MAP-Elites to Generate a Team of Agents that Elicits Diverse Automated Gameplay

Cristina Guerrero-Romero
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
c.guerreroromero@qmul.ac.uk

Diego Perez-Liebana
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
diego.perez@qmul.ac.uk

***Abstract*—The objective of this work is to provide a procedure to generate a *team* of players for a game so they are available to the developer to choose from and that can be used for automated gameplay. Our solution applies the MAP-Elites algorithm to generate agents with distinct behaviours. The resulting agents are distributed in the space of features based on the result of their actions when playing a game: *wins*, *score*, *% explored*, *interactions*, *kills*, *items collected*, etc. The criteria used as the performance of the elites does not come from how *well* an agent plays a game, but by the time it takes it to play it and determine the cell in the map it falls into. We present and implement the solution and include details about the agent used, as well as the list of heuristics created to elicit differentiated behaviours within the game, representing distinct types of players. We executed the algorithm implemented for three different games and a total of 33 configurations. The size and diversity of the pool of agents generated allow running automated gameplays in each of the games to elicit different expected behaviours. The options are limited by the distribution of the team within the space, given by the pair of features or the characteristics of the game. The methodology gives the flexibility to extend the features or modify the range of existing ones to have control over the behavioural space and, therefore, the characteristics of the generated team.**

***Index Terms*—games, automated gameplay, team generation, MAP-Elites, agent behaviour, heuristics, methodology, GVGAI**

I. Introduction

The ultimate objective of this work is to provide a resource so a *team* of agents is available to play a game in different ways. It concentrates on the means; the application of the diverse automated gameplay provided by these agents depends on the needs of the developer. These could consist of debugging, obtaining analytics, profiling, or any other; and could be employed during the development of the game or afterward.

We propose a method to generate a series of game-playing agents with distinct behaviours so they are expected to act and interact with the same game differently. These agents form a *team*, from which it should be possible to choose a *player* depending on the needs: 1) finishing the game with the highest score possible; 2) playing the game interacting with as many elements as possible while covering most of the map; 3) collecting few items but killing all the monsters; etc. We apply the MAP-Elites algorithm so each of the cells of the resulting map contains the definition of the behaviours of the agent, as well as their stats, obtained after playing the game several times. The features of the map come from these resulting stats (e.g. items, kills, % explored, etc), that will define the agent behaviour. Also, the illumination of the space of agents generated can provide insight into the level of the game. The performance of the final agents does not come from stats that measure how well they *perform* in terms of wins or score, but by the time it takes it to reach a similar pair of those final stats. It is possible to choose between different features to define the space of the map and these have an impact on the behaviours the agents are expected to elicit. Our work aims to show that this is a general technique and, as such, is tested in multiple games. However, the selection of features and agents can be made game-dependent for specific applications.

Section II presents our previous work on this area and introduces the MAP-Elites algorithm. Section III and IV give details about the implementation of the agent, heuristics, and the use of the MAP-Elites algorithm. Section V describes the games and configuration used in the experiments and Section VI presents a summary of the results. The paper is concluded in Section VII.

II. Background

A. Beyond Playing to Win: Heuristic Diversification

Our previous work introduces the diversification of behaviours in general algorithms when provided with heuristics that elicit different goals: *winning*, *exploration*, *knowledge discovery* and *knowledge estimation* [1]. Even when the core of the algorithms was the same, experiments results showed how 1) the performance between a set of them changed depending on the heuristic assigned, and 2) the behaviour and interactions in the game for each of these given heuristics were considerably different to each other. These results inspired us to outline the vision of creating a *team* of agents with distinct behaviours so they can be used to assist game design and automatic testing [2]. The generation of a team of these characteristics is the work presented in this paper.

B. MAP-Elites

The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) is an illumination algorithm that can search in a very high dimensional space created by all possible designs to find the highest performance criterion for each combinations of features [3] (e.g. *fastest* robot for each combination of *height*,

weight and *energy consumption per meter* features). This algorithm *illuminates* the relationship between performance and dimensions of interest solutions returning a set of high-performance diverse solutions. It requires defining:

1) Genome/genotype x: Candidate solution.

2) Phenotype p_x: Assessment of the characteristics of x.

3) Fitness function f_x: Measures the performance to evaluate each candidate x.

4) Feature space: N-dimensions of variation of interests.

5) Feature/behaviour function b_x: Defines the features that correspond to each x, determining the values that should be assigned to the N-dimensional vector of the map.

The relationship between genotypes, phenotypes, and the behaviour and performance of a candidate goes as follows: $x \rightarrow p_x \rightarrow b_x, f_x$. It is possible that a direct encoding exists between a characteristic of the candidates and phenotypical feature, but it is also possible that there is an indirect encoding, where a complex process maps the elements of the candidates with components of the phenotype [3].

The original version of the MAP-Elites algorithm, presented by Mouret *et al.* in [3], starts with the creation of an N-dimensional map of elites, initialized by generating random candidate solutions. In each new iteration of the algorithm, until the stop condition is reached, one of the current solutions of the map (elite) is randomly chosen and evolved to create a new candidate solution. This new candidate is then simulated to obtain its feature description, which defines its position on the map; and its performance. The new elite is automatically assigned to its correspondent cell if this is empty. If there is a current elite occupying it, the new elite replaces the previous solution if its performance is better.

