Interactive Evolution and Exploration Within Latent Level-Design Space of Generative Adversarial Networks

Jacob Schrum
Southwestern University
Georgetown, Texas, USA
schrum2@southwestern.edu

Jake Gutierrez
Southwestern University
Georgetown, Texas, USA
gutierr8@southwestern.edu

Vanessa Volz
modl.ai
Copenhagen, Denmark
vanessa@modl.ai

Jialin Liu
Southern University of Science and Technology, Shenzhen, China
liujl@sustech.edu.cn

Simon Lucas
Queen Mary University of London
London, UK
simon.lucas@qmul.ac.uk

Sebastian Risi
modl.ai, IT University of Copenhagen
Copenhagen, Denmark
sebastian@modl.ai

ABSTRACT

Generative Adversarial Networks (GANs) are an emerging form of indirect encoding. The GAN is trained to induce a latent space on training data, and a real-valued evolutionary algorithm can search that latent space. Such Latent Variable Evolution (LVE) has recently been applied to game levels. However, it is hard for objective scores to capture level features that are appealing to players. Therefore, this paper introduces a tool for interactive LVE of tile-based levels for games. The tool also allows for direct exploration of the latent dimensions, and allows users to play discovered levels. The tool works for a variety of GAN models trained for both *Super Mario Bros.* and *The Legend of Zelda*, and is easily generalizable to other games. A user study shows that both the evolution and latent space exploration features are appreciated, with a slight preference for direct exploration, but combining these features allows users to discover even better levels. User feedback also indicates how this system could eventually grow into a commercial design tool, with the addition of a few enhancements.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Genetic algorithms**; **Learning latent representations**;

KEYWORDS

Generative Adversarial Network, Interactive Evolution, Latent Variable Evolution, Procedural Content Generation, Video Games

ACM Reference Format:
Jacob Schrum, Jake Gutierrez, Vanessa Volz, Jialin Liu, Simon Lucas, and Sebastian Risi. 2020. Interactive Evolution and Exploration Within Latent Level-Design Space of Generative Adversarial Networks. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3389821

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '20, July 8–12, 2020, Cancún, Mexico
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3389821

1 INTRODUCTION

Recent work [11, 15, 29, 42, 46] has shown that it is possible to learn the structure of video game levels using Generative Adversarial Networks (GANs [13]). Although GANs can learn a representation of the underlying structure, the semantics of the game are generally not included in the representation. As a result, levels may not be beatable, because there may not be a key to a door, or a path between two points of interest. To combat this issue, some GAN-based approaches use some form of bootstrapping to enhance solvability [29, 42]. Similarly, the original work applying GANs to Mario [46] used latent variable evolution (LVE) to favor levels that could actually be beaten as determined by AI simulations. However, fitness-based optimization tends to converge on a limited set of points within a search space, which ignores evolution's massive potential for exploration.

In addition, it is often difficult to formally define an evaluation function that can assess the quality of generated game levels automatically. Even with divergent search techniques such as Novelty Search [20] and MAP-Elites [26], which seek to explore large areas of the search space, some explicit characterization of the space is needed (behavior characterization or binning scheme) to dictate which offspring are preferred.

An alternative is interactive evolution, which can explore a search space with a human in the loop [3, 10, 40, 50]. Interactive evolution allows a user to select whatever options are most appealing at the moment, allowing for serendipitous discovery of novel solutions without formalized domain knowledge. This paper applies interactive latent variable evolution to GANs trained on two video games: Super Mario Bros. and The Legend of Zelda.

A typical problem with interactive evolution, however, is a user's frustration with evaluating many low-quality individuals. Using a GAN as the mapping function addresses this issue [3, 50] by assuring that most genotypes presented to the user lead to well-formed phenotypes.

