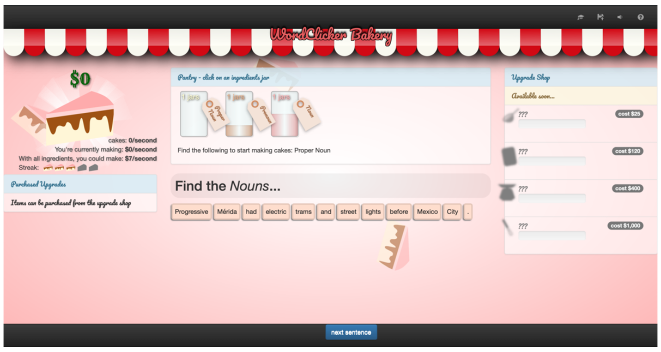

WordClicker Bakery
$0
Find the following to start making cakes: Proper Noun
Find the Nouns...
Progressive Mérida had electric trams and street lights before Mexico City .
Upgrade Shop
next sentence

Figure 3. Notification of required ingredients

4 Conclusion

This work is a demo of the continued design in WordClicker. With our latest designs we hope to increase precision, recall and player awareness of their current accuracy through the use of new in-game elements and visualizations without impacting the current entertainment factors. Further work will test these new elements and examine their effects.

ACKNOWLEDGEMENTS

This research was supported by the DALI project, ERC Grant 695662.

REFERENCES

[1] Seth Cooper, Firsas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popovic, and the Foldit Players, 'Predicting protein structures with a multiplayer online game', *Nature*, **466**, 756–760, (2010).

[2] Karen Fort, Bruno Guillaume, and Hadrien Chastant, ‘Creating *Zombilingo* , a game with a purpose for dependency syntax annotation’, in *Proceedings of the First International Workshop on Gamification for Information Retrieval - GamifIR ’14*, pp. 2–6, Amsterdam, The Netherlands, (2014). ACM Press.

[3] David Jurgens and Roberto Navigli, ‘It’s all fun and games until someone annotates: Video games with a purpose for linguistic annotation’, *Transactions of the Association of Computational Linguistics*, **2**(1), 449–464, (2014).

[4] Mathieu Lafourcade, ‘Making people play for Lexical Acquisition with the JeuxDeMots prototype’, 8, (2007).

[5] Mathieu Lafourcade, Alain Joubert, and Nathalie Le Brun, *Games with a Purpose (GWAPS)*, John Wiley & Sons, 2015.

[6] Christopher Madge, Richard Bartle, Jon Chamberlain, Udo Kruschwitz, and Massimo Poesio, ‘Incremental Game Mechanics Applied to Text Annotation’, in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY ’19, pp. 545–558, New York, NY, USA, (2019). ACM. event-place: Barcelona, Spain.

[7] Noortje Venhuizen, Kilian Evang, Valerio Basile, and Johan Bos, ‘Gamification for word sense labeling’, in *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)*, (2013).

[8] Luis Von Ahn, ‘Games with a purpose’, *Computer*, **39**(6), 92–94, (2006).