The simplicity of the MAP-Elites algorithm allows its application in different disciplines, including the field of games. A constrained version of the MAP-Elites has recently been used to procedurally generate levels for bullet hell games [4] and to assist mixed-initiative design of levels in the Evolutionary Dungeon Designer (EDD) [5]. Other applications include the use of the MAP-Elites algorithm to study the relationship between the parameters of a game and the behaviour of a well-performed agent [6]; to build and balance *Hearthstone* decks illuminating the space that influences gameplay [7]; as well as to generate a pool of agents to play and perform in the card game *Hanabi* [8].

C. GVGAI Framework

The General Video Game Artificial Intelligence (GVGAI) Framework is an open-source platform that facilitates the research in General Video Game Playing (GVGP) [9]. It has been used for several competitions, resulting in the availability of a range of agents [10]. These agents are general within the framework and can play any of its games, written in the Video Game Description Language (VGDL) [11]. All of these games are single or two-player 2D arcade games. Each element of the game is represented by a sprite and the rules are triggered by collisions between them. Given the generality of the framework, the sprites are not individually identified by the agent but it is possible to recognize what category they belong to: avatar (player) sprites, sprites generated from the avatar, sprites representing Non-Playable Characters (NPCs), resources, etc. Collisions between sprites are called interactions and those that happen between the avatar or the sprites generated by it, cause events. The *sampleMCTS* agent from GVGAI is an open-loop version of Monte-Carlo Tree Search (MCTS). Details about MCTS, variants and applications can be found in [12].

III. Agent with Distinct Playing Behaviours

A. Agent: MCTS with an Interchangeable Heuristic

This work uses the *sampleMCTS* algorithm provided in the GVGAI Framework, modified and extended so that the heuristic can be plugged in externally, following an idea similar to the one described in [1]. The evaluation of a state comes from the comparison between the information of the game in the current state and the final state reached with the forward model. To achieve this, it is necessary to keep track of the data in each of the states reached by the forward model. The information tracked depends on the heuristic and its characteristics; for example, the positions visited by the avatar, the enemies killed or the events triggered in each of the states reached with the forward model.

A series of heuristics that correspond to differentiated types of players have been identified and implemented. The behaviour of the agents resulting from applying each of these heuristics would be interesting on their own but we are also interested in combining them to produce and have at our disposal a very diverse range of behaviours. Hence, we define a *parent* heuristic called *TeamBehaviourHeuristic* (III-C).

B. Heuristics: Player Behaviours

The work in [2] served as inspiration to identify and create heuristics that correspond to different behaviours or *personas* that can be encountered in a same game: *Winner*, for winning the game; *Record breaker*, for achieving a high score; *Explorer*, focused on covering as much area as possible; *Curious*, focused on interacting with the elements of the game; *Scholar*, that learns about the game and discovers elements that conform it; *Killer*, focused on removing NPCs; and *Collector*, focused on gathering resources. These heuristics have been designed and implemented for this work, some of them merged together for simplicity. They are general in the scope of GVGAI and can be used in any of its games.

1) Winning and Score: It prioritizes winning the game while maximizing the score difference. This heuristic heavily penalizes states where the game is lost.

2) Exploration: It maximizes the physical exploration of the map, which is divided into tiles. In contrast to the Exploration Maximization Heuristic (EMH) implemented in [1] (which just indicates if a position has been visited or not), *Exploration* takes into consideration the number of times each position has been visited. It prioritizes visiting those positions that haven't been visited before and, once visited, those that have been visited the least. As it favours exploring as much as possible, reaching a game over state is penalized.

3) Curiosity: It maximizes the discovery and interaction with sprites in the game, prioritizing interactions with new sprites. We consider interactions the collisions of the avatar or any sprite generated from the avatar with other sprites of the game. When no new interactions are possible, it prioritizes interactions in new locations of the game, called *curiosity* interactions [1]. The heuristic considers and gives different values (from high to low) to the following cases: a) New sprites discovered and new interactions; b) New curiosity interactions; c) Total of different curiosity interactions; and d) Number of total interactions. As it tries to investigate as much as possible, reaching a game over state is penalized.

4) Killing: It maximizes destroying Non-Playable Characters (NPCs) and penalizes end of game states. The heuristic considers "kills" those interactions between sprites generated from the avatar (hits) and sprites of the *NPC* type. When using this heuristic in the GVGAI framework, the following assumptions are made a) Enemies are killed in one shot; b) Enemies are only killed by sprites generated from the avatar. c) The avatar is not able to kill an enemy by colliding with it or by using elements of the terrain.

5) Collection: It maximizes the collection of resources and penalizes end of game states. The heuristic considers "collections" those interactions between the avatar (collisions) and resources. When using this heuristic in the GVGAI framework, it is assumed that a) Items are of the *Resource* type; b) Items can only be collected by the avatar by colliding with them.

