Mech-Elites: Illuminating the Mechanic Space of GVGAI

Megan Charity
New York University
Brooklyn, NY
mlc761@nyu.edu

Michael Cerny Green
New York University; OriGen.AI
Brooklyn, NY
mike.green@nyu.edu

Ahmed Khalifa
New York University
Brooklyn, NY
ahmed@akhalifa.com

Julian Togelius
New York University
Brooklyn, NY
julian@togelis.com

ABSTRACT

This paper introduces a fully automatic method of mechanic illumination for general video game level generation. Using the Constrained MAP-Elites algorithm and the GVG-AI framework, this system generates the simplest tile based levels that contain specific sets of game mechanics and also satisfy playability constraints. We apply this method to illuminate mechanic space for 4 different games in GVG-AI: Zelda, Solarfox, Plants, and RealPortals.

CCS CONCEPTS

• **Theory of computation** → **Evolutionary algorithms**; • **Applied computing** → **Computer games**.

KEYWORDS

general video game, level generation, procedural content generation, map elites, evolutionary algorthims

1 INTRODUCTION

Procedural Content Generation (PCG) for video games is a challenging application and active research area for artificial intelligence. Search [30], machine-learning [27], and reinforcement-learning methods [16] are constantly being researched for applications as varied as tutorial generation [12, 13], self-training [5, 32], general video game AI [20], map/level generation [14, 28], and creating entire games [6, 7, 18].

Experienced-Based PCG is PCG applied to create a specific player experience, using a model of player experience as part of the algorithm [34]. Games are created specifically with the player experience in mind. The core emphasis of most games is on the player's actions, reactions, and choices made during gameplay, and game

mechanics are what drive that experience forward on an atomic level. From a player's perspective, a game mechanic is "...everything that affords agency in the game world" [25]. For example, a player performing a jump would be considered a game mechanic, as would a player collecting a coin or destroying a monster. Levels often showcase different game mechanics, either in an isolated setting to help the player hone that mechanic or other times in combination with many others to glean the complex interplay between them.

While simpler games only require a few tutorial environments or levels to teach a player their mechanics, more complex games might need more specific and niche levels in order to effectively teach the player without overwhelming them. Generating these tutorial levels by hand and identifying the critical mechanics needed to play the game can be tedious for the developer and ultimately may not be beneficial to the player if a more suitable tutorial level can be created to achieve the same goal. The AtDelfi project [12] mentioned automated "experience generation" as a future goal of tutorial generation, and this project was primarily motivated to help fill this need.

In this paper, we introduce the use of AI methods to identify the mechanics of a game and generate levels from this list of mechanics. These levels are be mutated to reach a level of fitness that both demonstrates the individual critical mechanic or combination of mechanics using the least amount of noise in the level and within a favorable time constraint. This has been done in the past as a proof of concept for Mario levels, [17], where mechanics were predefined for the algorithm. In this paper, we apply that method, with some modifications, to several different types of games with mechanics being automatically parsed from a game's description file. One potential application with this would be to augment a tutorial generation system, such as AtDelfi [12], so that it can automatically develop levels that teach the critical mechanics of any game. In this paper, we will focus on 4 different games from the GVG-AI framework: Zelda, Solarfox, Plants, and RealPortals. Although the generated levels look much different than the originals, all levels are playable and showcase the activation of a diverse array of mechanic combinations.

2 BACKGROUND

Search-based PCG is a technique that uses search methods to find game content [30]. Evolutionary algorithms is a class of stochastic

optimization methods popularly used for PCG. Such algorithms programmatically apply concepts from Darwinistic evolutionary theory, such as mutation, population, and trans-generational genetic heritage, to find optimized solutions. FI2Pop is one such algorithm that uses a dual-population technique [21]. One of these, the "feasible" population, attempts to improve the overall quality of solutions, or "chromosomes," contained within. These chromosomes can occasionally break constraints while improving and are consequently placed into the "infeasible" population. The fitness function of the infeasible population drives chromosomes toward parameters that bring them back within the constraints of the "feasible" population.

Quality diversity (QD) algorithms are a class of methods that fall under the evolutionary umbrella. QD allows for a simulateous focus on quality of results as well as diversity using explicit metrics for these, separating it from traditional multi-objective optimization strategies, and making it a great candidate for PCG [11]. The Map-Elites (ME) algorithm [22] is one such QD algorithm that maintains a map of n-dimensions in place of a population. The elites are sampled to recreate a competing younger generation, which try to replace their parents. Dimensions usually correspond to a unique behavioral characteristic or trait, such level size, the number of enemies present.

Constrained Map-Elites (CME) [19] is a hybrid genetic algorithm that combines the FI2Pop constrained optimization algorithm with Map-Elites. Within each cell are stored two types of chromosomes, those that are feasible, and those that are not. Chromosomes can be moved between cells (if their dimensions shift) and/or between areas within their cell (if they successfuly outgrow their constraints or fail to do so). Constrained Map-Elites allows for complex quality diversity search to optimize toward a given problem, making it a useful tool for PCG. PlayMapper [33] tweaked with the ME algorithm for Mario AI Framework [29] to illuminate how level and player specific level features to show how search-based pcg techniques can be more effectively used by game designers.