Besides standard interactive evolution, the system provides users a rich environment for interaction with the generated content. For example, users are also able to directly edit latent vectors, providing another means of latent space exploration. Applying the search to game levels also means that users can immediately play their creations, and although some levels are unbeatable, the overall experience is an enjoyable game in itself. The resulting software

is available online[1] and can be used to interactively evolve Mario and Zelda levels without modification. The approach is also easily extensible, and should therefore allow for interactive latent space exploration in various other domains.

A small user study is conducted to assess the system, and gather suggestions for improvement. In particular, users searched latent space using only evolution, or only direct latent vector editing. After trying both, they indicated their preference. Users then tried the combined system and were asked if the combination made results even better. Users had a slight preference for direct exploration, but found that combining search techniques was better than either individually. Users also gave advice for improving the system.

The paper proceeds by discussing related work (Section 2) and the particular video game domains used in this paper: Mario and Zelda (Section 3). Then the approach introduced in this paper is described (Section 4). Next, the procedure and results for a human subject study designed to analyse the new system are presented (Section 5). Finally, the results and possible future work are discussed (Section 6) before concluding (Section 7).

2 RELATED WORK

Procedural Content Generation for games is discussed, followed by descriptions of technical tools used in this paper such as Generative Adversarial Networks (GANs), which are used as a genotype-to-phenoype mapping for interactive evolution of game levels.

2.1 Procedural Content Generation

AI methods have been widely used for generating game content (e.g. game rules, levels, maps/mazes, characters, items, vehicles, stories, textures, and sound) with limited or indirect user input [18, 34, 49]. Yannakakis and Togelius [49] divide existing Procedural Content Generation (PCG) methods into the following categories: search-based methods, solver-based methods, grammar-based methods, cellular automata, and noise/fractals, all of which have been applied to the automatic generation of game levels/dungeons. In recent years, training of machine learning models on existing game contents to generate new content has emerged as a viable approach [37]: PCG via Machine Learning (PCGML). Two popular domains for applying PCGML are the generation of Mario levels and dungeons.

Dungeons are a common type of game level. The earliest algorithmic creation of dungeons goes back to Rogue (1980) [49]. Since then, many PCG methods have been used to generate dungeons [1, 5, 7, 43]: cellular automata, search-based and grammar-based methods, machine learning [38], and hybrid methods [21].

Mario level generation is a popular test case for PCG methods. Since the 2010 Mario AI Championship [35], the Mario AI framework has been widely used to validate approaches for generating levels for Super Mario Bros. [19, 33, 36]. For instance, Summerville and Mateas [36] applied Long Short-Term Memory Recurrent Neural Networks (LSTMs) to generate Mario levels, and then improved the generated levels by incorporating player path information. Jain et al. [19] trained auto-encoders to generate new levels using a binary encoding where empty (accessible) spaces were represented by 0 and the others by 1. More recently, GANs were used to generate Mario levels [24, 46].

[1]https://github.com/schrum2/GameGAN

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. [13]. Their training process can be seen as a two-player adversarial game in which a generator G (faking samples decoded from a random noise vector) and a discriminator D (distinguishing real/fake samples and outputting 0 or 1) are trained at the same time by playing against each other (Fig. 1). The discriminator D aims at minimizing the probability of misjudgment, while the generator G aims at maximizing that probability. Thus, the generator is trained to deceive the discriminator by generating samples that are good enough to be classified as genuine. Training ideally reaches a steady state where G reliably generates realistic examples and D is no more accurate than a coin flip. Formally, this training process can be described using the minimax objective:

$$\min_{G} \max_{D} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r} [\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g} [\log(1 - D(\tilde{\boldsymbol{x}}))], \quad (1)$$

where $\mathbb{P}_r$ is the distribution of the training data and $\mathbb{P}_g$ the distribution generated by $G(z)$ with z being the input to the generator. While the latent vector z is normally sampled from a noise distribution, in the work presented here it is optimized through interactive evolution.