C. TeamBehaviourHeuristic

This is the *parent* heuristic implemented that can be assigned to the agent so it can be given different combinations of the behaviours introduced in Section III-B. It is composed by the following elements, that must be assigned during its initialization: *members list*, *enabled heuristics* and *weights*:

1) Members list: $\{t_0, ..., t_m\}$, being m the total number of behaviours available. All the heuristics in this list participate in the collection of data to obtain the stats related to the behaviours they represent. However, it is possible that not all of them are *enabled* to participate in the final evaluation.

2) Enabled heuristics: $\{h_0, ..., h_n\}$, being $n <= m$ the total number of *enabled* behaviours, taken from the members list. This list contains the heuristics that participate in the evaluation of a state and that, therefore, take part in deciding the action that should be carried out next by the agent.

3) Weights: $\{w_0, ..., w_n\}$, being $n <= m$ the total number of *enabled* behaviours. Each of the weights can be given a value between 0.0 and 1.0, representing the significance of the behaviour it is assigned to.

The list of enabled heuristics and weights define the behaviour of the agent. The final value of the heuristic in a state (S) is obtained by the sum of each of the *enabled* heuristics multiplied by each of the correspondent *weights*: $H(S) = h_0 w_0 + ... + h_n w_n$.

IV. MAP-Elites Application

The MAP-Elites algorithm described in [3] has been applied to generate a *team* of agents that play a game eliciting different types of gameplay. When the algorithm finalizes, each of the cells of the map contains the description of an agent that implements the *TeamBehaviourHeuristic* given by a set of *weights* (III-C), which has been evolved during the execution. Each of the candidates is assigned to the map by observing the stats resulting from playing the game repeated times. The map is divided into a fixed number of cells given by the two feature dimensions. Each of the features is assigned a *minimum*, *maximum* and *bucket size* value, so a certain value is assigned to the bucket that contains the range it belongs to.

The performance considered to replace elites in the map is not related to how *well* they perform in the game (*wins* or *score*), as these are considered features, but in terms of how *fast* are the agents that end up getting a pair of features in a similar range. Algorithm 1 shows the pseudocode of the implemented MAP-Elites, with the following components:

1) Genotype x: Vector of weights $\{w_0, ...w_n\}$ that represents the presence of each of the *enabled* heuristics when assigned to the agent.

2) Phenotype p_x: Set of stats generated by the agent with $\{w_0, ...w_n\}$ after playing the game a certain number of times.

3) Fitness function f_x: How quick the game end is reached.

4) Feature description/behaviour function b_x: Values resulting from the actions of the agent, given by the stats obtained. These features are defined by the members list and are dependent on the game and its characteristics. The value of the chosen stats is converted into the correspondent map id.

A. Performance

In this work, we prioritize agents that would finish the game quickly. The idea is that, given two agents that behave in the same way (collide in the same MAP-Elites cell), one is considered *better* than the other if it took it least time playing the game, independently of the outcome.

B. Features

The features are computed from the stats resulting from the agent playing the game multiple times. Although the MAP-Elites supports an N-dimensional feature space, in our work we focus on two-dimensional MAP-Elites. We consider only pairs of features for simplicity in the setup and readability of the results. There is a wide list of features available:

1) Wins: Percentage of wins.

2) Score: Total amount of points at the end of the game.

3) Exploration: Different positions of the map visited.

4) Exploration percentage: Percentage of the map visited.

5) Discovery: Different types of sprites discovered.

6) Sprites interactions: Unique interactions with sprites of different types.

7) Curiosity: Unique interactions with sprites of different types in different locations of the map.

8) Collisions: Total avatar interaction with other sprites.

9) Hits: Total from-avatar sprites interactions with sprites.

10) Interactions: Total number of interactions of any type.

11) Kills: Total number of NPCs hit.

12) Items: Total number of resources collected.

Algorithm 1 Use of the MAP-Elites algorithm [3] to generate a team of agents that elicit differentiated gameplays. We use the following nomenclature: $X \leftarrow$ map, $x \leftarrow$ elite, $b \leftarrow$ feature description, $P, p \leftarrow$ performance, $\alpha \leftarrow$ *nEnabledHeuristics*, $\beta \leftarrow$ *nRandomInitialisations*

$X \leftarrow MAPElitesInitialisation()$ ▷ Add to the map α elites with only each of the behaviours enabled
▷ Generate β elites with random weights and add them to the map

for $iter = 1 \leftarrow nAlgorithmIterations$ **do**
  $x \leftarrow random\_selection(X)$ ▷ Select one of the current elites of the map randomly
  $team\_weights' \leftarrow evolution(x.team\_weights)$ ▷ Evolve the weights assigned to the elite to generate a new one
  $x' \leftarrow createGameplayElite(team\_weights')$ ▷ Agent plays the game *nGameRuns* times to get the stats
  $b' \leftarrow x'.featureStats()$ ▷ Get the stats corresponding to the features in use for the new elite
  $p' \leftarrow x'.performanceStats()$ ▷ Get the stats corresponding to the performance criteria for the new elite
  if $X(b') = \emptyset$ **or** $p' > P(b')$ **then** ▷ Appropriate cell is empty or current occupant's (elite) performance is worst
    $P(b') \leftarrow p'$
    $X(b') \leftarrow x'$
return X, P

C. Configuration

To facilitate running experiments, the implementation accepts a configuration file; allowing to dynamically set up the following options of the algorithm and its execution:

1) gameName, level: Game and id of the level the experiments are executed with.