Khalifa et al. [19] used Constrained Map-Elites to develop a range of level types for bullet-hell games, by characterizing levels by the required strategy and dexterity a player would need to be successful. The Evolutionary Dungeon Designer project [1] uses Map Elites to allow mixed-initiative dungeon design. Users can tune the dimension settings to their liking in order to generate interesting level layouts. Most similar to this work, Constrained Map-Elites has been used to generate mini-levels, or "scenes," [17] in the Mario AI Framework, by mapping mechanics triggered during gameplay.

Several research projects have attempted to generate game levels target to explore different dimensions of level space. Ashlock et al. [2, 3] explored different evolutionary techniques for puzzle generation of various difficulties. Jennings et al. built a system which dynamically constructs levels to be appropriately challenging to the player [15]. *Refraction* (Center for Game Science at the University of Washington 2010) generates levels that showcase certain features (which in turn are associated with certain mechanics) [26]. Green et al. [**?**] proposed a method to automatically generate mini-levels in the Mario AI Framework, called "scenes", which required the player to trigger a specific mechanic in order to win. By evolving scenes using constrained Map-Elites, Khalifa et al. [17] built upon this research and was able to generate a multitude of levels that

featured various subsets of game mechanics. The work in this paper is inspired by these previous two projects, as our system works within a general video game domain.

3 GENERAL VIDEO GAME ARTIFICIAL INTELLIGENCE (GVGAI)

The GVG-AI framework is a platform for automatic general video game research [23, 24]. The framework provides multiple competitions to allow AI agents to annually compete against one another [23], including level generation [20], learning [31], rule generation [18], and two-player gameplay [10].

Every game in the GVG-AI Framework is described in Video Game Description Language (VGDL) [9]. VGDL is encompassing enough to describe a wide variety of simple 2D games, yet remains easy to read for humans. Some of them are adaptations of classical games, such as *Plants vs Zombies* (PopCap Games 2009) and *Galaga* (Namco 1981), while others are brand new games, such as *Wait for Breakfast.* In VGDL, one needs to describe game element behaviours, what events occur when elements collide, and how the game is won or lost. A VGDL game consists of two file types: the game description file and one or more level files. Four parts make up the game description file: a *Sprite Set* which determines which game objects exist and how they look and behave, an *Interaction Set* which describes how sprites interact, a *Termination Set* to dictate how the game ends, and finally a *level mapping* between game sprites and their ASCII representation in the level files.

The 4 GVGAI games used in this work are Zelda, Solarfox, Plants, and RealPortals. Each was selected based on a previous work [4], which categorized GVGAI games based on how they were played. These for games contain a diverse array of mechanics, terminal conditions (time-based (Plants), lock-and-key (Zelda and RealPortals), and collection (SolarFox)), and aggregately incorporate ranging levels of complexity. For example, whereas Zelda is a relatively simple lock-and-key game, RealPortals requires complex problem-solving, and Plants contains relatively enormous maps to search. Thus, we selected these as a representative set of the GVGAI framework's games.

- **Zelda:** is a GVGAI port of the dugenon system in *The Legend of Zelda* (Nintendo 1986). The player must pick up a key and unlock a door in order to beat a level. Monsters populate the level and can kill the player, causing them to lose. The player can swing a sword, which can destroy monsters and grant points.
- **Solarfox:** is a GVGAI port of *Solar Fox* (Bally/Midway Mfg. Co 1981). The player must dodge both enemies and their flaming projectiles in order to collect all the "blibs" in the level. The player gains a point for each blib collected, and victory is granted after collecting all blibs on the level. Several levels contain "powered blibs," which are worth no points. If a player collides with a powered blib, it will spawn a "blib generator," which as the name implies, can spawn more blibs to collect and gain more points. If a player touches a blib generator, however, the generator will be destroyed and no longer generate any more blibs. Good gameplay invokes a balance of short- and long-term strategy, balancing the greed
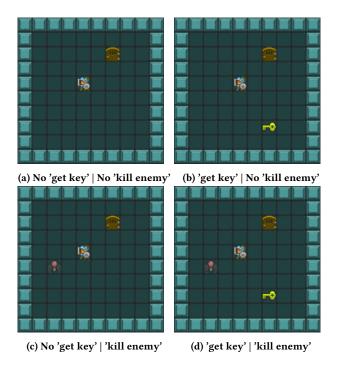
(a) No 'get key' | No 'kill enemy'

(b) 'get key' | No 'kill enemy'

(c) No 'get key' | 'kill enemy'

(d) 'get key' | 'kill enemy'

Figure 1: An example of an optimum MAP-Elites matrix for the Zelda game mechanics "get key" and "attack enemy".

of winning the level against getting more points and risking loss.

- **Plants:** is a GVGAI port of *Plants vs. Zombies* (PopCap Games 2009), a tower defense-style game. If the player survives for 1000 game ticks, they win. Zombies spawn on the right side of the screen and move left, and the player loses if a zombie reaches the left side. Plants, which the player must grow in specific "marsh" tiles, can destroy zombies by automatically firing zombie-killing peas. Each zombie killed is worth a point. Occasionally, zombies will throw axes, which destroy plants, so the player must regrow plants to maintain protection.
- **RealPortals:** is GVGAI 2D port of *Portal* (Valve 2007). The player must reach the goal, which sometimes is behind a locked door that needs a key. Movement is restricted by water, which kills the player if they touch it. To succeed, players need to be creative in overcoming this hazard by using portals which can teleport them across the map. Players need to pick up wands, which allow them to toggle between the ability to create *portal entrances* and *portal exits*. There are also boulders on some levels, which the player can push into the water to transform the water into solid ground, creating land-bridges on which they can walk.

