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

Martin Balla,[1] Adrián Barahona-Ríos,[2] Adam Katona,[2] Nuria Peña Pérez,[1] Ryan Spick[2]
[1]School of Electronic Engineering and Computer Science, Queen Mary University of London
[2]Department of Computer Science, University of York
[1]{m.balla,n.penaperez}@qmul.ac.uk
[2]{ajbr501,ak1774,rjs623}@york.ac.uk

Abstract

In this paper we demonstrated 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 to illuminate the game space instead of just finding 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 kind 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 game parameters selected. The agent's performance was plotted against its behaviour for further exploration, which allowed visualizing how the agent performed with selected behaviour traits.

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 good games can also be interesting, especially from a design point of view.

Some 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 [1]. 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 [2]. 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 [3]. 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 [4]. 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 [5], 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. Mixed-initiative systems combine PCG with user input, incorporating creative direction to the process [6].

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 [7] can be used to illuminate the game 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). A Rolling Horizon Evolutionary Algorithm agent (RHEA) was used to automatically test 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.

Background

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 or score 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 manually 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 [8]. Continuous parameter optimization is more complex as more combinations arise. Popular continuous optimization algorithms are Random Search and CMA-ES [9].

MAP-elites

MAP-elites is a divergent search algorithm which belongs to a family of algorithms called Quality Diversity (QD) [10]. The goal of Quality Diversity algorithms is to find a diverse set of good 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 Behavior 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 [11], 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 [12] the BC can be defined by how much each leg touches the ground. While some QD algorithms like Novelty Search with Local Competition [11] focuses more on quality, MAP-elites puts more emphasis on diversity. This is achieved by considering multiple dimensions of behaviour 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.

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 Fig. 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 Fig. 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. 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.

Rolling Horizon Evolutionary Algorithm

A variant of the Rolling Horizon Evolutionary Algorithm (RHEA)[13] with a single individual (based on a rolling horizon mountain climber (RHMC) [14] algorithm) was considered for this work.

This algorithm constructs a random sequence of actions, which is executed in the forward model and a score from the state reached is calculated. In Cave Swing the value of the state is based on the actual numerical score in the game, which is updated and available at every game state.

Then, the current solution (i.e the current action sequence) is mutated and the mutated copy is evaluated. If better, the previous solution is overwritten. The process continues until the solution is good enough or the maximum time is elapsed. 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.

Methodology

Experimental Procedure

The project was implemented as a client-server based architecture. The server in Java contained an implementation of the game Cave Swing and the RHEA agent. The client in Python contained the MAP-elites implementation and was used for visualizing the data received from evaluation in the server.

Firstly, given sets of parameters was selected by the MAP-Elites algorithm and passed to the server to run the evaluations on the game.

Cave Swing is fully deterministic, while the agent is stochastic due to the random initial sequence and random mutations. In order to run the evaluations of the game,

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

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

Table 3: MAP-elites parameters

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

the agent's parameters were fixed, but not the seed, so the agent's performance could differ between runs. To get a better estimate of the level's difficulty, every set of parameters was evaluated 20 times using the same agent with fixed parameters. The performance was defined as the agent's score for each run. The observed behaviour features and the final score were averaged over the 20 runs and returned to the client.

In the client, the MAP-elites created a grid (or map) with dimensions 50x50, where each cell belonged to a certain behaviour. This behaviour map was characterized by two features chosen for their simplicity and interesting emerging behaviours:

1. Game duration (d), which was calculated as the number of ticks per game run.
2. Average height $\bar{h}$ of the agent along one run of the game, defined by equation 2.

$$\bar{h} = \frac{\sum_{t=0}^{d} y_t}{w} \tag{2}$$

Where w is the map's width, and y_t is the agent's height for a specific time t.

In order to initialize the grid, the possible range for both of the behaviour features were obtained. This was achieved by running a thousand evaluations and measuring the minimum and maximum values. Then the grid was created such that each cell had uniform size.