2) agentName: Name of the algorithm to use.

3) nGameRuns: Number of times the agent (candidate) will play the game with each set of *weights* to obtain its stats.

4) featureX, featureY: Pair of features that define the map dimensions and behavioural space.

5) nRandomInitialisations: Number of random sets of *weights* that are generated to initialize the map.

6) nAlgorithmIters: Number of MAP-Elites iterations.

V. Experimental Work

The agents, heuristics and MAP-Elites implementation are in a Github repository[1]. Executable and configuration files used for the experiments can be found in an OSF repository[2].

A. Games and Levels

We selected 3 of the games available in the GVGAI Framework to carry out the experiments: *Butterflies*, *Zelda* and *Digdug*. They have different complexities in terms of wining conditions and number of sprites (*Butterflies* is the simplest, followed by *Zelda* and *Digdug*) and define different types of players, allowing a range of behaviours. The objective is to make comparisons of the results, studying how the methodology works when applied to distinct games, and showing its flexibility to adapt to different games and behaviours. The rules of the games have been modified so the assumptions described in Section III-B are met. All experiments are conducted on a same level of each game, corresponding to the screenshots.

[1]https://github.com/kisenshi/gvgai-agent-behaviour-research
[2]https://osf.io/whxm8/

Fig. 1: *Butterflies* at $t = 0$: 6 butterflies, 27 cocoons, 102 trees.

1) Butterflies (Fig. 1): The goal is colliding with all the butterflies before the time runs out or there are no more cocoons. Each collision with a butterfly adds 2 points to the score. The cocoons generate new butterflies when a butterfly collides with them. The butterflies are NPCs and they disappear when the avatar collides with them, so they would not be considered kills or items. Thus, *Killing* and *Collection* heuristics are not enabled in the experiments for this game.

2) Zelda (Fig. 2): The goal is leaving the dungeon by taking the key (resource) and opening the door before the time runs out, avoiding being killed by the monsters (NPCs) when they collide with the avatar. The player can kill the monsters by hitting them with a sword. Killing a monster increases the score by 2 and bringing the key to the door by 1.

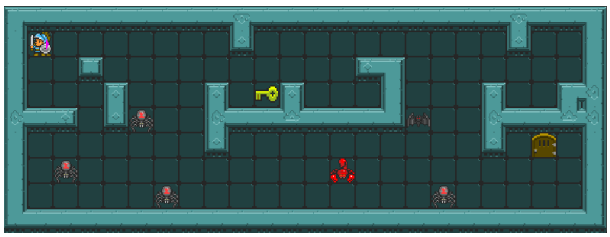


Fig. 2: *Zelda* at $t = 0$: 1 key, 1 door, 6 monsters.

3) Digdug (Fig. 3): The goal is to kill all monsters (NPCs) and collect all gems and gold (resources) of the map before the time runs out while avoiding being killed by the enemies. The player has a shovel that can be used to a) hit walls to

break them; b) hit rock blocks (those marked with a *G*) to create gold and c) hit monsters to kill them. Collecting gems increases the score by 1 and killing monsters by 2. This version has no boulders and gold is collected by colliding with it.

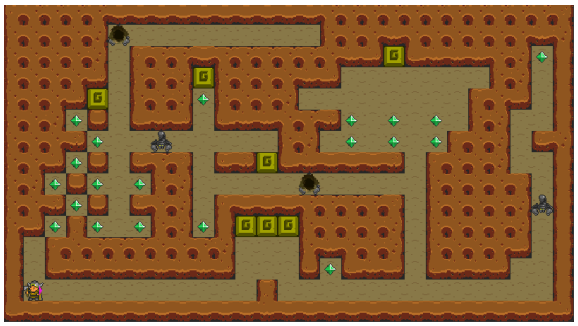


Fig. 3: *Digdug* at $t = 0$: 20 gems, 7 gold blocks, 4 initial monsters, 2 monster spawners.

B. Configuration for the experiments

All executions are carried out with an *MCTS* agent and *TeamBehaviourHeuristic*. Depending on the game and experiment, different heuristics (or behaviours) are enabled. As detailed in Section III-C, the final characteristics of the agent depend on the weights assigned to each behaviour by evolution by MAP-Elites. The algorithm has been executed with the following configurations (Section IV-C):

1) nGameRuns: 100. We consider that the most important characteristic to take into consideration for these experiments is the consistency of the stats obtained for the gameplay of the agents generated, to ascertain they are assigned to the right cell of the map. This consistency is achieved by making the agent play the game a high number of times so the outliers in the data do not skew the resulting average. A low number of data samples per game run (< 100) was not representative enough, while 1000 repetitions made the execution time too long.

2) nRandomInitialisations: 10.