4 METHODS

4.1 Constrained MAP-Elites (CME)

The behavior characteristic of the CME cells consists of multiple binary dimensions that correspond to the game mechanics read

in from their VGDL description file. Each dimension represents whether or not a particular mechanic was performed by the agent. For example, if the game mechanics for Zelda consisted of the list "get key" and "kill enemy", the CME behavioral characteristic will be 2 binary dimensions which will create 4 cells (No "get key"|No "kill enemy", No "get key"|"kill enemy", "get key"|No "kill enemy", and "get key"|"kill enemy"). Figure 1 shows the ideal levels for all these four possible cells.

The list of actions performed by the agent is output by GVG-AI to an external file after each agent run. This file is read back in by the system and parsed, searching through the "dictionary" of the game's mechanic list. The binary representation is built based on whether the pre-specified game mechanics were found in the file. With this, the number of possible cells that can be generated for each game is 2^n with n being the number of game mechanics defined for the game.

4.2 Level Constraints

The constraints of the level generation consisted of 3 parts: the end result of the level (win condition of the agent,) the time it took to complete the level versus the ideal time, and the ratio of successful level completions while the agent is idle.

If the agent successfully completed the level, then only the completion time is evaluated - otherwise, a penalty is applied to the constraint value. A preset value called the "ideal time" was used to compare to the completion time. The closer the completion time is to the ideal time, the better the constraint value. This is to ensure that the agent doesn't complete the level too fast - so the player cannot see the demonstration of the game's mechanics - or too slow - so the tutorial level doesn't drag out longer than needed. The inverse of the absolute value of the distance from the completion time to the ideal time is used as the constraint value, multiplied by any penalty from the win condition. Equation 1 shows the part of the constraint calculation that applies to the time to complete a level,

$$P = \frac{win}{|T_{win} - T_{ideal}|} + \frac{(1 - win) * 0.25}{|T_{survival} - T_{ideal}|} \tag{1}$$

$$E = \begin{cases} 1 & \text{if } \frac{N_{pass}}{N_{total}} \geq 0.5 \\ \frac{N_{pass}}{N_{total}} & \text{otherwise} \end{cases} \tag{2}$$

$$C = P + E \tag{3}$$

where win represents a 1 if the agent finished the level sucessfully and a 0 otherwise, T_{ideal} represents the ideal time pre-defined for the system, and T_{win} and $T_{survival}$ represent the finishing time of the agent before successful completion and unsuccessful completion respectively.

In addition to the win condition and completion time, a "Do Nothing" agent is run on the level for a certain number of times. This agent does not perform any user actions and remains idle in the level. If this agent dies before reaching T_{ideal}, the level fails the evaluation. If the agent does not survive for a majority of the evaluations tested, the ratio of successful idle agents over the number of times attempted is applied to the constraints. This is to remove the chance that the evaluation agent happened to "get lucky" on its performance in the level and keep the level reasonably user-friendly. Equation 2 demonstrates how the idle agent's trials

were applied, with N_{pass} representing the number of times the idle agent survived to the ideal time, and N_{total} representing the number of idle agent tests.

The total constraint score of a level is therefore equivalent to C in Equation 3, where P is from Equation 1 and E is from Equation 2. In order to be considered as an "elite" level for a MAP-Elites cell, the level must reach a certain threshold of constraints. If the threshold is reached, the level's fitness value will be evaluated to the cell's current elite fitness value. If the level evaluation does not reach this threshold, the level is placed in the MAP-Elites cell's "infeasible" population instead.

It is possible for an unwinnable level - or a level showing a representation of "lose" mechanics as opposed to "win" mechanics - to be evaluated as an elite level. In the case of the levels showing a representation of winning mechanics, the constraint value would be set to the maximum value of 1, so any penalties applied to the constraint value thereafter would be due to either a poor finishing time (i.e. T_{win} was not close enough to T_{ideal}) or by failing the idle agent tests. In the case of levels showing a representation of losing mechanics, a level can still have a good constraint value (i.e. pass the constraint threshold) if $T_{survival}$ is extremely close to T_{ideal} and also passes the idle tests. However, this end value will also be penalized to Âij of the original value since the level was still not finished sucessfully.

The constraint threshold is chosen with the intention that a level will both be winnable and pass the idle agent tests, but may be within a certain range for T_{ideal}. For example, if the constraint threshold is set to 0.1, based on the function defined for the constraint value, a level can be considered elite in two cases: if it is winnable and passes the DoNothing tests, so long as T_{win} is within 10 timesteps of T_{ideal}; or if it is unwinnable, but passes the DoNothing tests and $T_{survival}$ is within 2 timesteps of T_{ideal}.

