# Beyond Playing to Win: Creating a Team of Agents with Distinct Behaviours for Automated Gameplay

Cristina Guerrero-Romero, Simon Lucas, and Diego Perez-Liebana

***Abstract*—We present an approach to generate a *team* of General Video Game Playing (GVGP) agents with differentiated behaviours that can ultimately assist in the game development process. We consider the agent behaviour as the corresponding outcomes of playing the game: rate of wins, score, exploration, enemies killed, items collected, etc. We create and identify agents that are expected to achieve particular goals but do not necessarily simulate human behaviour during gameplay. We present a solution that, by *heuristic diversification*, provides a controller with different heuristics and a corresponding set of *weights*; driving its actions. Given the simplicity of this *behaviour-encoding* and its easiness to evolve, we use MAP-Elites to generate different solutions that elicit particular behaviours and assemble a *team*. The resulting agents are allocated in a feature space, used to identify the expectations of each of them. We generate a *team* for $4$ games of the General Video Game AI (GVGAI) Framework and find $6$ different *behaviour-type* agents in each. We include an experiment to check the portability of these agents when playing alternative levels and an exploratory work aiming to use them to detect design flaws in game levels.**

***Index Terms*—games, artificial intelligence, automated gameplay, agent behaviour, heuristics, heuristic diversification, playtesting, MAP-Elites, GVGP.**

## I. Introduction

VIDEO game playing approaches, and especially General Video Game Playing (GVGP), focus on creating agents with the ultimate goal of winning the game and, ideally, being the best at it. However, when human players play a game, they do not always focus on winning first. Even when they ultimately play to win, they present different ways to interact and react to the game, driven by their interests. During the development of a game, the existence of this diversity of player behaviours, known as player-types or *personas*, holds a relevant role in its design and the expected experience of the players. Furthermore, testing and Quality Assurance (QA) techniques are also directed to detect bugs and design flaws by asking testers to focus on specific tasks. These goals can be as simple as going through the level of a game, interacting with the walls and objects to detect errors with collisions, unintentional shortcuts, or other bugs that can break the gameplay. Game development is an incremental process, and changes often occur. Every change can have unexpected effects on the rest of the game, so QA tests should ideally be carried out after every new modification. However, it is not always possible because manual testing is tedious and requires organisation and resources. Artificial Intelligence (AI) agents are being used for testing to address those limitations, but these approaches are typically game-dependent. As a result, such agents also need to be updated after certain modifications to the game to fit its changes and maintain their purpose.

Authors are with the Game AI Group at Queen Mary University of London.

The ultimate goal of our work is to assist in the game development and testing processes by facilitating developers with a method to automatically trigger tests and tools during the development of the game. We introduce a new step towards tackling such a complex topic. Rather than removing humans entirely, we aim to provide the necessary means to make some of the tasks related to testing and QA easier. Our proposal is to use general agents, so they can adapt to changes in rules and levels, and play the game when needed without having to modify them. These agents are not expected to simulate players or QA testers, but they have goals beyond winning that allow them to elicit a particular behaviour. Their behaviour originates from focusing on different goals driven by their heuristics and accomplishing assorted tasks. An example would be colliding with the walls and objects distributed on the level. As game testers would do, these agents can still trigger errors and log information when playing the game, allowing developers to identify such bugs quicker or immediately after a change is made to the game. We believe that having a range of agents where different behaviours and proficiencies can be identified, as well as a method to generate new ones when needed, addresses the limitations of current approaches.

The work presented in this paper builds up from our previous one. We use *heuristic diversification* for the implementation of the agents, which is a technique in GVGP that we first introduced in [1]. We provided general agents with four different behaviours and compared their performance when taking as reference features related to each of them. The core of the algorithms was unchanged as the evaluation function was isolated and provided externally. The results showed differences in the performance of the algorithms for each of the heuristics and in their behaviours when playing the games. Those results inspired the proposal of using a *team* of general agents with distinct behaviours to assist in game development and testing [2], which is a long-term vision. The work presented in [3] was the first step towards reaching such vision by defining and implementing a technical solution that makes use of the MAP-Elites algorithm to generate a *team* with those characteristics. That work has been extended in this paper. This extension incorporates additional features for the MAP-Elites, a new game with distinctive characteristics to the ones used in the initial experiments (*Sheriff*), the presentation of a demo that allows interacting with the results and visualising the agents' gameplays, as well as experimentation and a proof of concept based on the proposed vision. We identify various *behaviour-type* agents from the *team* and test

their portability to study if their abilities and expectations are transferable to unseen levels, different from those used for their generation. Proving that this portability is possible is another step towards being able to put into practice the envisioned usage of the agents for automated testing. Finally, we conduct initial exploratory work to check the potential of using these agents to test the design and validity of levels by running the agents in 'broken' levels to study their outcomes.

### A. Scope of Research and Definitions

This section defines a series of terms used through the paper.

*a) Automated gameplay:* Refers to having an agent to play a game, instead of being played manually by a human.

*b) GVGP:* We use agents that play games and use heuristics that are general, i.e. the agents do not make decisions based on hard-coded or explicit information about the game. The heuristics use, in their calculations, elements available in the framework for every GVGAI game.

*c) Single-player, tile-based 2D video games:* Only games supported by the GVGAI Framework are in the scope of our research. These 2D tile-based games are developed in Video Game Description Language (VGDL) and their rules are triggered by interaction between the sprites. Each game session has a limited time set by default and is formed by a non-scrollable level fixed to the game screen. The AI takes control of the player, who is represented as an avatar and can move right, left, up, down, and act. These games do not have continuous physics or include non-avatar puzzles or strategy games. Similarly, we focus on single-player games, rather than other multi-agent simulations. In any case, the methods proposed in this paper are not limited to other types of games, which could be a matter of future research.

*d) Search algorithms:* We use agents with a Forward Model at their disposal. We do not look into learning approaches, human-like, or imitation learning techniques.

*e) Testing:* This work focus on the technical domain that ensures that no bugs break the playability, influence the gameplay, or affect any other element or characteristic of the game as an artifact.

*f) Team:* We use this term to refer to the pool of agents. They serve the same purpose, similarly to sports, where it describes a group of people (or athletes) that train together and compete, representing the same club. While in some sports, the competition can involve the athletes playing simultaneously and collaborating (e.g. basketball); in others like fencing, the team members compete individually while still representing the club. We base our perception of the *team* on the latter, meaning that the agents of this team are not expected to play the game simultaneously or collaborate between them.

*g) Goal:* The ultimate objective(s) of one of the agents in the team for when playing the game. Examples of goals are exploring the level, collecting items, or killing enemies.

*h) Heuristic:* "Criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some *goal*" [4]. The *heuristic value* is a game state evaluation, given in relative terms to a high value ($H$). How a heuristic is defined and implemented affects the agents' decisions while playing.

*i) Behaviour:* The way the agent ultimately reacts and interacts with the game. We consider the agents' behaviour as the results of their play (gameplay stats and features).

## II. Background

### A. Player-Types, Personas and Automated Testing

Although there is considerable secrecy around video games and the strategies followed by game studios during development, some examples of automated testing exist. Some of these focus in highlighting build or performance errors [5], the robustness of gameplay features [6] or creating human-like agents for automatic testing [7]. Automated testing and AI-assisted game design is a recurring topic, and various approaches exist to tackle it in different areas [8], [9]. However, most of these solutions fit closely the rules of the game and need to be updated to adapt to changes during development.

The concept of *persona* is used in Software Development to identify end-users and assist in the design of the product. It is also applicable to game development by taking into consideration the existence of *player-types* that can be identified in a game, resulting from humans playing in different ways based on their motivation [10], [11]. Players display distinct behaviours that are also dependent on the characteristics of the game, and metrics of their gameplay can be used to identify these and define *play-personas* [12]. *Personas* have also been generated procedurally to be used for automated testing [13], where the authors identified $4$ types of players in a dungeon game (runner, monster killer, treasure collector, and completionist) and evolved a mathematical formulae to replace the UCB1 equation of Monte Carlo Tree-Search (MCTS) [14] to simulate each behaviour. They analysed the performance of these *procedural personas* in various levels. Our work is inspired by [13] but it aims to have a more general and portable approach, capable of being applied to several different games without having to design specific types, or utility functions, to fit the game under consideration. Extending the idea to use general agents, developed with general goals that apply to several different games and adapt to changes in the levels can provide significant advantages. We name our agents *behaviour-types* instead of *player-types* or *personas*, as these concepts are either related to game design or with agents mimicking or modelling human behaviour. We look at the behaviour of the agents as the results of their gameplay, without necessarily resembling human players when playing.

A recent approach in Reinforcement Learning provides a diverse range of play-styles by making the rewards external and agnostic to the algorithms and has been applied to ACER [15]. We follow a similar idea of diversification but in the scope of search algorithms and present a solution to generate this diversity automatically.

### B. MAP-Elites

The Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) is an *illumination* algorithm that belongs to a family that includes Novelty Search, Novelty Search with Local Competition, and DeLeNox [16]. In MAP-Elites, the algorithm

searches candidates within all possibilities, finds diverse solutions, and locates them in a behavioural space [17]. There are a series of elements that need to be identified and take part in the algorithm: the description of a candidate (genotype $x$); its characteristics (phenotype $p_x$); the features that conform to the N-dimensional map and describe the values of interest corresponding to the candidate and its location in the map (feature/behaviour function $b_x$); and the performance of the candidate to evaluate its quality compared to other solutions (fitness function $f_x$). There is a relationship between these elements that can be direct or indirect, but it is represented as $x \rightarrow p_x \rightarrow b_x, f_x$. We identify each of these elements, and describe the algorithm in Section III-B. There have been a number of improvements of the MAP-Elites algorithm over the years, such as CMA-ME [18], which applies MAP-Elites with Covariance Matrix Adaptation in standard continuous optimization benchmarks. Our work uses the vanilla version of MAP-Elites, which has shown to be effective for creating different gameplay behaviours, and we leave testing other advance variants for future research.

MAP-Elites has been applied to various disciplines[1], including the field of games. In this area, it has been used to: generate levels [19], [20], influence gameplay [21], study the behavioural space of a game [22], examine the relationship between the parameters of the game and the behaviour of an agent [23] and generate gameplaying agents [24], [25]. The generation of such gameplaying agents is usually focused on the performance of the agents in terms of score or win rate. As we describe later in detail, our approach focuses on the diversity of the possible behaviours instead, and use the performance (End-of-Game, EoG, ticks) to merely break ties between two agents that obtain similar gameplay stats.