3) nAlgorithmIters: 200 (*Butterflies*); 250 (*Zelda*, *Digdug*).

4) Features: We use a 2-dimensional MAP-Elites so a pair of available features (IV-B) is assigned to each experiment. The selection and range of features depend on the game and its characteristics: rules, type, the number of sprites, etc.

For simplification in the tables and results, the experiments carried out for each game are given a unique code. Table I shows these codes and the different combinations of features used in each block of experiments: **B2** is the game *Butterflies* with 2 heuristics enabled: *Winner* and *Explorer*. **B3** is again *Butterflies* with 3 heuristics enabled: *Winner*, *Explorer* and *Curious*. **Z5** is *Zelda* with all 5 heuristics enabled: *Winner*, *Explorer*, *Curious*, *Killer* and *Collector*; and **D5** is *Digdug* with all 5 heuristics enabled.

VI. Results and Discussion

The *nGameRuns* value is high (100) so each iteration of the MAP-Elites algorithm takes a long time. This execution time is also dependent on the number of heuristics enabled, as each of them requires their own calculations; as well as the complexity of the game and the average time the game over is reached by the agents. The *nAlgorithmIters* value was initially set to 250 for Z5 and D5; requiring a total of 25000 plays for each of their experiments. As the total time required for these executions was higher than the maximum time allowed, results for these games ended up with fewer iterations of the MAP-Elites: 125 for Z5 and 100 for D5. Running the experiments has resulted in the generation of a total of 33 maps, containing between 4 and 49 elites each. The group of elites generated in each map forms the *team* of available agents for future automated gameplay. We include highlights of the results below; the full set of JSON data and generated graphs can be found in an OSF repository[3].

TABLE I: Combination of feature pairs used in experiments.

Y / X	Score	Exploration Percentage	Curiosity	Collisions (B2,B3) Interactions (D5,Z5)	Kills	Items
Wins	B2 B3	B2 B3				
Exploration Percentage	B2 B3 D5 Z5		B2 B3 D5 Z5	B2 B3 D5 Z5	D5 Z5	D5
Curiosity	B2 B3 D5 Z5			B2 B3		
Collisions (B2, B3) Interactions (D5,Z5)	B2 B3 D5 Z5				D5 Z5	D5
Kills						D5

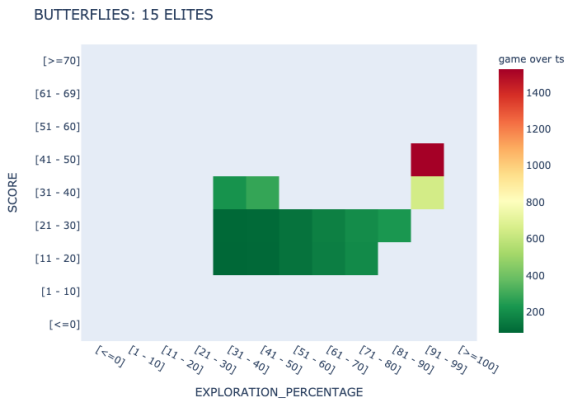
Fig. 4 shows the MAP-Elites generated for the same pair of features (*Exploration percentage* and *Score*) in each of the games and configurations. Comparing the results, we note:

1) Elites performance: Not all agents generated are equally *fast* in terms of when the game over is reached and, in some of the cases, they are quite slow. However, the performance is not as important of a factor as it was in other applications of the MAP-Elites, as the interest of the methodology implemented resides in the diversity of the solutions. The final performance value of the elite assigned to a certain cell serves as a reference to know what to expect when using the agent for automated playing and how long its execution would take on average.

2) Elites distribution: We see differences in the distribution of the resulting elites depending on the game. All the agents generated achieve an average exploration percentage higher than 21% and, within the dimensions of the game, are found in a certain range of scores. However, their alignment is very different between games: While in *Butterflies* there is a continuity in the occupied cells, the agents are *clustered* in blocks in *Zelda* and *Digdug*, having no coverage for certain exploration ranges. This aggregation is even more noticeable for the latter, where just a few exploration ranges ($21-30\%$, $61-65\%$, $71-75\%$, $>=96\%$) are included.

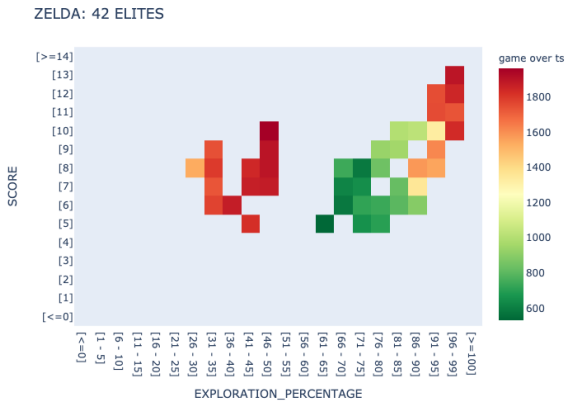
3) Diverse behaviour; diverse team: The team generated for both configurations (B2 and B3) have a similar distribution within the features space. However, including the *Curious*