The entire selection and evaluation was repeated many times. While all the cells in the behaviour map were empty, the algorithm randomly picked parameters and evaluated them. The length of this stage was controlled by a parameter "initial random evaluations".

Once this part was completed, the algorithm continued by randomly selecting a populated cell and mutating its parameters. Cave Swing's parameter space contains boolean, integer and floating point parameters so a simple mutation operator was used for simplicity. Whenever a parameter was mutated, it was reassigned with a uniformly distributed random number in the valid range. For each mutation the number of mutated parameters was randomly selected between one and the mutation rate.

For this experiment not all the parameter space of Cave Swing was explored. Those parameters related to the score calculation and the map dimensions were fixed, as they would affect differently the game evaluation and the characterization of both of the behaviour features selected. Please refer to table 1 for information regarding which parameters were explored.

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 behaviour map returned its associated parameters by clicking on them. This allowed to automatically send the selected set of parameters to the Java server and run a simulation of the game, where the behaviour of the agent could be visually inspected.

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 (i.e. 1440 out of 2500 different behaviours were found).

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

By visually inspecting Fig. 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.

Fig. 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 Fig. 3b 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. Fig. 3a 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. Fig. 3c 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 Fig. 4).

Fig. 4a 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 Figs. 3a and 3c. Visual inspection of Fig. 4a 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.

Figs. 4b to 4d use the color intensity to represent the value of the parameter selected instead of the performance.

Fig. 4b 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.

Fig. 4c displays the relationship between the number of anchors and the behaviour map and Fig. 4d 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.

Discussion

In general, the obtained results show that it is possible to explore the behaviour space by using the proposed algorithm. 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 this 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 allows 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, 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 proved by Cully et al [12].

Conclusion and Future Work

This work presents an exploration of the use of MAP-elites to a low-parameter game space in order to discover interesting combinations of game parameters. Agents are then simulated to play the game trying to achieve the highest possible score. 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 pre-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.

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, so optimizing an agent over the set of games could lead to an agent which plays the game overall very well. As Cave Swing is fairly simple, applying MAP-elites to more complex games would be more interesting, where we could evaluate more interesting behaviour features.

Acknowledgments

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

References

[1] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring game space using survival analysis.," in *FDG*, 2015.

[2] J. Tremblay, A. Borodovski, and C. Verbrugge, "I can jump! exploring search algorithms for simulating platformer players," in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[3] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Discovering unique game variants," in *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*, 2015.

[4] J. Togelius, E. Kastbjerg, D. Schedl, and G. 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.

[5] J. Togelius, N. Shaker, and M. J. Nelson, "Procedural content generation in games: A textbook and an overview of current research," *Togelius, N. Shaker, M. NelsonBerlin: Springer*, 2014.

[6] A. Baldwin, S. Dahlskog, J. M. Font, and J. Holmberg, "Mixed-initiative procedural generation of dungeons using game design patterns," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 25–32, IEEE, 2017.

[7] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[8] S. M. Lucas, J. Liu, and D. 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.

[9] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[10] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[11] J. Lehman and K. 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.

[12] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, p. 503, 2015.

[13] D. Perez, S. Samothrakis, S. Lucas, and P. 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, (New York, NY, USA), pp. 351–358, ACM, 2013.

[14] J. Liu, D. Pérez-Liébana, and S. M. Lucas, "Bandit-based random mutation hill-climbing," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2145–2151, IEEE, 2017.

Appendix

Individual Contributions

The idea for the project originated from Adam. We discussed the plan and the scope of the project together. As we used a client-server architecture we divided the tasks, Adam and Adrián worked on the python side, while Ryan, Martin and Nuria worked on the Java side. We made the design decisions together. After implementing the main workflow, Ryan implemented the Graphical Interface and generated the behaviour visualisation, Martin implemented the java function which runs the game with visuals and Nuria started working on the report. We discussed the structure and the details and then we wrote it together.