### C. GVGAI Framework

The General Video Game Artificial Intelligence (GVGAI) Framework is an Open-Source JAVA platform that facilitates the research in General Video Game Playing (GVGP) and provides a series of working controllers and 2D arcade-like games [26]. Agents do not have access to the details of the rules of the games or their characteristics but can simulate future states by executing a forward model and distinguish between types of sprites: player (avatar), resources, Non-Player Characters (NPCs), immovable elements, etc. When two sprites get in contact and one of them is the avatar or an element generated by it, an event is triggered. The GVGAI Framework allows running experiments in different games and levels without modifying the algorithms or heuristics. These are general within the framework; providing adaptability and a quick experimental set-up, allowing us to test our approach in different scenarios. We use the *sampleMCTS* controller provided by the framework, which is a vanilla implementation of the Open-Loop Monte Carlo Tree-Search (OLMCTS) algorithm [14], [27], with some modifications described next.

[1]https://quality-diversity.github.io/papers has a large collection of examples.

## III. Generating Agents with Distinct Behaviours

The objective is to create and have available a range of agents with differentiated behaviours and identifiable proficiency. These behaviours should be elicited by their heuristics and not the parameters of the controller, so they can easily vary without having to make updates to their core. We present: 1) an agent with *heuristic diversification*; 2) a *parent* heuristic that allows to combine different goals and obtain a final reward based on the *behaviour-encoding* provided; and 3) the use of the MAP-Elites algorithm to generate distinct behaviours.

### A. Agent: OLMCTS Using MemberBehaviour Heuristic

We follow an approach used in our previous work called *heuristic diversification*, which we have previously applied to single-player search algorithms from the GVGAI Framework [1]. The idea is to provide the goals (heuristics) to a controller externally, so they can easily change without having to modify the core of the algorithm. *Heuristic diversification* is applicable to any search algorithm, but in this work we use OLMCTS. We take the *sampleMCTS* provided by the framework and modify it to isolate and extract the evaluation function so it can be provided during its initialisation. We also store temporary information about the future states visited with the forward model to be able to make accurate calculations of the heuristics. This temporary information depends on the heuristics used and their characteristics, detailed in Section IV-A. During gameplay, the reward of a state is calculated and provided to the OLMCTS by the external heuristic, *MemberBehaviour* (Fig 1). This *parent* heuristic allows to combine independent heuristics. It receives a list of goals and corresponding weights, allowing to choose the importance given to each different objective. This approach allows more diversity of behaviours than the one given by simply swapping between different goals.

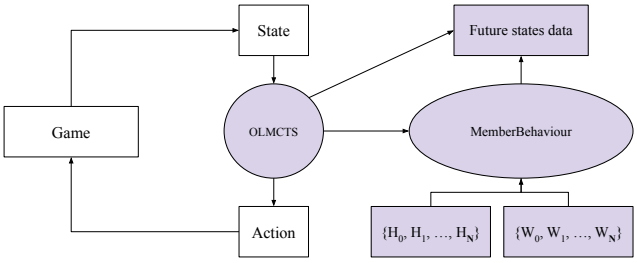



Fig. 1. OLMCTS agent with *MemberBehaviour*, which allows to provide a list of goals (heuristics) externally and assign a weight to each of them.

The objectives when designing the *MemberBehaviour* heuristic (Fig. 2) were: allowing an easy set-up of a list of different goals; gathering stats from each of them; being able to combine the goals with a solution easy to generate and evolve. As a solution, we define three elements:

*1) Members goals:* $\{t_0, ..., t_m\}$*:* $m$ is the total number of goals available to gather stats about the gameplay of the agent.

*2) Enabled heuristics:* $\{h_0, ..., h_n\}$*:* $n(\leq m)$ is the total number of enabled goals, taken from the available ones. Only the enabled goals take part in the evaluation of the state and, therefore, in the calculation of the final heuristic.

*3) Weights:* $W = \{w_0, ..., w_n\}$*:* $n$ is the number of enabled heuristics and each weight is assigned to one of them. The weight gets a value between $[0.0, 1.0]$. This value determines the importance that the corresponding goal is given in the final calculation. Therefore, $W$ ultimately describes the final behaviour of the agent. It is easy to define, generate, and evolve, so it can be used as an individual in MAP-Elites.

The final value of the heuristic resulting from evaluating a certain state comes from combining all the enabled heuristics, normalised to a common range $[0.0, 1.0]$. The heuristic obtained for each goal is the result of the linear combination:

$$H(S) = h_0 w_0 + ... + h_n w_n$$

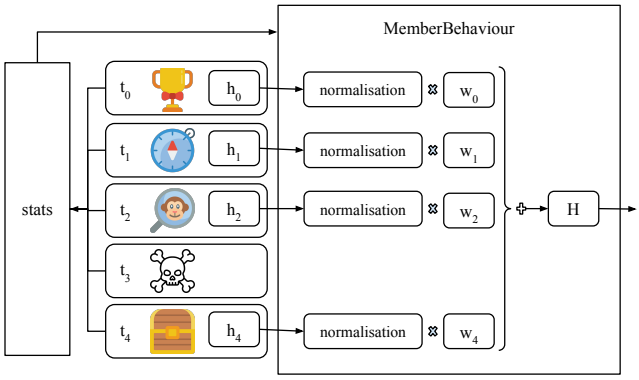



Fig. 2. Example of *MemberBehaviour*. There are 5 goals but only 4 of them are enabled and take part in the calculation of the heuristic $H$.
*Icons made by Smashicons: https://www.flaticon.com/authors/smashicons.*

## B. Using MAP-Elites to Generate the Team

In our approach, we identify the elements required for the MAP-Elites algorithm as follows:

*1) Genotype* $x$*:* Vector of weights $W = \{w_0, ..., w_n\}$ for the heuristic function as shown in Fig. 2.

*2) Phenotype* $p_x$*:* Resulting stats when providing the agent with $W$ and playing the game several times.

*3) Feature function* $b_x$*:* Characteristics of the gameplay of the agent, taken from the stats: wins, score, exploration percentage, interactions, etc. These illustrate the results of the behaviour of the agent when playing the game, and it is the information we are interested in to get a diverse team.

*4) Fitness function* $f_x$*:* How quick the end of the game (EoG) is reached. We establish that between two agents with a similar set of features (stats), one is better if the game ends earlier; i.e. if they get similar stats in less time, they are more efficient at the task.

The pseudocode of our application of MAP-Elites is included in Algorithm 1. The initialisation of the MAP-Elites is carried out in two steps. First, we define the baseline candidates. Each of these make use of just one of the goals enabled, i.e. the weight assigned to their corresponding heuristic is set to 1.0 while the rest are assigned to 0.0. After this, the second initialisation step generates a series of random candidates (weights are given random values between $[0.0, 1.0]$). In both cases, each candidate is assigned with the corresponding *behaviour-encoding* and assigned to their corresponding cell. These two steps expect to 1) provide a baseline of behaviours by using each goal independently, and 2) provide a baseline of diversity given by the random assignments. The algorithm starts its iterations until a certain limit, provided as an algorithm parameter in the configuration, is reached. In each iteration, a cell is selected uniformly at random. The weights are evolved with a simple mutation hill climber (one of the weights is randomly updated to a new value between $[0.0, 1.0]$) and that new candidate is assigned to its corresponding cell. This simple evolutionary method proved enough to generate a good range of different behaviours.

The assignment, both during initialisation and during the main execution of the algorithm, works as follows. First, the weights are provided to the agent and it plays the game to obtain the resulting stats. Then, these stats are used to get the features that constitute the map and assign the candidate to its corresponding cell. 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 the resulting value obtained by the agent is assigned to the bucket that contains the range it belongs to. If the cell is empty, the agent is directly assigned to it and the algorithm moves to the next iteration. If the cell is occupied, the performance of both candidates are compared, keeping the one with better performance as the elite within the cell. When the algorithm finishes, the map contains a set of *behaviour-encodings* (W) of a diverse range of agents. Their behaviour is implied by their location and set of features assigned in the aforesaid map. This group of agents is what we call the *team*. An example of a map generated is included in Fig. 3. If various maps are generated for the same game and enabled goals, the *team* is constituted by the ensemble of the different maps.

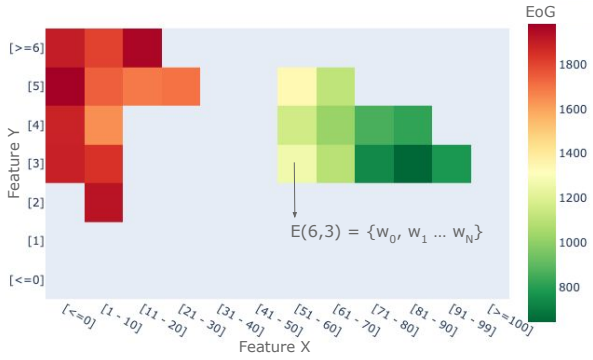



Fig. 3. Example of a resulting MAP-Elites for a two-dimensional map with Features X and Y. These features are in different ranges and have buckets of different sizes. The map represents a *team* of 23 agents, and each cell contains the *behavioural-encoding* ($W$) of each of them.

## C. Interactive Tool for Gameplay Visualisation

We are interested in the characteristics of each of the agents and studying their behaviour when playing the game. The graphs generated give information about the distribution of the agents in the feature space, and it is possible to identify what to expect from them by looking at their correspondent

**Algorithm 1** Use of the MAP-Elites algorithm to generate the *team*.
Nomenclature: $X \leftarrow$ solutions (map of elites); $P \leftarrow$ solutions' performances; $x \leftarrow$ elite; $W \leftarrow$ agent description; $x' \leftarrow$ candidate solution; $b' \leftarrow$ feature descriptor of $x'$; $p' \leftarrow$ performance of $x'$; $\alpha \leftarrow$ *nEnabledHeuristics*, $\beta \leftarrow$ *nRandomInitialisations*.