[3]https://osf.io/whxm8/

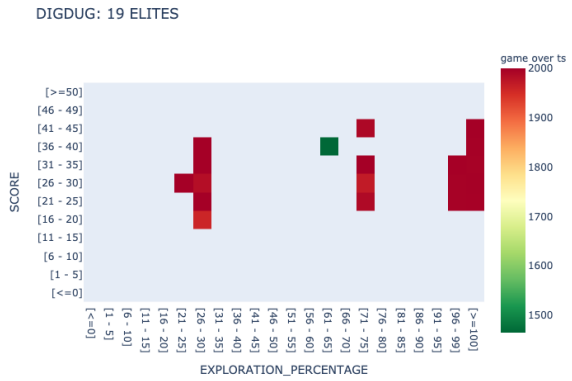


(a) *Butterflies* with *Winner* and *Explorer* behaviours (B2)

(b) *Butterflies* with *Winner*, *Explorer* and *Curious* behaviours (B3)



(c) *Zelda* with *Winner*, *Explorer*, *Curious*, *Killer*, *Collector* behaviours (Z5)



(d) *Digdug* with *Winner*, *Explorer*, *Curious*, *Killer*, *Collector* behaviours (D5)

Fig. 4: Resulting MAP-Elites for Feature pair *Exploration percentage* and *Score* for all games and configurations (*B2*, *B3*. *Z5*, *D5*). The team of agents generated for each experiment is of different sizes (15, 20, 42 and 19) and the behaviours and distribution in the space depends on the characteristics of the game.

as additional behaviour allows the generation of more elites, increasing the choices of agents when looking for a high score $(41-50)$: B2 only produces an agent capable of reaching this high score, tight to a high exploration percentage $(91-99\%)$. B3, in contrast, generates a total of 5 agents related to this high score, each of them in different ranges of exploration $(31-40\%,\ 61-70\%,\ 71-80\%,\ 91-99\%$ and $100\%)$. We observe similar results in maps resulting for other pair of features: *Exploration percentage* x *Collisions* results in 10 final elites for B2 and 49 for B3; and *Curiosity* x *Score* in 4 and 16 respectively. We can infer that having more behaviours enabled generates a more diverse team.

4) Team size: The number of elites generated for *Zelda* is 42, a much higher number than the ones generated for *Digdug* (19). When looking at the number of elites at the iteration 100 for *Z5*, this value was 39. In general, the team size resulting for the *D5* experiments is not very high compared to *Butterflies* and *Zelda*. The reasons could be that *Digdug* is such a complex game that requires many more iterations, or that it is not possible to elicit further behaviours in *Digdug* and, as a result, the pool of agents available will be smaller.

Fig. 5 shows the resulting MAP-Elites for B3 when the feature pair is *Exploration Percentage* and *Collisions*. The solution generates 49 agents. The higher the number of collisions, the higher its relationship with the exploration percentage and the average game over time of the agents. It is possible to select agents to cover different percentages of the map but, when requiring an agent able to obtain more than 300 collisions, this selection gets reduced to agents that attain a very high exploration $(>90\%)$. Yet, if the target is obtaining more than 1000 collisions, two additional agents are found in a lower percentage range. Taking a look at the stats for the latter, we see how their curiosity value average is 100.64 and 103.76. There are 102 trees and 33 butterflies in the game, so the behaviour expected from these agents would be interacting mostly with the trees, sticking to the borders of the map, and avoiding getting into the *open* areas, unless they get the chance to interact with a butterfly at reach.

Fig. 6 shows the resulting MAP-Elites for Z5 when the feature pair is *Interactions* and *Kills*. The solution generates 35

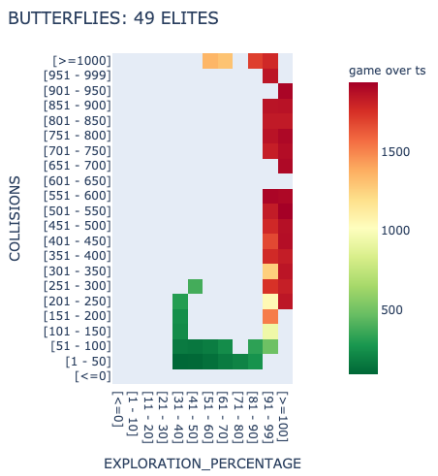


Fig. 5: B3 MAP-Elites for Feature pair *Exploration percentage* and *Collisions*

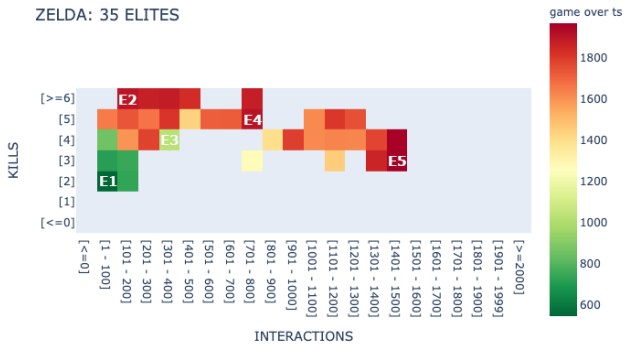


Fig. 6: Z5 MAP-Elites for Feature pair *Interactions* and *Kills*. Table II includes details about the elites highlighted (En).

agents capable of killing at least 2 monsters while interacting with the elements of the game in different capacities. The range of the average of iterations elicit by the team goes from $[1-100]$ to $[1401-1500]$. The pool of agents in the team where we can find the most diverse skill at killing monsters is tight to the lowest number of iterations ($[1-200]$). When requiring an agent to reach higher iterations, it is expected that the agent ends up killing at least 4 or 5 monsters when playing the game. If it is desired that an agent kills all monsters when playing, the pool is reduced to 5 agents, any of them able to reach an average of interactions higher than 800.