4.3 Level Fitness

The fitness of a level was determined by the tile entropy and the derivative-tile entropy of the level. Since the level was not seperated into vertical slices and mutated on the slices like the Mario levels, [17] neighboring tile values in the level were used as comparison instead. Levels that are simplistic tend to be more aesthetically pleasing and enjoyable than ones that are noisy and chaotic. Minimalistic levels can also more clearly showcase meaningful elements within the level. All of this coincides with the motivation of this paper, which is to use this method to generate tutorial levels, which should be simple [13]. Minimizing entropy in a level will create fewer distractions for the player while they are playing the level and exploring different mechanics. The system evolves to create open, mostly empty-tiled levels or levels with similar tiles placed adjacent to each other that still demonstrate the game mechanics needed to play the game. Weights were given to the importance of the tile entropy versus the tile derivative entropy to create less noisy levels. Equation 4 was used for the level fitnesses:

$$fitness = H(lvl) * w + H(\Delta lvl) * (1 - w) \qquad (4)$$

where $H(lvl)$ represents the raw tile entropy of a level, $H(\Delta lvl)$ represents the entropy of the derivative of the same level which was based on the number of identical surrounding tiles in the 12x10 level, and w represents the pre-set weight value.

4.4 Mutation

The elite levels and best infeasible population levels from the MAP-Elites cells are randomly selected and used as a basis for mutated levels. From there, a random tile of the level is selected and turned into randomly to a new tile value specified by the game's VGDL file. Then, based on some set probability, another tile from the level is randomly chosen and mutated. This process continues until the probability check fails. The end result is a mutated level taken from the elite level that will, ideally, yield a better fitted level for the current game mechanic combination representation cell or another representation cell in the MAP-Elites matrix. This mutated level is evaluated in the next iteration's population of levels.

4.5 Population Generation

Initially, levels are generated randomly using GVG-AI's random level generator class. An agent created from the GVG-AI competitions was used to evaluate the levels generated in each population. The evaluation agent runs through all the levels of the population. At the end of each run, the system determines the game mechanic binary string representation from the actions taken by the agent and calculates the constraint and fitness value. The levels are placed into the MAP-Elites matrix according to their constraint and fitness scores and triggered mechanics.

The consecutive population is generated by randomly selecting the elite levels or the best infeasible level - based on a predefined probability value - from the MAP-Elites matrix cells and mutating the levels. A set portion of the population is generated from mutated elite levels, while the remaining portion is generated from GVG-AI's random level generator. This is to help to prevent the algorithm from potentially reaching a local optimum during generation and mutation. This process is repeated indefinitely to create the best levels for each game's mechanic combination representation.

5 EXPERIMENTS

4 games from the GVGAI framework were used to test the effectiveness of our level generation method. Zelda, Solarfox, Plants, and RealPortals are all described in Section 3. The dimensionalities used in the system are shown in Tables 1, 2, 3, and 4. These mechanics were extracted from each game automatically by using the AtDelfi system [12], which is able to parse game rules directly from a GVGAI game's VDGL description file.

All experiments used batch size of 50 chromosomes. Each generation, 20% of the levels are randomly initialized, and the remainder are filled with mutated versions of elites. For the Solarfox experiment, levels are much more dependent on having fewer empty tiles for functional gameplay and thus a less constrained initialized population. To assist evolution, only 10% of Solarfox levels were randomly initialized each generation, and 90% filled with mutated version of elites.

Our system uses the AdrianCTX algorithm that comes with the GVGAI framework [23] as the *evaluation agent*. The dimensions for the level are calculated using the AdrianCTX agent's playthrough. The *idle agent* is run a total of 5 times on the level, of which it must survive 3 in order to pass the constraint test. In RealPortals, it is impossible for a non-moving agent to die (the only way to lose is to fall into water), and therefore this constraint test is not necessary.
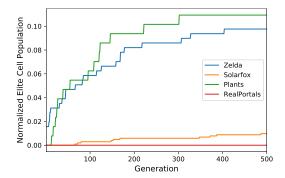
Figure 2: Number of Elite MAP Cells (Normalized) across generations. Although RealPortals is considerably lower than the other games, it's map contained 231 elites, nearly ten times more than any other game.

*	Dimension	Description
z1	space-nokey	Agent pressed the space bar when the avatar did not have a key
z2	space-withkey	Agent pressed the space bar when the avatar had a key
z3	stepback	A sprite ran into another sprite
z4	kill-nokey	A sprite killed the avatar when the avatar did not have a key
z5	kill-withkey	A sprite killed the avatar when the avatar had a key
z6	sword-kill	The agent killed an enemy sprite with a sword
z7	getkey	The agent picked up a key
z8	touchgoal	The agent touched a goal with the key and won the game

Table 1: Constrained MAP-Elites dimensions for the GVG-AI game Zelda

Both agent's "ideal times" (T_{ideal}) were set to 70 timesteps for all experiments.

If the constraint test is passed, the level's fitness is evaluated according to Equation 4, where w is 0.25 and $(1 − w)$ is 0.75 in the Zelda and RealPortals experiments. Plants and Solarfox levels are more dependent on tile uniformity and open areas, as opposed to Zelda and RealPortals. Therefore, w was 0.2 while $(1 − w)$ is 0.8 for thes experiments. After evaluation, the chromosome is compared to its respective dimensional family, as specified in Section 4.1.

After evaluation, the system populates the next generation using the elites and infeasible populations contained in the MAP-Elite cells. A newly initialized level has a 50% chance of being mutated from the elite/feasible population of a MAP-Elite cell. Otherwise, the level is mutated from the cell's best level from the infeasible population. This repolulation process is iteratively done until the generation contains 50 new chromosomes. For all four games, a single MAP-Elite cell is allowed to store a maximum of 20 infeasible levels and 1 elite level. Each experiment ran for a total of 500 iterations.