**procedure** MAP-ELITES-TEAM
  $X \leftarrow MAPElitesInitialisation()$ ▷ adds to the map $\alpha$ elites with only each of the goals enabled
  ▷ generatse $\beta$ elites with random weights and add them to the map
  **for** $iter = 1 \leftarrow nAlgorithmIterations$ **do**
    $x \leftarrow random\_selection(X)$
    $W \leftarrow behaviourWeights(x)$
    $W' \leftarrow evolution(W)$ ▷ evolves elite's weights to generate a new candidate
    $x' \leftarrow createGameplayElite(W')$ ▷ the agent plays the game *nGameRuns*
    $b' \leftarrow x'.featureStats()$ ▷ candidate's stats of the map features
    $p' \leftarrow x'.performanceStats()$ ▷ candidate's stats of the performance criteria
    **if** $X(b') = \emptyset$ **or** $p' > P(b')$ **then**
      $P(b') \leftarrow p'$
      $X(b') \leftarrow x'$
  **return** $X$, $P$

cells. However, static images have a limitation when looking at a more extensive analysis. Moreover, the results are produced in JSON files, so going through the data of the agents assigned to each cell of the map to get detailed information about them is a tedious process. To facilitate the processing and reading of the results, we created an online interactive tool[2], which complements and extends our work, by allowing access to the results of the experiments. It contains the following features:

*1) Welcome page:* Includes an overview of the tool, definitions, and instructions. It provides details about the goals, games, and features used to generate the MAP-Elites.

*2) Data selection:* Choose the game and pair of features to load and visualise the corresponding MAP-Elites generated.

*3) Team visualisation:* Presents an overview of the game, the heuristics enabled for the agents, the features used, and a heatmap that represents the MAP-Elites generated. This graph is interactive, and each cell is given a colour based on the agent's performance (EoG time).

*4) Agent details and gameplay:* Selecting a cell from the map gives details about the corresponding agent. It shows its *behaviour-encoding* (weights assigned to each of its heuristics), the stats resulting from the 100 gameplays of the level, and a pre-recorded video exemplifying its gameplay.

*5) Download files to run the agents locally:* We provide a standalone to run the agents locally. The files and instructions are available to download.

This tool generates interactive maps to retrieve detailed information about each of the agents: resulting stats, performance, and an example of their gameplay. Its visualisation of results allows witnessing the distinct behaviours of the agents. We use it to navigate the maps generated for each game and identify behaviours corresponding to different players and tasks, presented in the following sections.

## IV. EXPERIMENTAL SET-UP

We run a series of experiments to test the approach in games with different characteristics. In this section, we describe the goals identified that correspond to the heuristics implemented for the OLMCTS and the games used. The code is on Github[3].

### A. Goals and Heuristic Implementation

We identify 5 goals that are based on *player-type* goals and inspired by the list of general heuristics presented in [2]. These heuristics gather information related to their goals. This data is used to obtain the stats required to assist during the generation of the *team*. These heuristics are general within the GVGAI Framework and can be used in any of its games.

*1) Winning and score:* It prioritises winning the game while maximising the score. The heuristic is designed to heavily penalise states where the game is lost and to reward those where it is won. It collects winning status (1 for win, 0 for lose), final score, game tick when the score changed last, and game tick when the last positive score change occurred.

*2) Exploration:* It maximises the physical exploration of the map, divided into tiles. The heuristic is designed to take into consideration the number of times each position has been visited. It prioritises visiting those locations that haven't been visited before and, once visited, those that have been visited the least. It favours exploring as much as possible so reaching an EoG state is penalised. It gathers the number of different locations visited, the final exploration matrix with details about the number of visits in each location, and the game tick when the last exploration happened.

*3) Curiosity:* It maximises the discovery and interaction with sprites in the game, prioritising interactions with new sprites or, when this is not possible, in new locations of the game (*curiosity*). Interactions are defined as the avatar or any sprite generated from the player getting in contact with elements of the game (*collisions* and *hits*, respectively). The heuristic is designed to provide a high reward for gameplay that maximizes: a) new sprites discovered and new unique interactions; b) new curiosity interactions; c) total of different curiosity interactions; and d) number of total interactions. It penalises EoG states. It collects the total number of different

[2]https://demo-visualize-diverse-gameplay-xqjmp.ondigitalocean.app

[3]https://github.com/kisenshi/gvgai-agent-behaviour-research

sprites discovered during gameplay, the list of their ids and the game-tick when the last discovery happened. It also gathers data about the number of unique interactions with sprites, number of curiosity interactions, number of total collisions, number of total hits, and the last game-tick when each of these last three interactions occurred.

*4) Killing:* It maximises destroying Non-Player Characters (NPCs). The heuristic is designed to penalise EoG states. 'Kills' are those interactions between sprites generated from the avatar (hits) and sprites of the type NPC. It collects the total number of enemies killed, their sprite id and the game-tick when the last kill happened. As VGDL rules can represent killing enemies in different ways, we make the following assumptions that the game should fulfill for the heuristic to work as expected: a) enemies are killed in one shot; b) enemies are only killed by hits; c) the avatar is not able to kill an enemy by colliding with it or by using elements of the terrain.

*5) Collection:* It maximises collecting items. The heuristic is designed to penalise end of game states. 'Collections' are those interactions between the avatar (collisions) and sprites of the type *Resource*. It gathers the total number of items collected, their sprite id and the game-tick of the last collection. As VGDL rules can represent collecting items in different ways, we make the following assumptions that the game should fulfill for the heuristic to work as expected: a) all collectible items are of the type *Resource*; b) items can only be collected by the avatar by colliding with them.

Not all of the goals apply to every game, as they depend on its characteristics (e.g. there is no point in guiding the agent to collect resources when the game does not include collectibles). In the next section, we present the games and establish which heuristics are included in the experiments for each of them.

### B. Games and Levels

The approach has been applied to four GVGAI games with different characteristics. The levels used in each of them to generate the teams are shown in Fig. 7. We have reviewed the rules of the games and carry out some updates to make sure they fit the assumptions described in the heuristics.

*1) Butterflies:* The goal is to collect all *butterflies* before the time runs out. Butterflies move randomly and *create* other butterflies by interacting with *cocoons* scattered around the map. If all cocoons transform, the game is over. This game does not allow *killing* NPCs or collecting items, so the two corresponding heuristics are not enabled during the experiments. For this game, we run two sets of experiments with two sets of goals to compare the distribution and diversity of the agents generated.

*2) Zelda:* The objective is to collect the *key* and bring it to the *door*. There are *monsters* that can be destroyed by hitting them with a *sword*, but they can also kill the player when colliding with them. All heuristics are used.

*3) Digdug:* The objective is to collect all items (*gems* and *gold*) and kill all *monsters* before the time runs out. The player has a *shovel* that destroys 1) the *walls*, 2) the monsters, and 3) the *blocks of gold* to spawn gold. Monsters can also kill the player by colliding with them. All heuristics are used.

*4) Sheriff:* The objective is to kill all the *bandits* before the time runs out. This game is a new addition to those used in [3]. In contrast to the games listed above, this one is a shooter and the avatar does not have access to the area where the enemies are (the *jail*), as they surround the player. Although there are *barrels* dispersed in the map that protect the player from the enemies' *bullets* and can be destroyed when hit, they are not collectibles. Therefore, we only enable four heuristics.

## V. Experiment I: Team Generation

We test the approach in games with different characteristics by generating a *team* of agents with distinct behaviours for each of them. In this section, we describe the features and values of the MAP-Elites and include an overview of the resulting maps generated. The *jar*, configuration files and resulting JSON data can be found in an OSF repository[4]. The interactive tool described in Section III-C allows a real-time navigation of the results. For simplicity in the tables and results, we give a unique code to each set of experiments carried out for each game, as follows: B2 for *Butterflies* with 2 heuristics enabled: *Winning* and *Exploring*; and B3 when it also includes *Curiosity*. Z5 and D5 for *Zelda* and *Digdug* respectively with all 5 heuristics enabled: *Winning*, *Exploring*, *Curiosity*, *Killing* and *Collection*. Lastly, S4 for *Sheriff* with 4 heuristics enabled: *Winning*, *Exploring*, *Curiosity* and *Killing*.

### A. Experiments Configuration

For an easy experimental set-up, each execution of the algorithm is configurable with an external file; being able to choose the *game*, *level*, *controller*, and following attributes:

*1) nGameRuns:* Number of times the agent (candidate) plays the game with a certain *behaviour-encoding* ($W$) . We use 100 gameplays in every game and experiment.

*2) nRandomInitialisations:* Number of random candidates generated during initialisation, fixed to 10.

*3) nIterations:* Number of iterations of the MAP-Elites. We only have a limited allocated time to run each experiment, so the value set depends on the complexity of the game and the number of heuristics provided. We set 200 iterations for B2, B3, and S4. For Z5 and D5, the time spent on each iteration of the algorithm is higher, so we obtain the *team* after 125 and 100 iterations of the MAP-Elites, respectively.

*4) feature X, feature Y:* We generate 2-dimensional maps, for simplicity in the processing and display of the results. For each game, we execute the MAP-Elites with different pairs of features, which depend on their characteristics. Table I shows the full set of experiments and the pair of features used.

### B. Resulting Maps

Running the experiments results in 68 maps for the five sets of experiments: B2, B3, Z5, D5 and S4. A total of 124, 302, 486, 293 and 352 agents are generated for each game and set of goals, respectively. The group of elites generated in each map forms part of the *team* of available agents to play the game. Going through every map and game is prohibited for the sake of space. The overall observations that stand out when going through the *teams* generated are the following:

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

TABLE I
COMBINATIONS OF FEATURE PAIRS USED FOR EACH SET OF EXPERIMENTS: B2 (10 CONFIGURATIONS), B3 (10 CONFIGURATIONS), Z5 (14 CONFIGURATIONS), D5 (19 CONFIGURATIONS), AND S4 (15 CONFIGURATIONS); FOR A TOTAL OF 68 EXECUTIONS OF MAP-ELITES.

| Y / X | Score | Exploration Percentage | Curiosity | Collisions (B2,B3) Interactions (D5,Z5,S4) | Kills | Items |
|---|---|---|---|---|---|---|
| **Wins** | All | All | All | All | D5 Z5 S4 | D5 |
| **Exploration Percentage** | All | | All | All | D5 Z5 S4 | D5 |
| **Curiosity** | All | | | All | D5 Z5 S4 | D5 |
| **Collisions (B2, B3) Interactions (D5,Z5,S4)** | All | | | | D5 Z5 S4 | D5 |
| **Kills** | S4 | | | | | D5 |

*1) Agents performance:* The performance is given by how fast the agent reaches the End of Game (EoG). Not all the agents from the different *teams* are fast in terms of reaching the EoG, and in some cases, they are very slow with an average of EoG ticks very close to the maximum. However, our approach does not focus on the performance of the agents, and it merely serves as assistance to replace the elites when there is a collision. The interest comes from the diversity of the solutions generated and, thus, the location of the agents in the space of features. It serves as a reference to identify different behaviours. The final performance value of the elite (EoG) gives a hint about its average execution time.