Table II contains details about 5 of the resulting agents for this experiment, highlighted in Fig. 6. The features in the MAP-Elites work as guidance on diversifying the behavioural space but the stats also show great differences, exhibiting diversity in the way each of the agents relates to the game in general. This information helps to have a better understanding of what to expect from an automated gameplay when using each agent from the team: While *E1* would always be expected to win and *E5* would always be expected to lose, the latter would achieve a slightly higher score than the former and interact with elements of the game much more. Given this

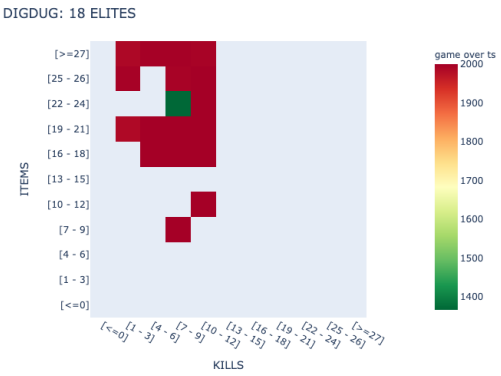


Fig. 7: D5 MAP-Elites for Feature pair *Kills* and *Items*

information, plus *E5*'s low exploration skills and the fact that it rarely picks the key (the only item in the game), we deduce that this agent would rarely reach the area where the key is, expecting it to stick to the left and bottom zones of the map.

Most of the solutions of the MAP-Elites generated for *Digdug* show an average of end of game ticks very close to the maximum allowed (2000), the lowest value being at 1200. Fig. 7 shows the resulting MAP-Elites for D5 when the feature pair is *Kills* and *Items*. Only one of the 18 agents available in the pool has a lower average of game ticks (1400). This particular agent is in the range of $[22-24]$ items and $[7-9]$ kills, so it is on the top tier of the agents encountered in this space, although it is not the one that reaches the most number of monster kills and items. All the agents can collect at least 7 items and kill between 7 and 12 monsters, but they rarely win the game (the highest win rate find in the team is 0.05%); so they are either killed by a monster or play for its whole duration. Therefore, future automated gameplays for this game are expected to be slow, independently of the description of the agent chosen.

VII. Conclusions and Future Work

This work proposes and implements an adaptation of the MAP-Elites to generate a *team* of agents with distinct behaviours so they can be used for automated gameplay. The solution provides a number of agents in a feature space; the location of each of them gives an idea of what to expect when they are selected to play the game. The features of the map are defined by the results of the actions of the agents: *wins*, *score*, *exploration*, *kills*, *items collected*, etc. The performance of the agent to be assigned to a cell to the map is given by the time it takes them to play the game obtaining a similar range of values for the features, and not how *well* they perform on it in terms of score or wins. The algorithm used to play the game is the *MCTS*, which has been adapted so it is possible to provide a list of *heuristics* and corresponding *weights*. The heuristics implemented represent type of players that can be found on the games: *Winner*, *Explorer*, *Curious*, *Killer* and *Collector*. The list of features and heuristics used in this work can be adapted and extended based on the game under consideration

TABLE II: Details about elites highlighted (En) in Fig 6. Includes the description of the *weights* of each behaviour (*Winner*, *Explorer*, *Curious*, *Killer*, *Collector*) as well as the associated value of their features, game over timestamp (ts) and average stats resulting when an agent with this description plays the game (*Zelda*) 100 times.

	E1	**E2**	**E3**	**E4**	**E5**
Behaviour *weights*	[0.66, 0.13, 0.01, 0.04, 0.16]	[0.23, 0.13, 0.01, 0.64, 0.16]	[1.0, 0.13, 0.68, 0.58, 0.16]	[0.52, 0.13, 0.68, 0.64, 0.16]	[0.0, 0.0, 0.8, 0.0, 0.77]
Feature X: *Interactions*	37.91	150.07	314.52	713.64	1411.24
Feature Y: *Kills*	1.95	5.53	3.62	5.38	3.32
Game over ts	544.32	1905.71	1023.71	1912.60	1970.68
Gameplay stats (average of 100 plays)					
% Wins	91.99%	6.00%	70.00%	8.00%	0.00%
Score	5.72	12.12	8.8	11.81	7.18
Exploration percentage	70.35%	95.80%	78.00%	96.01%	34.61%
Unique interactions	3.33	4.99	4.46	5.07	3.75
Curiosity	25.70	80.65	61.82	90.95	37.61
Collisions	35.96	144.54	310.90	708.26	1407.92
Hits	1.95	5.53	3.62	5.38	3.32
Kills	1.95	5.53	3.62	5.38	3.32
Items	0.97	0.98	0.98	0.99	0.60