6 RESULTS

The normalized elite counts across generations for our experiments is displayed in Figure 2. Each experiment was normalized against its total possible elite cell count, calculated using the game's mechanic dimensionality specified in Tables 1, 2, 3, and 4.

6.1 Mechanical Frequency in Elites

Figure 7 displays the symbolicly represented mechanics present across all games and how prevalent each exists among that game's elite population. There are 3 mechanics (d, h, and i) in RealPortals that are never expressed within any of the elites. These correspond to the "drown," "teleport-exit," and "no-moving-boulder" mechanics. The irony of the low activation of teleport mechanics ("teleport-entrance" = 8%, "teleport-exit" = 0%) in a game called "RealPortals" is not lost on us, and this is further represented when looking at the elites themselves, which do not require teleportation to win. However, no agent has ever been submitted to the GVGAI competition can reliably beat RealPortals levels. The system's constraint

function, which drives evolution to produce beatable levels, causes the generator to develop levels simple enough for the agent to win. Because teleportation drastically expands the space that the agent needs to search, the simplest solution for the generator is to remove the need to teleport.

6.2 In-Depth Game Analysis

In the following subsections, we present a representative subset of each game's generated levels. The *mean* and *mode* levels correspond to the mean and mode number of mechanics triggered across all elites. When multiple elites contained the identical amount of mechanics for either mean or mode, we randomly sampled among these elites to display one of them. We realize that this is only a

(a) mean: z1, z2, z3, z5, z7

(b) mode: z1, z2, z3, z7, z8

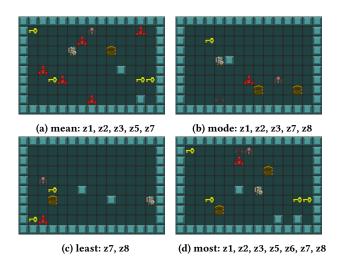

(c) least: z7, z8

(d) most: z1, z2, z3, z5, z6, z7, z8

Figure 3: A subset of generated elite levels for Zelda. Their string representation corresponds to their showcased mechanics detailed in Table 1.

*	Dimension	Description
s1	hit-wall	Agent hit a wall
s2	hit-enemyground	Agent touched enemy ground
s3	hit-avatar	An enemy sprite hit the Agent
s4	touch-powerblib	The agent touched a powerblib
s5	spawn-more	A turning powerblib created a blib
s6	change-blib	A turning powerblib changed into a normal blib
s7	overlap-blib	A powerblib overlapped with a blib
s8	get-blib	The agent got a blib
s9	reverse-direction	A sprite hit a wall and reversed direction
s10	enemy-shoot	An enemy fired a missile at the avatar

Table 2: Constrained MAP-Elites dimensions for the GVG-AI game Solarfox

*	Dimension	Description
p1	space	Agent pressed the SpaceBar
p2	hit-wall	A sprite touched a wall
p3	kill-plant	A zombie destroyed a plant
p4	zombie-goal	A zombie sprite reaches the goal
p5	pea-hit	A pea sprite hits a zombie sprite
p6	tomb-block	A tomb sprite blocks a pea sprite
p7	make-plant	Agent placed a plant on a marsh tile

Table 3: Constrained MAP-Elites dimensions for the GVG-AI game Plants

subset of the possible elites, and that dimensional similarity does not necessarily equate to structural similarity.

6.2.1 Zelda. After 500 iterations, 55 out of 256 possible cells were populated for the Constrained MAP-Elites matrix of Zelda, with 25 cells containing an elite map. The average fitness for these cells was 0.5186. Figure 3 displays 4 elites at opposite ends of the dimensional spectrum. The mean and mode elites are calculated to have 4.64 and 5 dimensions. respectively. The map with the least dimensionality (Figure 3c) showcased 2 mechanics, out of a possible 8. The map with the most mechanic-dimensionality 3d contained 7.

The least dimensional map matches up with the AtDelfi system's critical mechanics [12]. We can see however, that other mechanics from Table 1 are capable of being triggered in this space, such as killing, or being killed by, monsters. Most likely due to the wide-openess of the space, the agent failed to bump into any walls or monsters on its bee-line route to the key and the door. The most dimensional map showcases nearly all possible mechanics, only missing kill-nokey. We can interpret this to mean the agent went for the key first, before dealing with monsters. At a glance, it would be possible to also trigger the kill-nokey mechanic, depending on agent priority.

6.2.2 Solarfox. After 500 iterations, 52 out of 1024 possible cells were populated for the Constrained MAP-Elites matrix of Solarfox game, with 10 cells containing an elite map. The average fitness across all cells was 0.4311. Figure 4 displays 4 elites at opposite ends of the dimensional spectrum, while Figure . The mean and mode are calculated to be 5.1 and 3 respectively. The least mechanic-dimensional map contained only 3 mechanics (Figure 4c), whereas the most (Figure 4d) contained 8 out of the possible 10 mechanics.

The representative least dimensional elite (Figure 4c presents a lightly populated level containing just a few enemies, blibs, and walls. It contains no powerblib-generators, only normal blibs, which

are placed incredibly close to the player at start for an easy win. The most dimensional elite contains nearly every mechanic in the game except for 2: the agent hitting a wall (which are placed far from the player's initial start), and the agent touching and dying on the enemy ground tile.