*2) Team size:* The size of the map generated varies both between games and pair of features used. For B2, the number of agents generated for each configuration of the MAP-Elites is found between 4 and 24; for B3, between 16 and 49, for Z5, between 22 and 44, for D5, between 7 and 28; and for S4, between 13 and 44. Overall, the size of the MAP-Elites generated for B2 and D5 is the smallest. For B2, only two of the three possible goals are applied to the agents, resulting in less diversity of final behaviours. *Digdug* is a game comparably more complex than the others. We speculate that in this game, either the MAP-Elites requires more iterations to reach diversity, or it is not possible to elicit further behaviours. As a result, the pool of agents available is smaller.

*3) Agents distribution:* The distribution of the agents in the feature space is different between games for similar pair of features. This disposition provides information about the expected behaviour of the agents based on their location and the values obtained for each feature.

*4) Diverse goals results in a more diverse team:* When comparing the maps generated for B2 and B3, we can infer that the inclusion of the new goal (*Curiosity*) in B3 results in a more diverse team of agents for a similar setup and number of iterations of the MAP-Elites (a total of 124 agents for B2 and 302 for B3). This diversity is clear in the maps generated for the pair of features related to curiosity and interactions. This increased diversity is true even for pair of features not directly related to the new goal, as shown in Fig. 4. Including *Curiosity*, in this case, allows generating agents that obtain different rates of exploration for a higher range of winning rates, providing more diversity and flexibility on the selection of agents. By having more heuristics enabled in a game, it is possible to generate more behaviours and create a more diverse team. For other examples and results discussions, see [3].

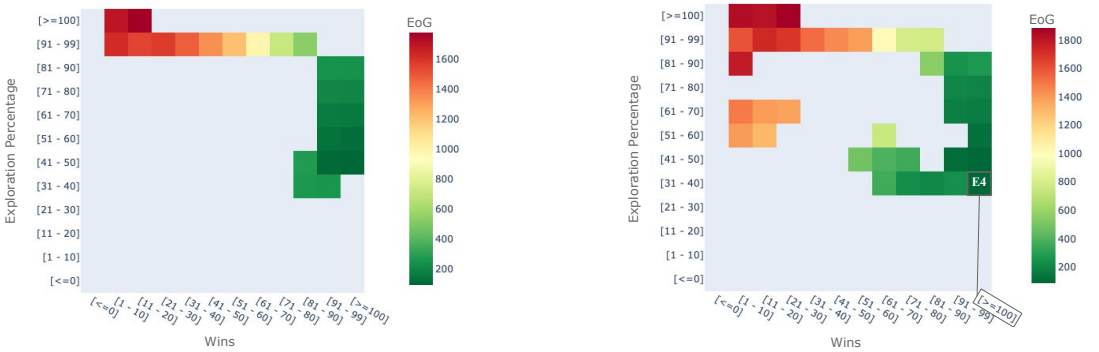
### C. Results Summary

The aim of this experiment is to generate agents in games with different characteristics to prove the generality of our approach. The results provide a *team* of agents, assembled from the group of maps generated, able to play each game automatically. The *team* size per game varies, having a total of 124, 302, 486, 293, and 352 agents with different performances and resulting stats. Each agent of the pool is expected to interact and behave differently in the game. However, not all of them would be practical, so we need to identify the most useful ones based on the game under development. They are located in behavioural spaces based on the outcomes of their gameplay, so this distribution allows identifying agents with target abilities based on their map location. The following section describes such identification by selecting 6 agents of different characteristics for each game.

## VI. BEHAVIOUR-TYPE AGENTS IDENTIFICATION

We identify 6 agents for each game that correspond to behaviours and tasks we are interested in. These *behaviour-types* agents have been selected from the resulting maps that assemble the *team* for each game (Section V-B). The descriptions of the different *behaviour-type* agents identified across the games, in alphabetical order, are the following:

- *Barrels Shooter*. It only applies to *Sheriff*. Agent with a high rate of *interactions* and hits but a low rate of *kills*, so it targets the barrels instead of the bandits. It is identified in the map with the corresponding features.
- *Collector (High/Low)*. Agent that gathers a high or low number of *items*.
- *Curiosity (High/Low)*.
- *Explorer (High/Low)*.
- *Interactions (High/Low)*.
- *Killer (High/Low)*.
- *Scorer (High/Low)*.
- *Speed-runner*. Agent with a high victory rate that tends to finish the game fast. It is identified by looking through the agents with a very high win rate in the maps with such a feature to find one with one of the lowest EoG.
- *Survivor* It only applies to *Sheriff*. Agent that survives until the time runs out, winning the game. It is identified in the map with the corresponding feature by looking at a resulting high EoG (performance).
- *Walls Breaker*. It only applies to *Digdug*. Agent that focuses on breaking the walls and tends to follow them instead of moving through open areas. It is identified by having a high curiosity but a mid-range exploration rate in the map with the corresponding features.
- *Walls Interaction*. It only applies to *Butterflies*. Agent that focuses on interacting with the trees and butterflies. It is expected to move close to the walls bypassing open areas. It is identified by having high curiosity but average




(a) *B2*

(b) *B3. Includes agent identification: Speed-runner (E4), with* $\simeq 100\%$ *win rate and one of the lowest EoG ticks.*

Fig. 4. Resulting MAP-Elites for features *Wins* and *Exploration Percentage*.

exploration features in the map with the corresponding features, and confirmed by observing its gameplay.

*Curiosity*, *Explorer*, *Interactions*, *Killer*, and *Scorer* are identified from the maps with the corresponding feature. The group of agents chosen is different between games. All the agents selected for *Butterflies* are taken from the results of B3, as we have determined that the diversity of behaviour in the team is richer than the one resulting for B2. For each game, we include one of the MAP-Elites where at least one of the agents is found (Figs. 4b, 5a, 5b, and 5c). Table II shows an overview of all the agents selected for each game.

## VII. Experiment II: Team Portability

We carry out an experiment in each game to determine the portability of the *behaviour-type* agents identified. We propose that if an agent presents similar stats or a similar tendency across different levels, it suggests that their strength is carried between them. Therefore, that agent can be used with similar expectations independently of the level used to generate it.

### A. Description of the New Levels

We present four new levels for each game, with a similar size but different characteristics and distribution of the elements. The outline of these levels is shown in Fig. 7. We run each of the agents identified a total of 100 times in each of the levels to obtain the stats and compare them to their corresponding features. We also repeat the gameplay in the original level and obtain stats from it as guidance.

*1) Butterflies:* The levels have a different number of butterflies, trees and cocoons, presenting different distributions of elements and area shapes; some of them being more open (7e, 7m) than the original level. Level 7i is horizontally symmetric. Level 7q is designed to have player, butterflies and cocoons in differentiated areas, with only one access point between them. Its design is very different from the other levels, which causes a small noise in the stats collected.

*2) Zelda):* The levels maintain the number of monsters (except 7n where we include two additional ones), so the differences between them come from their shape given by the distribution of walls. Levels 7f and 7n are open, the latter with blocks in the middle, surrounding the key. Levels 7j and 7r have many walls. The former has narrow corridors and the latter large rooms, connected by one access point.

*3) Digdug:* All elements provide differences between the levels for this game. Level 7g contains breakable walls covering most of the level and three spawners, but few items. Level 7k contains a high density of walls but with an open area in the middle that separates them. Level 7o has a heterogeneous distribution of walls and open spaces, with few monsters and items. Level 7s is the most different one, as the walls surround the level and separate two big open areas. One of these areas is where the player starts the game, while the other contains a high number of items and monsters.

*4) Sheriff:* The shape of the game is similar on all levels (given by the rules and the mechanics of the game), and the number of bandits varies between 9 and 11. The main difference between levels come from the number and distribution of barrels. In levels 7h and 7l, the barrels are distributed around the borders. On the contrary, in levels 7p and 7t, there are barrels also located in the middle of the map.

### B. Results: Portability of the Agents Between Levels

The experiments result in stats from 100 gameplays of each agent in each level. We present the results in Tables III, IV, V and VI showing, for each of the agents, the stats applicable to their *behaviour-type*. We also include the relevant information about each level for context.

*1) Butterflies:* We see a trend on the stats confirming that most of the agents are portable between levels, and it is quite clear for the agents related to exploration (E5, E6), curiosity (E3, E6) and win rates (E4). In the latter, we can highlight the fact that the win rate and exploration drops in level 7q compared to the other ones. However, looking at the win rate stats overall the agents, E4 has the highest percentage of victories (89%), compared to the rest 85% (E1), 9% (E2), 4% (E3), 8% (E5), and 5% (E6). E4 also shows the lowest EoG value (61.13), vs 78.68, 959.46, 862.94, 793.78, and 860.37. Similarly, E5 achieves one of the highest exploration rates (74.56%), vs 32.25% (E1), 78.05% (E2), 42.82% (E3), 25.60% (E4), and 43.25% (E6). Therefore, the strength of these agents (winning the game quickly and achieving a high

TABLE II

FULL SELECTION OF AGENTS FROM THE TEAM PER GAME, IDENTIFIED BASED ON THEIR BEHAVIOUR. FOR EACH *behaviour-type* AGENT, IT INCLUDES THE FEATURE PAIRS OF THE MAP-ELITES WHERE IT WAS FOUND AND THE RESULTING AVERAGE OF STATS (*Map features*), THE CORRESPONDING CELL IN SUCH MAP (*Cell id*), END OF GAME TICKS (*EoG*), AND THE WEIGHTS THAT ENCODE THE HEURISTIC ASSIGNED TO IT (*Weights W*). THE WEIGHTS CORRESPOND TO THE HEURISTICS IN THE FOLLOWING ORDER. B3: *Winning and Score*, *Exploration*, *Curiosity*; S4: *Winning and Score*, *Exploration*, *Curiosity*, *Killing*; Z5, D5: *Winning and Score*, *Exploration*, *Curiosity*, *Killing*, *Collection*.