GANs quickly became popular in some sub-fields of computer vision, such as image generation. However, training GANs is not trivial and often results in unstable models. Many extensions have been proposed, such as Deep Convolutional Generative Adversarial Networks (DCGANs [30]), a class of Convolutional Neural Networks (CNNs); Auto-Encoder Generative Adversarial Networks (AE-GANs [25]); and Plug and Play Generative Networks (PPGNs [27]). A particularly interesting variation are Wasserstein GANs (WGANs [2, 14]). WGANs minimize the approximated Earth-Mover (EM) distance (also called Wasserstein metric), which is used to measure how different the trained model distribution and the real distribution are. WGANs have been demonstrated to achieve more stable training than standard GANs.

At the end of training, the discriminator D is discarded, and the generator G is used to produce new, novel outputs that capture the fundamental properties present in the training examples. The input to G is some fixed-length vector from a latent space (usually sampled from a block-uniform or isotropic Gaussian distribution). For a properly trained GAN, randomly sampling vectors from this space should produce outputs that would be misclassified as examples of the target class with equal likelihood to the true examples. However, even if all GAN outputs are perceived as valid members of the target class, there could still be a wide range of meaningful variation within the class that a human designer would want to select between. A means of searching within the real-valued latent vector space of the GAN would allow a human to find members of the target class that satisfy certain requirements.

2.3 Interactive Evolutionary Computation

In interactive evolutionary computation (IEC) the traditional objective fitness function employed in evolutionary computation is replaced by a human manually selecting the candidates for the next generation [40]. IEC has traditionally been used in optimization tasks of subjective criteria or in open-ended domains where it is difficult to define an objective. Another common area of application

is where optimization is based on subjective aspects of aesthetics and beauty. All of these characteristics apply to content generation for games. IEC has thus been applied to various areas of content generation for games, as diverse as evolving maps [22, 28], weapons [16], and also flowers [31]. However, none of the current IEC for games employ Latent Variable Evolution, which is described next.

2.4 Latent Variable Evolution

The first *latent variable evolution* (LVE) approach was proposed by Bontrager et al. [4]. In their work the authors train a GAN on a set of real fingerprint images and then apply evolutionary search to find latent vectors matching subjects in the dataset.

In another paper Bontrager et al. [3] present an interactive evolutionary system, in which users can evolve the latent vectors for a GAN trained on different classes of objects (e.g. faces or shoes). Because the GAN is trained on a specific target domain, it becomes a compact and robust genotype-to-phenotype mapping (i.e. most produced phenotypes do resemble valid domain artifacts) and users were able to guide evolution towards images that closely resembled given target images. Such target-based evolution has been shown to be challenging with other indirect encodings [47]. In more recent work, Zaltron et al. [50] extended the approach in Bontrager et al. with the ability to freeze and manually edit certain features, allowing users to easily create facial composites.

In video games, LVE has been applied to the first-person shooter Doom using a GAN [11]. An alternative way to induce a latent space for LVE is via an autoencoder, as done in the game Lode Runner [41]. This paper builds on the first application of GANs and LVE to video game levels, specifically for Super Mario Bros. [24, 45, 46]. In fact, this paper uses the publicly available GAN models produced from that research. New GAN models for Zelda are also trained using the same code as the Mario models.

3 VIDEO GAME DOMAINS

The games explored in this paper rely on data from the Video Game Level Corpus (VGLC [39]). Specifically, GAN models for Super Mario Bros. and The Legend of Zelda are used, though in each case some specialized processing of the data is required.

3.1 Super Mario Bros.

Super Mario Bros. (1985) is a classic platform game that involves moving left to right while running and jumping. In order to visualize and play the levels, the Mario AI framework is used[2]. At the start of evaluation, Mario is in his *fire flower* state, which means he can launch fire balls and is the size of two tiles on screen. Contact with an enemy reverts him to *big* state (no more fire balls), and subsequent contact shrinks him to *small* state, which takes up only one tile, meaning that he can access areas that are inaccessible by Mario when large. When Mario is *small*, any further enemy contact causes him to die, ending evaluation.