and the needs of the user. The MAP-Elites has been executed for 3 games of different characteristics and complexity with various configurations. In one of the games, two different sets of heuristics were enabled, distinguishing between 4 blocks of experiments (B2, B3, Z5, D5). A total of 33 different configurations of the MAP-Elites were executed and 95, 227, 253 and 175 agents were generated respectively. The size and diversity of the pools of agents generated by the MAP-Elites make it possible to find agents to run automated game playthroughs in each of the games based on different needs. The options are limited by the distribution of the *team* within the space, caused by the pair of features, the characteristics of the game, and the *enabled heuristics* set for the agent.

The implementation and experiments were carried out in the GVGAI Framework: the generality of the algorithm and heuristics within the framework allows to execute the methodology in different games. For simplicity, only a pair of features were set in each experiment, but the MAP-Elites algorithm is not limited to a 2-dimensional feature space. The approach can be extended to N-dimensional spaces to generate a more diverse and versatile archive of agents.

The results and stats refer to the level the algorithm was executed on. The portability of the agents generated in each game was out of scope for this paper, but it is possible to study it with further experimentation. The generality of the heuristics used allows for a fair comparison running the agents in other levels. Future work also includes extending the heuristics so it allows setting new player types and features, using larger dimensions for the MAP-Elites, alternative algorithms; as well as porting the idea to more complex but specific games, using heuristics particularly tuned for them.

A shortcoming of the technique is the necessity of coming up with the heuristics to describe the behaviours applied to the agent. We propose a list of heuristics in our previous work [2] that can serve as inspiration, but we believe that game designers can also come up with other behaviours based on player types suitable for the game. The downside of this adaptation of the MAP-Elites is its long execution time, which comes from the necessity of having to play the game with each candidate solution a high number of times, to make sure it is assigned to the right cell. This repetitive execution causes that each iteration of the MAP-Elites takes a long time, slowing the overall processing time. However, in contrast to other usages of the MAP-Elites in games, this application is not expected to be used "online". The ultimate objective of this work is to provide a resource so a *team* of agents is available for a game, either during its development or afterward. Some possible applications are using these agents for automated gameplay to 1) find bugs and code debugging; 2) obtain game analytics; 3) check game performance using profiling tools.

ACKNOWLEDGMENT

Work funded by the EPSRC CDT IGGI EP/L015846/1.

REFERENCES

[1] C. Guerrero-Romero, A. Louis, and D. Perez-Liebana, "Beyond Playing to Win: Diversifying Heuristics for GVGAI," in *IEEE CIG*, 2017, pp. 118–125.

[2] C. Guerrero-Romero, S. M. Lucas, and D. Perez-Liebana, "Using a Team of General AI Algorithms to Assist Game Design and Testing," in *IEEE CIG*, 2018, pp. 1–8.

[3] J.-B. Mouret and J. Clune, "Illuminating Search Spaces by Mapping Elites," *arXiv preprint arXiv:1504.04909*, 2015.

[4] A. Khalifa, S. Lee, A. Nealen, and J. Togelius, "Talakat: Bullet Hell Generation through Constrained MAP-Elites," in *Proc. of The Genetic and Evolutionary Computation Conference*, 2018, pp. 1047–1054.

[5] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "Empowering Quality Diversity in Dungeon Design with Interactive Constrained MAP-Elites," in *IEEE Conference on Games (CoG)*, 2019, pp. 1–8.

[6] M. Balla, A. Barahona-Rıos, A. Katona *et al.*, "Illuminating Game Space Using MAP-Elites for Assisting Video Game Design," in *11th AISB Symposium on AI & Games (AI&G)*, 2021, pp. 1–6.

[7] M. C. Fontaine, S. Lee *et al.*, "Mapping Hearthstone Deck Spaces through Map-Elites with Sliding Boundaries," in *Proc. of The Genetic and Evolutionary Computation Conference*, 2019, pp. 161–169.

[8] R. Canaan, J. Togelius, A. Nealen, and S. Menzel, "Diverse Agents for Ad-Hoc Cooperation in Hanabi," in *IEEE CoG*, 2019, pp. 1–8.

[9] J. Levine, C. Bates Congdon, M. Ebner *et al.*, "General Video Game Playing," in *Artificial and Computational Intelligence in Games*, ser. Dagstuhl Follow-Ups. Dagstuhl Publishing, nov 2013, p. 8.

[10] D. Perez-Liebana, J. Liu *et al.*, "General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms," *IEEE Trans. on Games*, vol. 11:3, pp. 195–214, 2019.

[11] T. Schaul, "A Video Game Description Language for Model-Based or Interactive Learning," in *IEEE Conference on Computational Inteligence in Games (CIG)*, 2013, pp. 1–8.

[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. on CI and AI in games*, vol. 4:1, pp. 1–43, 2012.