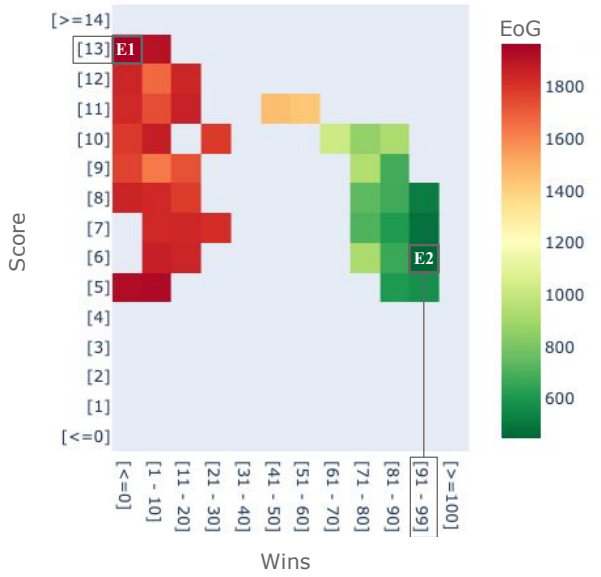
| | **E1** | **E2** | **E3** | **E4** | **E5** | **E6** |
|---|---|---|---|---|---|---|
| **Butterflies (B3)** | | | | | | |
| **Behaviour-type** | ***Low scorer*** | ***High scorer*** | ***High curiosity*** | ***Speed-runner*** | ***High explorer*** | ***Walls interaction*** |
| **Map features** | *Collisions*: 8.09<br>*Score*: 19.18 | *Collisions*: 370.36<br>*Score*: 50.72 | *Curiosity*: 121.92<br>*Collisions*: 1398.23 | *Wins*: 100%<br>*Exploration*: 37.72% | *Exploration*: 99.92%<br>*Collisions*: 540.19 | *Exploration*: 65.63%<br>*Curiosity*: 108.15 |
| **Cell id** | $[1, 2]$ | $[8, 6]$ | $[7, 21]$ | $[11, 4]$ | $[11, 11]$ | $[7, 6]$ |
| **EoG** | 95.69 | 1860.99 | 1675.32 | 87.93 | 1939.34 | 1479.75 |
| **Weights** $W$ | $0.92, 0.52, 0$ | $0.1, 0.45, 0.67$ | $0, 0, 0.21$ | $0.94, 0.08, 0.02$ | $0.02, 0.29, 0.68$ | $0.04, 0, 0.9$ |
| **Zelda (Z5)** | | | | | | |
| **Behaviour-type** | ***High scorer*** | ***Speed-runner*** | ***High explorer*** | ***Low killer*** | ***High killer + high explorer*** | ***High killer + low explorer*** |
| **Map features** | *Wins*: 0.00%<br>*Score*: 12.85 | *Wins*: 99.00%<br>*Score*: 6.37 | *Exploration*: 97.00%<br>*Curiosity*: 62.66 | *Interactions*: 37.91<br>*Kills*: 1.95 | *Exploration*: 95.64%<br>*Kills*: 5.50 | *Exploration*: 46.92%<br>*Kills*: 4.64 |
| **Cell id** | $[0, 13]$ | $[10, 6]$ | $[20, 7]$ | $[1, 2]$ | $[20, 6]$ | $[10, 5]$ |
| **EoG** | 1966.42 | 450.05 | 1902.58 | 544.32 | 1849.54 | 1951.38 |
| **Weights** $W$ | $0, 0.62, 0, 0.59, 0$ | $0.96, 0.54, 0.61, 0, 0$ | $0.92, 0.61, 0.01, 0.93, 0.36$ | $0.66, 0.13, 0.01, 0.04, 0.16$ | $0.08, 0.55, 0.04, 0.84, 0.51$ | $0, 0, 0, 1, 0$ |
| **Digdug (D5)** | | | | | | |
| **Behaviour-type** | ***High collector + high killer*** | ***High collector + low killer*** | ***Low collector + high killer*** | ***Walls breaker*** | ***High explorer*** | ***Low explorer + high scorer*** |
| **Map features** | *Kills*: 10.55<br>*Items*: 26.90 | *Kills*: 2.84<br>*Items*: 26.98 | *Kills*: 9.91<br>*Items*: 9.83 | *Exploration*: 67.14%<br>*Curiosity*: 450.88 | *Exploration*: 100.00%<br>*Curiosity*: 324.97 | *Exploration*: 29.77%<br>*Score*: 35.77 |
| **Cell id** | $[4, 10]$ | $[1, 10]$ | $[4, 4]$ | $[14, 19]$ | $[21, 13]$ | $[6, 8]$ |
| **EoG** | 1995.89 | 1989.92 | 1999.99 | 1783.33 | 1999.01 | 1999.93 |
| **Weights** $W$ | $0, 0, 0.7, 0.92, 1$ | $0, 1, 0, 0, 1$ | $0, 0, 0, 1, 0$ | $0, 0, 0.69, 0, 0.01$ | $0, 0.7, 0.86, 0, 0.01$ | $0, 0, 0, 0.49, 0.33$ |
| **Sheriff (S4)** | | | | | | |
| **Behaviour-type** | ***Survivor + low killer*** | ***High killer + high explorer*** | ***High killer + low explorer*** | ***Speed-runner*** | ***Barrels shooter*** | ***High curiosity + low interactions*** |
| **Map features** | *Wins*: 100%<br>*Kills*: 3.19 | *Exploration*: 99.50%<br>*Kills*: 7.00 | *Exploration*: 14.04%<br>*Kills*: 6.99 | *Wins*: 100.00%<br>*Score*: 7.99 | *Interactions*: 350.80<br>*Kills*: 5.03 | *Curiosity*: 101.29<br>*Interactions*: 127.83 |
| **Cell id** | $[11, 3]$ | $[11, 7]$ | $[2, 7]$ | $[11, 9]$ | $[8, 5]$ | $[6, 3]$ |
| **EoG** | 999.00 | 966.77 | 412.30 | 359.17 | 549.71 | 819.76 |
| **Weights** $W$ | $0.84, 1, 0, 0$ | $0, 0.26, 0, 0.46$ | $0.09, 0, 0, 1$ | $0.57, 0.05, 0.03, 0.92$ | $0, 0, 0.19, 0$ | $0.28, 0.25, 0.9, 0.46$ |

exploration) is still transferred to this level 7q, even when such portability is not noticeable at first. This level is also the most different one, as there are two distinct areas with a unique access points between them. Thus, the agents need to find that access to be able to collect the butterflies and win. There are only 3 cocoons, and the butterflies are close to these, so the game may end before the agent can even reach the area to win the game or explore it. Regarding the agents related to score, either high (E1) or low (E2), the resulting stats implies that they are not as easily transferable between levels as the other agents. In *Butterflies*, the score depends on the disposition of the butterflies on the map and their distance to the player.

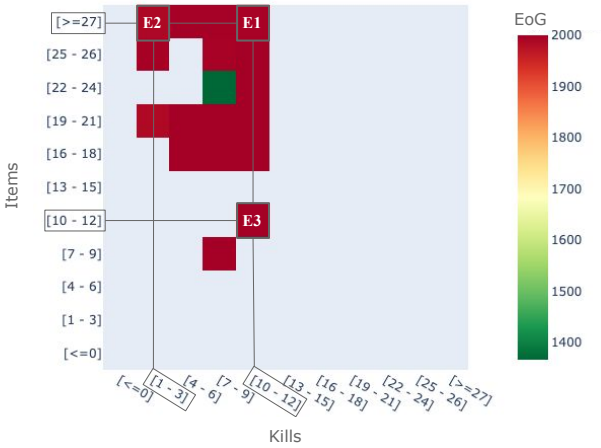
*2) Zelda:* All agents selected for this game seem to be transferable between levels. For the *High scorer* (E1), the final score is similar through all levels, increasing accordingly to the additional monsters in level 7n. We believe these results come from the characteristics of the game and the fact that we use a similar number of the elements involved in its calculation. The number of kills increase and decrease relatively to the characteristics of the agents related to that feature: E5 and E6 achieve a high number of kills across levels, while E4, on the other hand, obtains a comparably low number. We see similar results for exploration, where agents expected to obtain a high rate (E3 and E5) achieve more than $93.33\%$ in every level, while the low explorer (E6) achieves $54.79\%$. Taking a look at the *Speed-runner* (E2), it is not clear the tendency is maintained through the levels. However, when taking a look at the overall win rates and EoG ticks for the rest of the agents, we can assert that it actually does. Although the win rate is slightly higher for E4 in level 7r, the resulting rates for the other four agents is significantly lower: $0\%$ (E1), $8\%$ (E3), $12\%$ (E5), and $0\%$ (E6). Therefore, $96\%$ is a high value in comparison. On the rest of the levels, all the rest of agents obtain lower win rate than E2: $0 - 73\%$ vs $76\%$ in level 7f, $0 - 60\%$ vs $65\%$ in level 7j, and $1 - 79\%$ vs $89\%$ in level 7n. We observe similar results for the EoG ticks, as E2 obtain comparatively lower EoG ticks across levels than the other agents, even when these are objectively high in levels 7f and 7j.

*3) Digdug:* All agents selected seem to be transferable between levels in this game. Those expected to collect a high number of items (E1, E2) manage to do so, achieving an average close to the maximum number of resources available. On the other hand, the *Low collector* (E3) gathers at most a third of those resources. The stats of these agents also match with the corresponding number of kills expected from each of them. While E1 and E3 achieved a high number of kills, E2 does not. Therefore, these three agents with different proficiency in collecting items and killing enemies are transferable, and the trend of their results is maintained between the different levels. We observe a similar tendency in the agents with stats related to exploration: E4 as a *Wall breaker* is expected to achieve a mid-average exploration percentage ($<= 70.12\%$) and high curiosity, and it gets similar stats in the new levels. The *High explorer* (E5) achieves between $98.51\%$ and $100\%$ exploration across levels. E6, on the other hand, reaches a maximum of $37.98\%$ while achieving a high score, as expected.