The VGLC description of Mario levels is tile-based. To use the Mario AI framework, a mapping from VGLC tiles to Mario AI tiles was required (Table 1). Each Mario level from VGLC uses a particular character symbol to represent each possible tile type. This encoding was extended to represent a wider variety of tile types

[2]http://marioai.org/

Table 1: Tile types used in generated Mario levels. The symbol characters (*sym*) come from the modified VGLC encoding, and the numeric identity values (*num*) are mapped to the corresponding values in the Mario AI framework to produce the visualization (*vis*) shown. Numeric identity values are expanded into one-hot vectors when input into the discriminator network during GAN training.

Tile type	sym	num	vis
Stone	X	0	
Breakable	x	1	
Empty (passable)	-	2	
Question Block with coin	q	3	
Question Block with power up	Q	4	
Coin	o	5	
Pipe	t	6	
Piranha Plant Pipe	p	7	
Bullet Bill	b	8	
Goomba	g	9	
Green Koopa	k	10	
Red Koopa	r	11	
Spiny	s	12	

(e.g. different enemies). In order to avoid generating incomplete *pipes*, the VGLC encoding of pipes was modified as well. Instead of using four different tile types for a *pipe*, a single tile is used as an indicator for the presence of a *pipe* and extended automatically as required. A detailed explanation of all modifications made to the encoding can be found in work by Volz [44, Chap. 4.3.3.2].

3.2 The Legend of Zelda

The Legend of Zelda (1986) is an action-adventure dungeon crawler. The main character, Link, explores an overworld to find and explore several maze-like dungeons full of enemies, traps, and puzzles. This game also has a tile-based VGLC description of the dungeons (not the overworld). In this paper, the game is visualized and played with a custom-made, ASCII-based Rogue-like game engine[3], and thus requires a way of mapping the VGLC representation to the Rogue-like representation (Table 2).

In this game, the large set of tiles inherent to Zelda was actually reduced to a smaller set based on functional requirements. Some Zelda tiles differ in purely aesthetic ways, and others rely on complicated mechanics not implemented or even necessary in the Rogue-like. The Rogue-like accommodates three tile types: a floor tile which directly corresponds to the Zelda floor tile, an impassable tile that corresponds to all impassable tiles in Zelda, and a water tile that corresponds to a semi-passable obstacle.

In Zelda, water is not passable by Link until he has obtained the raft item, and even then he can only pass over a single water tile. This item is implemented in the Rogue-like, and is in each level. Zelda also has some enemies that can pass over water, and others that cannot. In contrast, all Rogue-like enemies can pass over water.

[3]Based off of http://trystans.blogspot.com/

Table 2: Tile types used in generated Zelda rooms. The symbol characters (*sym*) come from the VGLC encoding, and the corresponding tile images from the original Legend of Zelda are also shown (*game*). The diversity of VGLC tile types is mapped down to a smaller set of tiles. The numeric values for GAN training are shown (*num*), and the final tiles depict how they appear in the Rogue-like game engine used in this paper (*rogue*).

Tile type	sym	game	num	rogue
Floor	F		0	
Wall	W		1	
Block	B		1	
Door	D		1	
Stair	S		1	
Monster statue	M		1	
Water	P		2	
Walk-able Water	O		2	
Water Block	I		2	

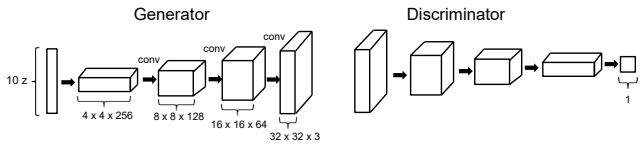


Figure 1: GAN Architecture for Zelda. Mario uses the same architecture, but with a latent vector size of 5 instead of 10, and an output depth of 13 instead of 3.

Enemies are not represented in the VGLC data because the authors neglected to include them[4]. In the Rogue-like, they randomly populate certain rooms. VGLC does include information about doors linking rooms, but that information is excluded from the Rogue-like encoding because door placement is handled not by the GAN, but by a generative graph grammar (Section 4.4).