6.2.3 Plants. After 500 iterations, 31 out of 128 possible cells were filled for the Constrained MAP-Elites matrix of Plants, with 14 cells containing an elite map. The average fitness across all cells was 0.3993. Figure 5 displays 4 elites of varying dimensions. The mean and mode are calculated to be 3.93 and 5 respectively. The map with the least dimensionality showcased just 1 mechanic. The map with the most mechanic-dimensionality contained 6 out of the 7 possible mechanics.

Unlike any of the other elites of any other game, the representative least dimensional elite of Plants contains a single mechanic, which happens to be one that causes the player to lose the game. Based on the game rules, it is not possible to win this level no matter what actions the player does, as the zombies will spawn several tiles to the right of the villager and inevitably collide with it. The elite with the most activated dimensions, on the other hand, was
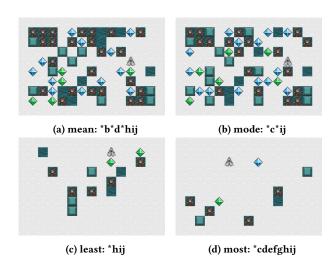

(a) mean: *b*d*hij

(b) mode: *c*ij

(c) least: *hij

(d) most: *cdefghij

Figure 4: A subset of generated elite levels for Solarfox. The string representation corresponds to their showcased mechanics detailed in Table 2.

(a) mean: p1, p3, p4, p5, p7

(b) mode: p5, p6, p7
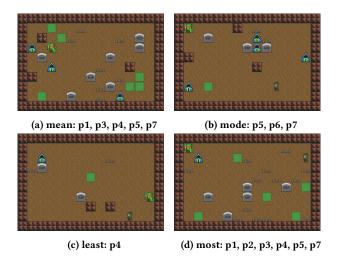
(c) least: p4

(d) most: p1, p2, p3, p4, p5, p7

Figure 5: A subset of generated elite levels for Plants. Their string representation corresponds to their showcased mechanics detailed in Table 3.

(a) mean: r1, r3, r5, r11, r14, 15, r16, r19, r20, r21, r24

(b) mode: r1, r3, r5, r13, r15, r17, r18, r19, r20, r21, r22

(c) least: r1, r3, r14, r19, r22

(d) most: r1, r5, r7, r10, r11, r13, r14, r15, r16, r19, r20, r23, r25, r26, r34, r35
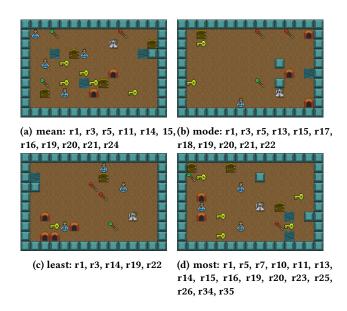
Figure 6: A subset of generated elite levels for realportals. Their string representation corresponds to their showcased mechanics detailed in Table 4.

possible to win. The triggered mechanics guarantee that a player could encounter most of the mechanics in the game during play.

6.2.4 RealPortals. After 500 iterations, 6966 cells were filled for the Constrained MAP-Elites matrix of RealPortals, with 231 cells containing an elite map. The average fitness across all cells was 0.4257. Figure 6 displays 4 elites of varying dimensions. The mean and mode are calculated to be 10.78 and 10 respectively. The least mechanic-dimensional map contained 5 mechanics, and the most contained 16 out of a possible 35.

*	Dimension	Description
r1	space	Agent pressed the SpaceBar
r2	change-key-blue	changes blue avatar's current resource to a key
r3	hit-wall	A sprite touched a wall
r4	drown	destroy any sprite that falls in the water
r5	toggle-blue	avatar changes current portal shot to blue
r6	no-lock	any sprite tried to move through a lock
r7	no-portalexit	any sprite tried to move through an exit portal
r8	teleport-exit	orange avatar steps through the entrance portal
r9	no-moving-boulder	sprite tried to move through a moving boulder
r10	no-idle-boulder	sprite tried to move through an idle boulder
r11	change-key-orange	changes orange avatar's current resource to a key
r12	toggle-orange	avatar changes current portal shot to orange
r13	teleport-entrance	blue avatar steps through exit portal
r14	get-weapon	avatar picks up a weapon
r15	get-key	avatar picks up a key
r16	back-to-wall	portal turns back into a wall
r17	fill-water	moving boulder falls into water to fill it
r18	open-lock	avatar unlocks a lock
r19	touchgoal	The agent touched a goal and won the game
r20	make-portal	wall turns into a portal
r21	portal-missile-velocity	send a missile through a portal at the same velocity
r22	cover-goal	goal is covered by a missile
r23	blue-missile-in	send a blue missile thru a portal entrance
r24	orange-missile-in	send an orange missile thru a portal entrance
r25	portal-boulder	send a boulder through a portal at the same velocity
r26	stop-boulder-key	moving boulder stops after hitting a key
r27	stop-boulder-wall	moving boulder stops after hitting a wall
r28	stop-boulder-blue-toggle	moving boulder stops after hitting a blue portal toggle
r29	stop-boulder-orange-toggle	moving boulder stops after hitting an orange portal toggle
r30	stop-boulder-lock	moving boulder stops after hitting a lock
r31	teleport-boulder	sends a boulder to the other portal
r32	stop-boulder-boulder	moving boulder stops after hitting another boulder
r33	stop-boulder-avatar-blue	moving boulder stops after hitting the blue avatar
r34	stop-boulder-avatar-orange	moving boulder stops after hitting the orange avatar
r35	roll-boulder	boulder moved over a tile