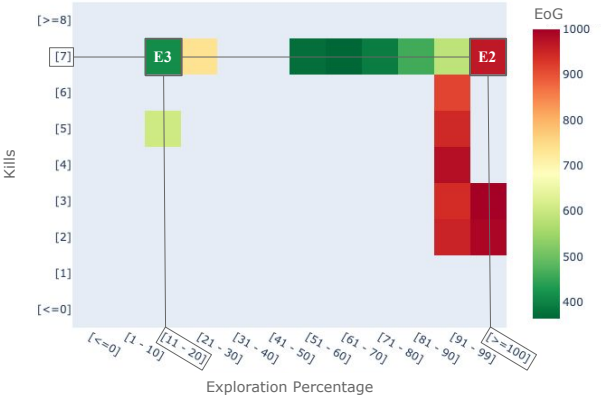
*4) Sheriff:* All agents display the characteristics they were selected for through the different levels. E1 and E4 are agents




(a) *Zelda*: *High scorer* (E1), with the highest score feature across the maps, and *Speed-runner* (E2), with one of the highest win rate and lowest EoG.




(b) *Digdug*: *High collector and killer* (E1), with a high number of items collected and enemies killed, *High collector and low killer* (E2), with a high number of items collected but a low number of enemies killed, and *Low collector and high killer* (E3), with a low number of items collected but a high number of enemies killed.




(c) *Sheriff*: *High killer and explorer* (E2), with the highest value pair of exploration rate and average of kills, and *High killer and low explorer* (E3), with the highest number of kills and lowest exploration rate.

Fig. 5. *Behavior-type* agents identification from one of the resulting MAP-Elites for the games *Zelda*, *Digdug*, and *Sheriff*.

related to winning the game but in different ways. E1 is a *Survivor* that wins the game by reaching the end of the game while avoiding killing many enemies. Its behaviour is shown on the stats overall levels: $>= 98\%$ win rate, less than half of the bandits kill in average and EoG close to the maximum. E4, on the contrary, achieves a low average of EoG ticks across levels ($<= 418.7$) while still achieving a high rate of wins ($>= 97\%$). E2 and E3 are both *High killers* (achieving similar killing stats through the levels) with different skills on exploration: E2 is a *High explorer* and reaches a high exploration overall levels ($>= 97.89\%$), while E3's exploration is low ($<= 17.81\%$). Similarly, stats for E5 and E6 follow a similar trend across the levels.

### C. Results Summary

The generality of the heuristics allows running the *behaviour-type* agents at any level of the game. This experiment aimed to show if the proficiency identified in each of those agents is portable to new levels. Results show that the tendency of the resulting gameplay stats related to their *behaviour-type* is generally maintained between levels. In most cases, this portability is straightforward to confirm by just looking at the results of the agent across levels. However, there are cases where it is also necessary to consider the global stats between agents at a particular level to reach this conclusion. It is also possible that the characteristics of a game impede a particular *behaviour-type* agent from being portable. For example, the score in *Butterflies* is very dependent on the distribution of the elements at the start of the game, so the agents related to it, E1 and E2, are not capable of carrying out their proficiency when they play new ones.

Thus, a limitation of this approach materializes when features relevant to the agent are very dependent on the rules of the game. Running similar portability experiments after identifying the agents to ensure they can generalise to new levels should mitigate this constraint. Portable agents could be used to test new or updated levels to help find potential design flaws. These design flaws could be detected by highlighting those cases where the stats obtained by the agents do not fit the expectations. That is the goal pursued in the next section.

## VIII. Using the *Team* for Automated Testing

This section exemplifies using *behaviour-type* agents for testing the design and validity of new levels. We modify the original levels of the games used to generate the *team* to be impossible to win, removing crucial elements or increasing difficulty (Fig. 7). We hypothesise that these modifications will have an effect in the tendency of the gameplay stats achieved by the *behaviour-type* agents. Each of the agents play the level 100 times. Tables III, IV, V, and VI compare, per game, the resulting stats for each *behaviour-type* agent from playing the new level to the ones obtained in the portability experiments.

*1) Butterflies:* The new level is similar to the original, but all the butterflies are cocoons instead. An isolated butterfly is included so the game does not end immediately. This butterfly does not have access to the area where the player is. Therefore, the player cannot win the game, and the time always runs out.

The resulting score for both high and low *Scorers* (E1 and E2) is 0. We have previously discerned that these agents may not be portable between levels, but these results are intriguing. Furthermore, in this level, *Speed-runner* (E4) never wins the game and averages an EoG ticks similar to the maximum allowed $(2,000)$. Looking at these results, we can infer that the level is not winnable, and that there are no butterflies at reach to the player because the score does not increase. The resulting stats of E3, E5, and E6 do not raise any areas of concern. These agents still achieve the expected high curiosity, exploration, and wall interaction, respectively, as the modifications in the level do not impede fulfilling their tasks.

TABLE III
*Butterflies*: PORTABILITY AND EXPERIMENTAL LEVEL TESTING COMPARISON. COLUMNS SHOW AVERAGE GAMEPLAY STATS PER LEVEL. THE STATS SHOWN ARE THE ONES RELATED TO THEIR PROFICIENCY. THE DIMENSIONS HIGHLIGHTED ARE DIRECTLY OR INDIRECTLY IMPACTED BY CHANGES IN THE 'BROKEN' LEVEL. EOG IS 2000.

| | **7a** | **7e** | **7i** | **7m** | **7q** | **'Broken'** |
|---|---|---|---|---|---|---|
| Max. score | 64 | 38 | 58 | 38 | 18 | 64 |
| Total walls | 102 | 93 | 91 | 83 | 99 | 102 |
| **Low scorer (E1)** | | | | | | |
| *Score* | 20.44 | 29.58 | 43.56 | 21.2 | 12.64 | **0** |
| **High scorer (E2)** | | | | | | |
| *Score* | 47.76 | 30.78 | 51.1 | 22.58 | 11.86 | **0** |
| **High curiosity (E3)** | | | | | | |
| *Curiosity* | 116.53 | 98.99 | 116.38 | 82.13 | 76.38 | 103.74 |
| *Collisions* | 1436.82 | 1754.85 | 1362.33 | 1780.46 | 715.56 | 1858.56 |
| **Speed-runner (E4)** | | | | | | |
| *Win rate* | 100% | 100% | 99% | 100% | 89% | **0%** |
| *EoG* | 102.15 | 103.49 | 114.38 | 73.61 | 61.13 | **2000** |
| **High explorer (E5)** | | | | | | |
| *Exploration* | 99.29% | 99.30% | 99.33% | 99.77% | 74.56% | 100% |
| **Walls interaction (E6)** | | | | | | |
| *Exploration* | 67.43% | 60.28% | 68.15% | 44.6% | 43.25% | 59.40% |
| *Curiosity* | 114.59 | 96.53 | 120.46 | 79.86 | 75.16 | 103.72 |

*2) Zelda:* The new level is similar to the original, but an area of the map is inaccessible and contains one of the monsters and the key. Thus, not all locations are reachable and the player cannot collect the key to win. The agents whose gameplay results suffer significant changes are E2, E3, and E5. The win rate for the *Speed-runner* (E2) drops to $0\%$, while the EoG ticks increase to $1897.65$. This value is higher than any result previously obtained by this agent ($<= 1242.26$). These results suggest the impossibility of winning the game. In addition, the exploration rate for the *High explorer* agents (E3 and E5) drops from higher than $90\%$ to lower than $70\%$. These results hint that part of the level is not accessible. Coincidentally, all these agents relate to either winning or achieving a high exploration, so they are directly impacted by the issues in the level. There is also a difference in the results when the agents are indirectly affected. E4 (*Low killer*) achieves a higher average of kills than in the previous cases $(4.06)$. The number of enemies does not increase, but not being able to finish the game provides more time for this agent to kill enemies. Similarly, the final score of E1 drops to $9.72$ because it is not possible to get the key, win, or kill the enemy that is out of reach. Stats related to E6 (*High killer and low explorer*) are not impacted. Fig. 6 shows a graph with the *Exploration* stats of the $100$ play-throughs for E3, E5, and E6 in the 'broken' level compared to the portability experiment.

TABLE IV
*Zelda*: PORTABILITY AND LEVEL TESTING RESULTS. EOG IS 2000.

| | **7b** | **7f** | **7j** | **7n** | **7r** | **'Broken'** |
|---|---|---|---|---|---|---|
| *Max. score* | 14 | 14 | 14 | 18 | 14 | 14 |
| *Total enemies* | 6 | 6 | 6 | 8 | 6 | 6 |
| **High scorer (E1)** | | | | | | |
| *Score* | 12.25 | 12.11 | 11.7 | 16.44 | 11.55 | **9.72** |
| **Speed-runner (E2)** | | | | | | |
| *Win rate* | 90% | 76% | 65% | 89% | 96% | **0%** |
| *EoG* | 617.18 | 1187.98 | 1242.26 | 689.03 | 450.86 | **1897.65** |
| **High explorer (E3)** | | | | | | |
| *Exploration* | 98.03% | 95.36% | 95.25% | 98.31% | 93.33% | **68.25%** |
| **Low killer (E4)** | | | | | | |
| *Kills* | 1.62 | 3.25 | 2.97 | 3.53 | 1.79 | **4.06** |
| **High killer and explorer (E5)** | | | | | | |
| *Kills* | 5.68 | 5.66 | 5.44 | 7.71 | 5.41 | 4.63 |
| *Exploration* | 97.19% | 97.93% | 94.71% | 98.61% | 94% | **66.47%** |
| **High killer and low explorer (E6)** | | | | | | |
| *Kills* | 4.78 | 5.17 | 4.21 | 7.26 | 3.77 | 4.48 |
| *Exploration* | 48.38% | 50.66% | 42.25% | 54.79% | 35.73% | 44.28% |