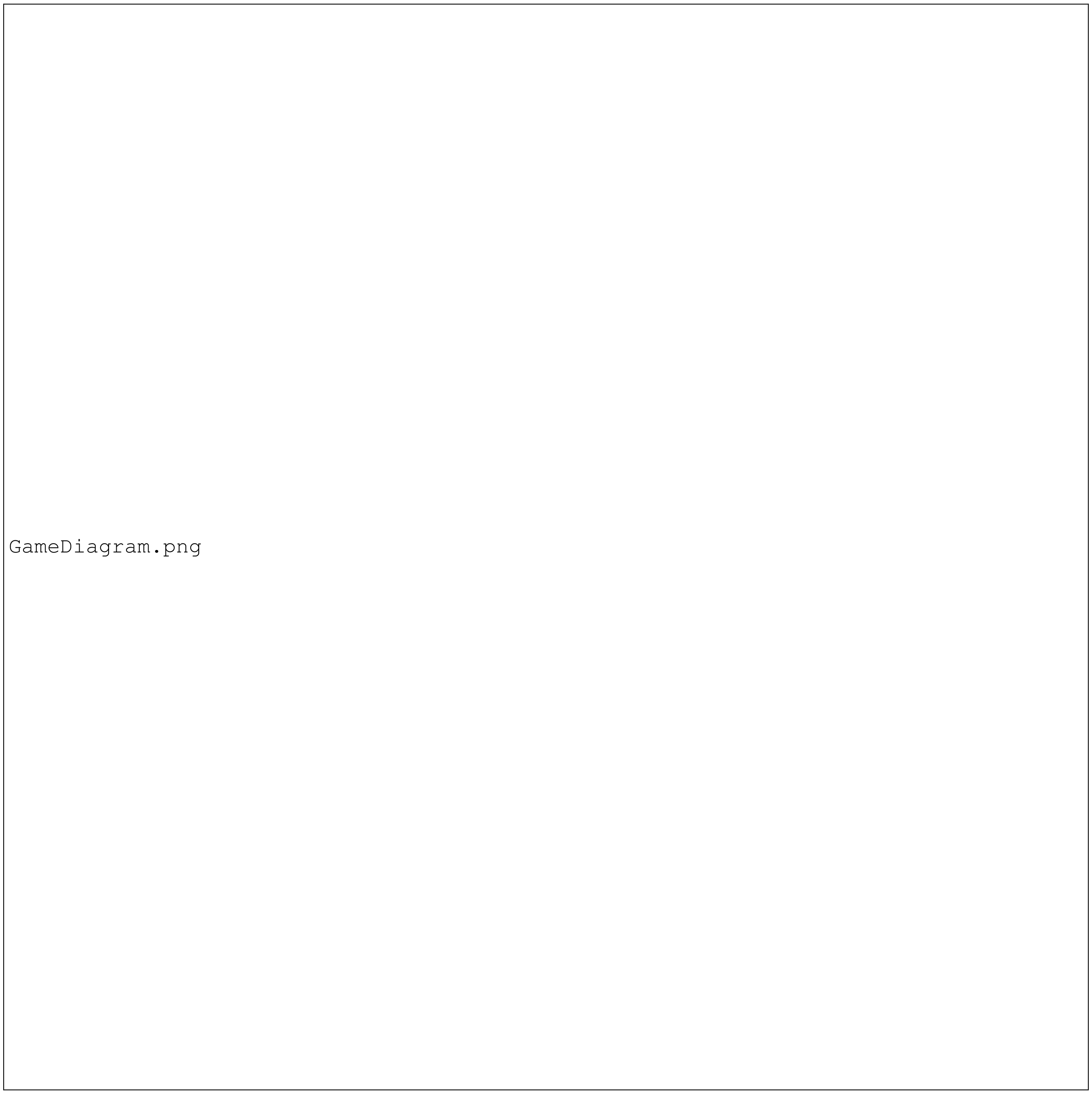


Figure 1: Schematics of the game *Cave Swing*. The agent controlling the character travels along the cave hanging from a rope of stiffness k, which can only attach to the anchors. G_x and G_y are the horizontal and vertical components of the external gravity force acting on the agent. The game finishes 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.