Table 4: Constrained MAP-Elites dimensions for the GVG-AI game RealPortals

In contrast to the other games described above, RealPortals is extremely complex, echoed in the sheer amount of elites. Ironically, the generated levels are all extremely simple to solve, unlike any of the GVGAI included levels. Without water dividing the map and requiring the player to use portaling, the levels are transformed into a simple find-the-goal game even simpler than Zelda. Even if the agent uses a portal, there is no need to do so or to use any of the other game mechancis which are normally required by the framework game levels (pushing boulders into water, unlocking the lock with a key, etc). The least dimensional representative elite still activates five mechanics (with others still possible, just not activated during playthrough), whereas the most dimensional elite can be beaten by taking two steps to the right.

7 DISCUSSION

Constrained MAP-Elites was able to populate more than 10% and slighly less than 10% of the total cells with elites for Plants and Zelda respectively. Compared to Solarfox (approx. 1%) and Real-Portals(>1%), these two games' dimensions were relatively well-explored. At first glance, it would make sense that RealPortals was not as explored, due to its 34-dimensional complexity compared to Zelda's meager 8-dimensions. However, Solarfox (10) is also many less dimensional than RealPortals, but has a similarly low relative elite population. Keep in mind that though the dimensionality of Solarfox relative to Zelda is only 2 higher, the dimensional space increases from 256 possible cells to 1024. We also hypothesize that elite population is impacted not only by the total number of game mechanics, but also by the intricacies of how the mechanics interact with each other. In Solarfox, the player wins by collecting all the "blibs"; however, not all blibs are equally scored. If the player is patient, they can let certain "powerblibs" spawn more blibs, and therefore gain more points. We theorize that it is difficult to create levels which force an agent to wait to collect these blibs, and therefore trigger these spawning mechanics in conjunction with others.

Across all games, when compared to the original levels, the generated levels provide a sense of uniformity. Solarfox levels tend to be sparsely populated with blibs, with a few exceptions (Figure 4b. There tend to be no water tiles present in RealPortals, or marsh tiles in Plants, relative to each games' original levels. The Zelda elites consist of wide open spaces, instead of the usual maze-like patterns. We hypothesize that all of this is due to the entropy pressure from the fitness calculation specified in Equation 4, which drives evolution towards creating simplistic levels.

Because of how the fitness is defined and dimensions are calculated, any activated mechanic on a map is guaranteed to have the *possibility* of occuring during a playthrough. However, this does not guarantee that the mechanic *must* be activated or that *other mechanics* do not have the possibility of happening at all. To guarantee either of these, the system would have to exhaustively search all possible game states of the level, which is not computationally feasible within a timely manner for any of these games. Better agents tend to be ones that are better informed of the game, and one way to do this might be to take advantage of hypterstate information [8]. Another direction would be to aggregate the unique mechanics triggered across a multitude of agents, to get a better sense of the mechanic space.

It is interesting that none of these levels incorporate the patterns that the original GVGAI levels contained. For example, most original Solarfox levels presented geometric arrangements of blibs and powerblibs, which were aesthetically pleasing levels to play. The original levels also contained enemies firing missiles from the top and bottom part of the level, with the player and blibs sandwhiched between, unlike the system-generated levels in which enemies and blibs can be anywhere on the map. The entropy pressure exerted by the fitness function pushed for uniformity during generation, which resulted in either highly populated levels filled with certain tiles (Figures 4a and 4b), or lowly populated levels with large amounts of empty space (Figures 4c and 4d).

8 CONCLUSION

Outside of actual gameplay, isolating mechanics from each other to allow players to examine the full breadth of each game mechanic is a non-trivial problem. Our system looks to solve this issue by generating levels that are constructed using a bottom-up approach, evolving levels that are beatable and simple, while using the illuminating power of MAPElites to categorize levels by mechanics triggered. Using our method, one can generate a sampling of the possible mechanic combinations for four GVG-AI games and examine the behavior of their evolution throughout each generation as well as which game mechanics were critical to gameplay.

This system serves as a proof of concept for developing isolated mechanic levels and could be beneficial in developing tutorial levels where mechanics would need to be demonstrated individually or in combination in a controlled environment. This would go hand-in-hand with an automatic tutorial generation system, which could use these levels for player practice. This system could also be beneficial in examining the minimal level structure needed for a game mechanic. The level design for each mechanic representation could be based their architechture on the MAPElites cells generated from the pre-set list of game mechanics.

In future work, we would like to test our system on more games from the GVG-AI framework, such as games with fewer mechanic combinations needed but with more variability in the level structure as well as games with more complex series of mechanic combinations needed to win the game (i.e Frogs or Sokoban.) Expansion of the system to games outside of the framework would also help to prove the usability and generability of the system for games that do not have a predefined description language.

REFERENCES

[1] Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. 2019. Empowering quality diversity in dungeon design with interactive constrained MAP-Elites. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–8.