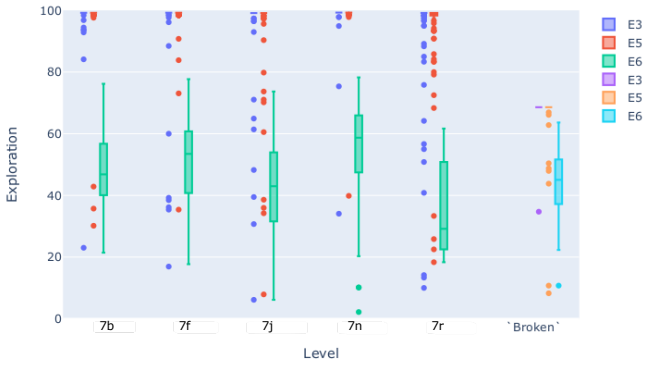



Fig. 6. Experimental level testing in *Zelda*. Resulting *Exploration* in the 'broken' level compared to the portability results, from 100 gameplays. E3: *High explorer*, E5: *High killer and explorer*, E6: *High killer and low explorer*

*3) Digdug:* The new level is similar to the original, but all the breakable walls have been removed. These are crucial game elements, so we expect that removing them affects playability. In the results, neither the collectibles, kills, or high exploration features hint that there may be an issue, as these stats maintain the expected trend. The tweaked level has the walls removed, so the agent expected to break them (E4) should be the one directly impacted by the change. This hypothesis is confirmed by looking at its results: curiosity and hits drop drastically, from $464.6$ and $269.65$, respectively, on the original level to $47.58$ and $18.31$ on the new one. The exploration of this agent also is impacted and drops to $45.53\%$ because it is likely that this agent makes use of its ability to break walls to cover the map. On the other hand, the exploration for the *Low explorer* (E6) increases (slightly) to $44.32\%$, while in the rest of the levels it is between $30-37\%$. By analysing this agent on its own, this result could be taken as an outlier. However, when analysed in conjunction with the rest, we can assume that this change could also be linked to the nonexistence of walls. The agent is indirectly impacted because it does not use actions or game-ticks to break the walls, being able to move instead, covering more space.

*4) Sheriff:* The shape and distribution of the barrels in the new level are similar to the original. However, we have

TABLE V
*Digdug*: PORTABILITY AND LEVEL TESTING RESULTS. EOG IS 2000.

| | 7c | 7g | 7k | 7o | 7s | 'Broken' |
|---|---|---|---|---|---|---|
| *Max. score* | 44 | 48 | 61 | 14 | 53 | 44 |
| *Total enemies* | 12 | 16 | 13 | 5 | 14 | 12 |
| *Total items* | 27 | 19 | 41 | 8 | 50 | 27 |
| *Breakable walls* | 267 | 292 | 247 | 196 | 103 | 0 |
| **High collector and killer (E1)** | | | | | | |
| *Kills* | 10.64 | 13.74 | 11.56 | 3.88 | 12.68 | 10.34 |
| *Items* | 26.92 | 18.91 | 40.76 | 7.95 | 49.90 | 26.01 |
| **High collector and low killer (E2)** | | | | | | |
| *Kills* | 2.81 | 3.19 | 3.05 | 0.60 | 4.69 | 2.61 |
| *Items* | 27 | 19 | 40.99 | 8 | 50 | 27 |
| **Low collector and high killer (E3)** | | | | | | |
| *Kills* | 9.64 | 12.62 | 12.53 | 4.7 | 13.31 | 11.54 |
| *Items* | 9.14 | 5.05 | 11.36 | 1.72 | 9.74 | 10.33 |
| **Walls breaker (E4)** | | | | | | |
| *Exploration* | 70.12% | 67.66% | 67.70% | 64.73% | 46.98% | **45.53%** |
| *Curiosity* | 464.6 | 463.77 | 434.05 | 358.54 | 249.87 | **47.58** |
| *Hits* | 269.65 | 282.32 | 245.53 | 197.70 | 120.67 | **18.31** |
| **High explorer (E5)** | | | | | | |
| *Exploration* | 99.98% | 100% | 99.99% | 100 | 98.51% | 100% |
| **Low explorer and high scorer (E6)** | | | | | | |
| *Exploration* | 30.92% | 26.49% | 36.32% | 26.96% | 37.98% | **44.32%** |
| *Score* | 35.49 | 35.46 | 51.3 | 11.02 | 47.71 | 40.26 |

TABLE VI
*Sheriff*: PORTABILITY AND LEVEL TESTING RESULTS. EOG IS 1000.

| | 7d | 7h | 7l | 7p | 7t | 'Broken' |
|---|---|---|---|---|---|---|
| *Total enemies* | 8 | 9 | 9 | 11 | 11 | 35 |
| *Total barrels* | 55 | 65 | 27 | 71 | 51 | 55 |
| **Survivor and low killer (E1)** | | | | | | |
| *Win rate* | 98% | 99% | 100% | 99% | 100% | 96% |
| *Kills* | 3.28 | 4.02 | 3.89 | 4.78 | 4.87 | 16.51 |
| *EoG* | 992.58 | 990.32 | 1000 | 991.82 | 1000 | 971.60 |
| **High killer and explorer (E2)** | | | | | | |
| *Exploration* | 97.89% | 98.97% | 99.01% | 98.2% | 99.55% | 94.86% |
| *Kills* | 6.96 | 7.95 | 7.96 | 9.91 | 10 | 32.73 |
| **High killer and low explorer (E3)** | | | | | | |
| *Exploration* | 15.33% | 16.36% | 15.26% | 17.81% | 15.96% | 20.61% |
| *Kills* | 6.95 | 7.93 | 8 | 9.95 | 10 | 32.11 |
| **Speed-runner (E4)** | | | | | | |
| *Win rate* | 100% | 97% | 98% | 99% | 100% | 95% |
| *EoG* | 364.75 | 360.53 | 363.59 | 417.71 | 418.7 | **650.20** |
| **Barrels shooter (E5)** | | | | | | |
| *Interactions* | 439.42 | 372.63 | 344.74 | 385.54 | 310.89 | **70.21** |
| *Kills* | 5.83 | 5.89 | 6.22 | 7.46 | 6.65 | 9.18 |
| *Hits* | 35.96 | 38.19 | 17.49 | 39.86 | 29.28 | 20.86 |
| **High curiosity and low interactions (E6)** | | | | | | |
| *Curiosity* | 95.77 | 101.81 | 73.46 | 99.92 | 89.47 | 75.2 |
| *Interactions* | 122.52 | 127.46 | 106.28 | 125.93 | 116.98 | 118.84 |

increased the number of bandits from 8 to 35. We hypothesise that almost quadrupling the enemy count would increase the difficulty drastically, being unmanageable for some players. The resulting gameplay stats of the *Barrel shooter* (E5) are the ones impacted the most, as the number of interactions drops from 439.42 to 70.21. This agent dies quickly (149.38 EoG) and often (1% win rate), not having enough time to destroy the barrels or interact with them. We believe that the main cause of this change is the indifference of the agent to killing enemies and, given the high number of these, it is unable to dodge them. The EoG doubles for the *Speed-runner* (E4) from 364.75 to 650.2, taking it longer to kill all enemies and win the game. It also decreases slightly for the *Survivor* (E1), from 992.57 to 971.6. The win rate decreases only slightly in both cases (95% vs 100%, and 96% vs 98%, respectively). Despite the drastic increase in the number of enemies, both agents can still win the game most of the time. There is a slight change in the trend of the stats for the rest of agents, but these are not radically different when compared to the original level.

### A. Results Summary

We observe a discrepancy in the resulting stats in agents that have a direct or indirect a connection to the features affected by the problem identified in the level. For example, in *Zelda*, the *Speed-runner* (E2) and *High explorer* agents (E3, E5) are directly affected by part of the level not being accessible, making impossible to win the game. As a result, they suffer a drastic change in the stats (Fig. 6). Similarly, the *High scorer* (E1) and *Low killer* (E4) are indirectly affected by the issues in the level, which also causes a change in their stats. Unrelated agents still achieve similar results that fit the expectations. Therefore, having a *team* of different agents help identify issues in different aspects of the level.

## IX. CONCLUSIONS AND FUTURE WORK

We present and implement an approach to generate a *team* of agents with differentiated behaviours and identifiable tasks that can be used to play the game automatically and, ultimately, assist in the development and testing of games and levels. In contrast to other solutions, the difference between agents come from the combination of heuristics focused on distinct goals related to play-styles: winning, exploring the game, interacting with their elements, killing enemies or collecting items; instead of updates on internal parameters of the controller. The core of the agents do not change, and the goals are provided externally, so these can be easily interchangeable while maintaining the foundation of the algorithm. In addition, we are interested in the results of the agents after playing the game, instead of looking at replicating real players. We look at the resulting stats to identify different *behaviour-types*, who we do not expect to behave as human players would. We design and implement a new *parent* heuristic that grants the proposed diversity by the assignation of *weights* (*encoding-behaviour*) to a list of goals. We apply the MAP-Elites algorithm for the generation of the *team*, by randomly generating and evolving these *weights*. It results in agents distributed in a 2-dimensional behavioural space from where we can identify different *behaviour-types* based on their location. Each of these *behaviour-type* agents would be expected to play the game reaching particular outcomes.

We use the GVGAI Framework, so the heuristics implemented are general within its supported games. This generality allows to run experiments and demonstrate the use of the approach in four games with different characteristics. We differentiate between five sets of experiments, corresponding to the games and goals enabled: B2, B3, Z5, D5 and S4. We run the experiments with different pairs of features, generating 68 maps that result in a total of 124, 302, 486, 293 and 352 agents for each set, respectively. We present an interactive tool that allows exploring the results and checking the details of each of these agents. We include an experiment to test the portability of the agents that would allow running them on newly created levels or after modifications have been made on

existing ones. For this experiment, we include 4 new levels to each game with different characteristics. The final stats obtained after playing each of the levels follow a similar tendency for most of the agents, supporting the hypothesis that virtually all the agents identified are portable between levels because they are able to carry their strengths to new ones. Therefore, they could be used to highlight issues when the resulting stats do not match the expectations. We include a preliminary work exploring this idea and use the *behaviour-type* agents to test a 'broken' level in each of the games. The agents whose behaviour is directly or indirectly related to the issue in the level suffer a variation in their resulting stats, breaking the tendency found in the portability experiments. On the contrary, those unrelated to the feature affected still fulfill the expectations. We conclude that having a *team* with a range of different behaviours related to different features in the game should allow covering different type of cases and issues. Consequently, the *team* of agents proposed could be used for testing and enriched by defining and implementing a further selection of heuristics that enable further behaviour combinations for agent selection.