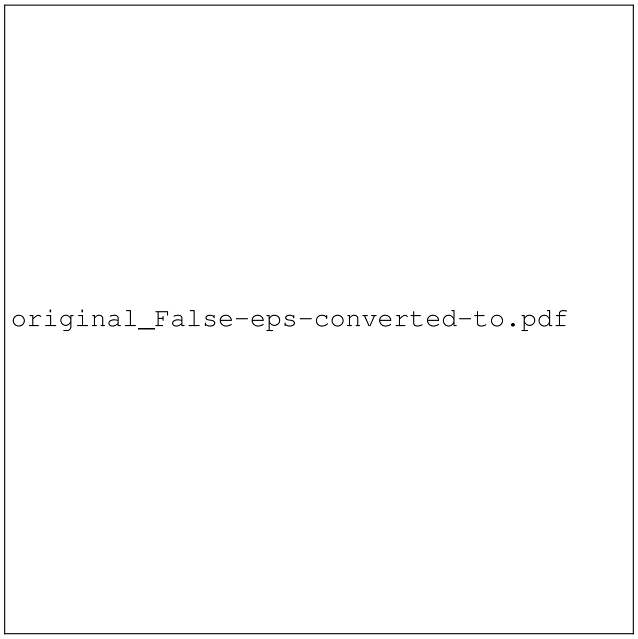
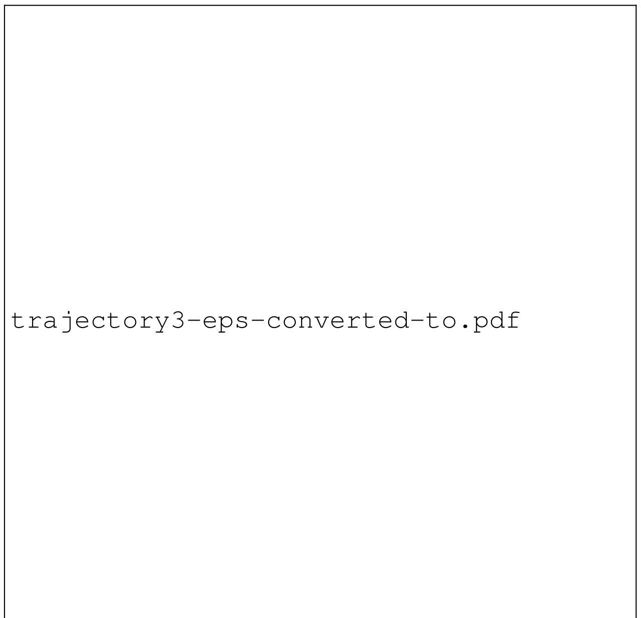


Figure 2: Average game ticks (X axis) against average game height (Y axis). The intensity of the heatmap represents the score achieved by the agent, with the given parameter set (the brighter, the higher scores were achieved by the agent). White regions correspond to areas where no behaviours were found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

(a) Trajectory plot for agent with ticks 60 and height 208

(b) Trajectory plot for agent with ticks 260 and height 140



original_True-eps-converted-to.pdf

(a) Performance with gravity vectors (2x2 strided average)

hooke_False-eps-converted-to.pdf

(b) Rope stiffness (Hooke constant)

nAnchors_False-eps-converted-to.pdf

(c) Number of anchors

lossFactor_False-eps-converted-to.pdf

(d) 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 (4a), the rope stiffness (given by the Hooke constant)(4b), the number of anchors (4c) and loss factor (4d)