(a) Zelda

(b) Solarfox
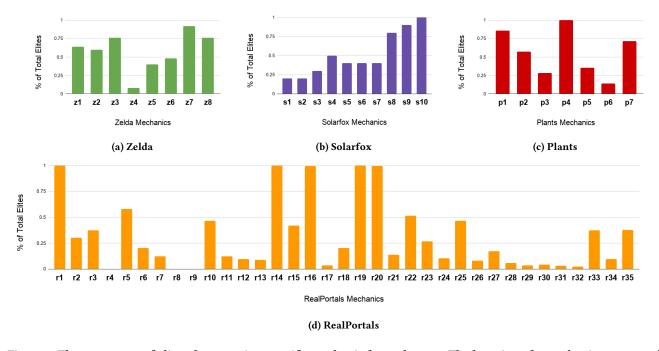
(c) Plants

(d) RealPortals

Figure 7: The percentage of elites that contain a specific mechanic for each game. The lettering of a mechanic corresponds to that games mechanic table. Zelda: Table 1; Solarfox: Table 2; Plants: Table 3; RealPortals: Table 4.

[2] Daniel Ashlock. 2010. Automatic generation of game elements via evolution. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 289–296.

[3] Daniel Ashlock, Colin Lee, and Cameron McGuinness. 2011. Search-based procedural generation of maze-like levels. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 260–273.

[4] Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. 2016. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[5] Michael Cerny Green, Benjamin Sergent, Pushyami Shandilya, and Vibhor Kumar. 2019. Evolutionarily-Curated Curriculum Learning for Deep Reinforcement Learning Agents. *arXiv preprint arXiv:1901.05431* (2019).

[6] Michael Cook, Simon Colton, and Jeremy Gow. 2016. The angelina videogame design systemâĂŤpart i. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 2 (2016), 192–203.

[7] Michael Cook, Simon Colton, and Jeremy Gow. 2016. The angelina videogame design systemâĂŤpart ii. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 3 (2016), 254–266.

[8] Michael Cook and Azalea Raad. 2019. Hyperstate space graphs for automated game analysis. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[9] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. 2013. Towards a video game description language. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] Raluca D Gaina, Diego Pérez-Liébana, and Simon M Lucas. 2016. General video game for 2 players: Framework and competition. In *Computer Science and Electronic Engineering*. IEEE, 186–191.

[11] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. 2019. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[12] Michael Cerny Green, Ahmed Khalifa, Gabriella AB Barros, Tiago Machado, Andy Nealen, and Julian Togelius. 2018. AtDELFI: automatically designing legible, full instructions for games. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM, 17.

[13] Michael Cerny Green, Ahmed Khalifa, Gabriella AB Barros, and Julian Togellius. 2017. " Press Space to Fire": Automatic Video Game Tutorial Generation. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[14] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. 2009. Evolving content in the galactic arms race video game. In *2009 IEEE Symposium on Computational Intelligence and Games*. IEEE, 241–248.

[15] Martin Jennings-Teats, Gillian Smith, and Noah Wardrip-Fruin. 2010. Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the*

2010 Workshop on Procedural Content Generation in Games. 1–4.

[16] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. 2020. PC-GRL: Procedural Content Generation via Reinforcement Learning. (2020). arXiv:cs.LG/2001.09212

[17] Ahmed Khalifa, Michael Cerny Green, Gabriella Barros, and Julian Togelius. 2019. Intentional computational level design. In *Proceedings of The Genetic and Evolutionary Computation Conference*. 796–803.

[18] Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. 2017. General video game rule generation. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 170–177.

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

[20] Ahmed Khalifa, Diego Perez-Liebana, Simon M Lucas, and Julian Togelius. 2016. General video game level generation. In *Genetic and Evolutionary Computation Conference*. ACM, 253–259.

[21] Steven Orla Kimbrough, Gary J Koehler, Ming Lu, and David Harlan Wood. 2008. On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190, 2 (2008), 310–327.

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

[23] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D Gaina, Julian Togelius, and Simon M Lucas. 2019. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. *Transactions on Games* (2019).

[24] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M Lucas. 2016. General video game ai: Competition, challenges and opportunities. In *AAAI Conference on Artificial Intelligence*.

[25] Miguel Sicart. 2008. Defining game mechanics. *Game Studies* 8, 2 (2008), n.

[26] Adam M Smith, Erik Andersen, Michael Mateas, and Zoran Popović. 2012. A case study of expressively constrainable level design automation tools for a puzzle game. In *International Conference on the Foundations of Digital Games*. ACM, 156–163.

[27] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

[28] Adam James Summerville, Shweta Philip, and Michael Mateas. 2015. Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

[29] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. 2010. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*. IEEE, 1–8.

[30] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[31] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. 2018. Deep Reinforcement Learning for General Video Game AI. In *Computational Intelligence and Games*. IEEE, 1–8.

[32] Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius. 2019. Bootstrapping Conditional GANs for Video Game Level Generation. *arXiv preprint arXiv:1910.01603* (2019).

[33] Vivek R Warriar, Carmen Ugarte, John R Woodward, and Laurissa Tokarchuk. 2019. PlayMapper: Illuminating Design Spaces of Platform Games. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–4.

[34] Georgios N Yannakakis and Julian Togelius. 2011. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing* 2, 3 (2011), 147–161.