An important limitation to our approach is that, even when we have shown that it is possible for these agents to operate properly when there is changes in the levels, modifications in the core dynamics of the game would require the heuristics to be adjusted. Similarly, if new goals come up as part of the iterative design process, new heuristics will need to be defined. Furthermore, the approach defined to generate the *team* of agents has been implemented with a search algorithm in a general environment. Not every GVGP approach makes use of a forward model, so integrating the methodology proposed into a system without a forward model would require further investigation. First, it would be needed to define and provide a new diversification of behaviours adapted to its characteristics. Once the *heuristic diversification* is defined for the new scope, a similar approach followed for the generation of the agents using the MAP-Elites could be followed, as it simply requires a vector of weights that encode the behaviour. Therefore, this approach could be implemented using methods that use Reinforcement Learning and Quality-Diversity, such as [28].

The current approach has shown good performance in simple GVGAI levels, but agents would need to be able to play at a higher skill level to tackle more complex games, which may require agents to deal with long-term planning, partial observability, stochasticity and/or multiple players, among others. However, the core contribution of this paper still stands; a similar approach could be extended to more complex games (3D), other types of games not based on avatars (e.g. strategy games), or use multi-dimensional feature spaces. It should also be possible to look beyond the manual design of levels of a game and integrate the approach with Procedural Content Generation (PCG) techniques to assist in their automated generation. We believe that following the line of research started by our work to fit different areas and types of GVGP agents would allow adopting the approach in a more diverse range of games and frameworks, ultimately being applicable to the games industry and assisting on game development and testing processes.

## 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 Conference on Computational Intelligence and Games (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 Conference on Computational Intelligence and Games*, 2018, pp. 1–8.

[3] C. Guerrero-Romero and D. Perez-Liebana, "MAP-Elites to Generate a Team of Agents that Elicits Diverse Automated Gameplay," in *IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–8.

[4] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[5] J. van Valburg, "Automated Testing and Profiling for Call of Duty," 2018, GDC. [Online]. Available: {https://www.youtube.com/watch?v=8d0wzyiikXM}

[6] R. Masella, "Automated Testing of Gameplay Features in 'Sea of Thieves'," 2019, GDC. [Online]. Available: {https://www.youtube.com/watch?v=X673tOi8pU8}

[7] S. Ariyurek, A. Betin-Can, and E. Surer, "Automated video game testing using synthetic and humanlike agents," *IEEE Transactions on Games*, vol. 13, no. 1, pp. 50–67, 2019.

[8] I. Zarembo, "Analysis of Artificial Intelligence Applications for Automated Testing of Video Games," in *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, vol. 2, 2019, pp. 170–174.

[9] A. M. Albaghajati and M. A. K. Ahmed, "Video Game Automated Testing Approaches: An Assessment Framework," *IEEE Transactions on Games (Early Access)*, pp. 1–15, 2020.

[10] R. Bartle, "Hearts, Clubs, Diamonds, Spades: Players who Suit MUDs," *Journal of MUD research*, vol. 1, no. 1, p. 19, 1996.

[11] N. Yee, "The Gamer Motivation Profile: What we Learned from 250,000 Gamers," in *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, 2016, pp. 2–2.

[12] A. Canossa and A. Drachen, "Patterns of Play: Play-Personas in User-Centred Game Development." in *DiGRA Conference*, 2009, pp. 5:1–10.

[13] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics," *IEEE Trans. on Games*, vol. 11, no. 4, pp. 352–362, 2018.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 143–192, 2012.

[15] P. L. P. de Woillemont, R. Labory, and V. Corruble, "Configurable Agent With Reward As Input: A Play-Style Continuum Generation," in *IEEE Conference on Games (CoG)*, 2021, pp. 1–8.

[16] A. Liapis, H. P. Martínez, J. Togelius, and G. N. Yannakakis, "Transforming Exploratory Creativity with DeLeNoX," in *International Conference on Computational Creativity*, 2013, pp. 56–63.

[17] J.-B. Mouret and J. Clune, "Illuminating Search Spaces by Mapping Elites," *arXiv preprint arXiv:1504.04909*, pp. 1–15, 2015.

[18] M. C. Fontaine, J. Togelius, S. Nikolaidis, and A. K. Hoover, "Covariance matrix adaptation for the rapid illumination of behavior space," *CoRR*, vol. abs/1912.02400, pp. 94–102, 2019. [Online]. Available: http://arxiv.org/abs/1912.02400

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

[20] 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)*. IEEE, 2019, pp. 1–8.

[21] M. C. Fontaine, S. Lee, L. B. Soros, F. de Mesentier Silva, J. Togelius, and A. K. Hoover, "Mapping Hearthstone Deck Spaces through MAP-Elites with Sliding Boundaries," in *Proceedings of The Genetic and Evolutionary Computation Conference*, 2019, pp. 161–169.

[22] I. Bravi and S. Lucas, "Rinascimento: Searching the Behaviour Space of Splendor," *arXiv preprint arXiv:2106.08371*, pp. 1–11, 2021.

[23] 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.

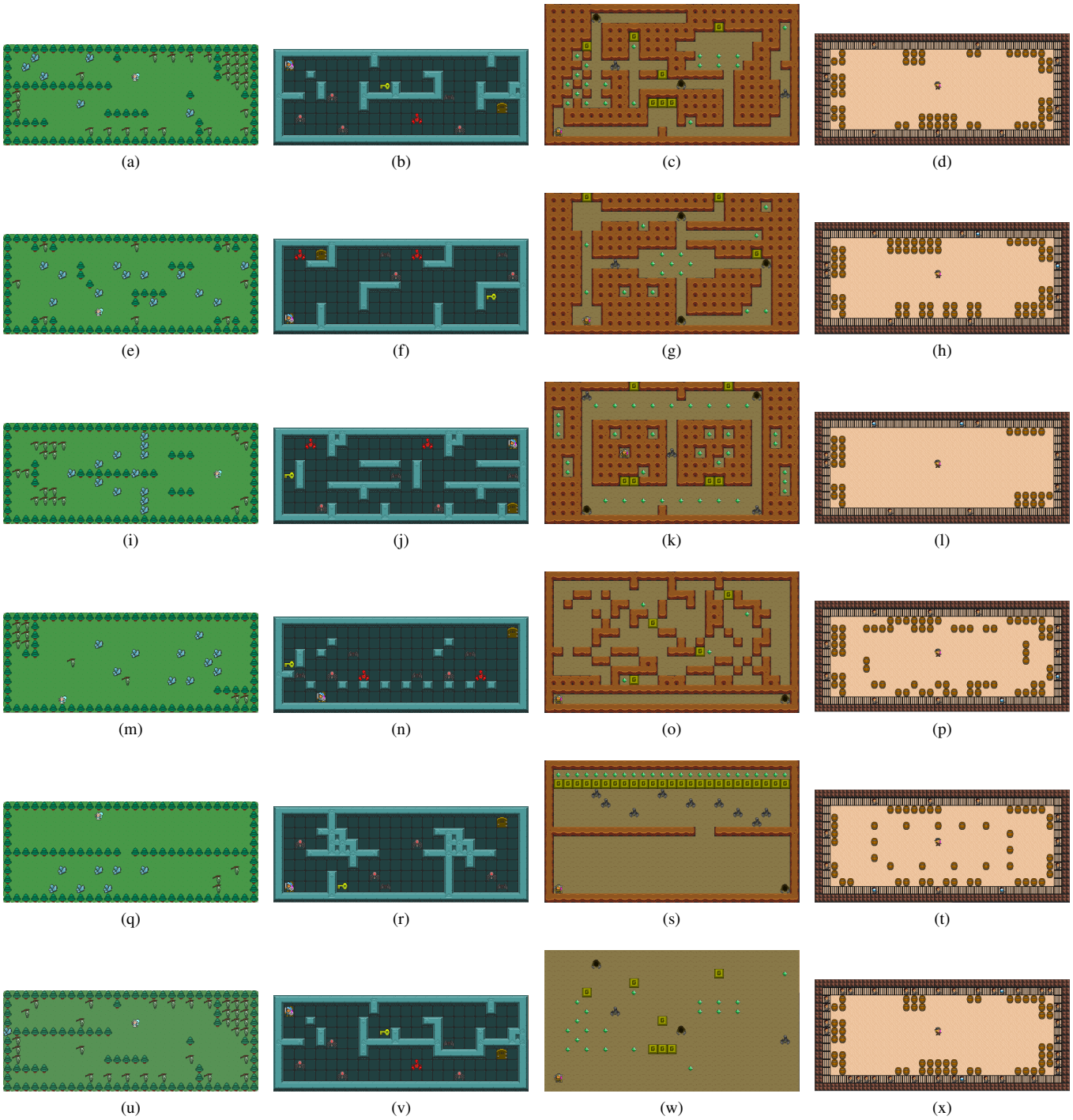



Fig. 7. Screenshot of the levels used in the experiments at t=0. Left to right, the corresponding games are *Butterflies*, *Zelda*, *Digdug* and *Sheriff*. The first row corresponds to the levels used to generate the team of agents (Section V), the levels in rows $2-4$ corresponds to diverse maps with different characteristics used in the portability experiments (Section VII) and last row represents the levels used for the exploratory testing experiment (Section VIII).

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

[25] D. Perez-Liebana, C. Guerrero-Romero, A. Dockhorn, D. Jeurissen, and L. Xu, "Generating Diverse and Competitive Play-Styles for Strategy Games," in *IEEE Conference on Games (CoG)*, 2021, pp. 1–8.

[26] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: A Multitrack Framework for Evaluating agents, Games, and Content Generation Algorithms," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.

[27] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open loop Search for General Video Game Playing," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 337–344.

[28] T. Pierrot, V. Macé, F. Chalumeau, A. Flajolet, G. Cideron, K. Beguir, A. Cully, O. Sigaud, and N. Perrin-Gilbert, "Diversity policy gradient for sample efficient quality-diversity optimization," in *ICLR Workshop on Agent Learning in Open-Endedness*, 2022, pp. 1075—-1083.