4 APPROACH

Separate GANs are trained on level data from Mario and Zelda. Interactive evolution is then used to search the latent spaces induced by the GAN models. Other interface features are also described.

4.1 GAN Training Details

The Mario model used in this paper was taken from a publicly available repository associated with previous research [45], but details of its training are repeated here for clarity. The Zelda model used in this paper is new, but it is closely based on another model from a recent paper [15]. Furthermore, the system described in this paper is compatible with all previously published GAN models associated with these papers. All models are Wasserstein GANs [2] differing only in the size of their latent vector inputs (10 for Zelda, 5 for Mario), and the depth of the final output layer (3 for Zelda, 13 for Mario). The Zelda architecture is in Fig. 1. The output depth corresponds to the number of possible tiles for the game. The other output dimensions can match for both games because they are larger than the 2D region needed to render a level segment. During training and generation, the upper left corner of the output region is treated as a generated level, and the rest is ignored.

To encode the levels for training, each tile type is represented by a distinct integer, which is converted to a one-hot encoded vector before being input into the discriminator. The generator also outputs levels represented using the one-hot encoded format, which is then converted back to a collection of integer values. Mario levels in this integer-based format can be sent to the Mario AI framework for rendering, and Zelda rooms in this format can be rendered by the Rogue-like engine. The mapping from VGLC tile types and symbols, to GAN training number codes, and finally to Mario AI/Rogue-like tile visualizations is detailed in Tables 1 and 2.

The GAN input files for Mario were created by processing all 12 *overworld* level files from the VGLC for the original Nintendo game *Super Mario Bros.* Each level file is a plain text file where each line of the file corresponds to a row of tiles in the Mario level. Within a level all rows are of the same length, and each level is 14 tiles high. The GAN expected to always see a rectangular image of the same size, hence each input image was generated by sliding a 28 (wide) × 14 (high) window over the raw level from left to right, one tile at a time. The width of 28 tiles is equal to the width of the screen in Mario. In the input files each tile type is represented by a specific character, which was then mapped to a specific integer in the training images, as listed in Table 1.

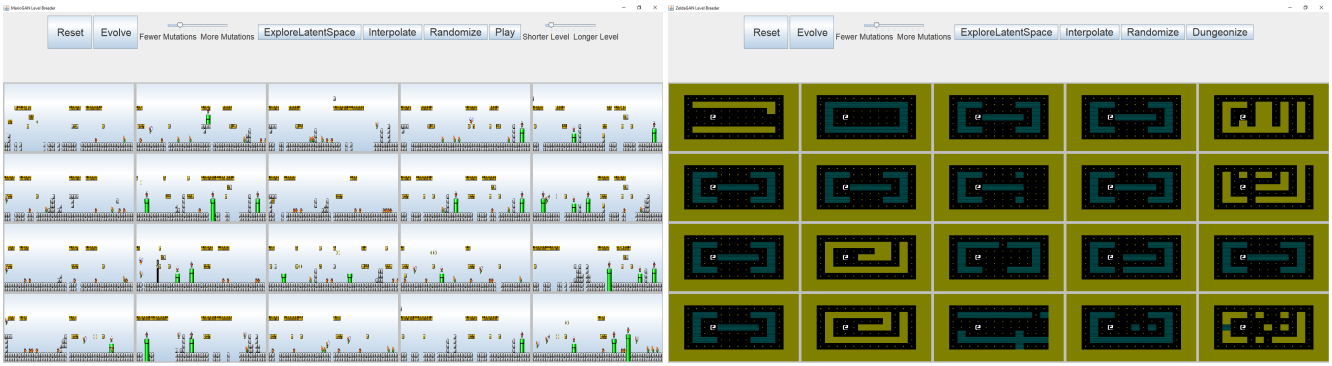
The GAN input for Zelda was created from the 18 dungeon files in VGLC for *The Legend of Zelda*, but the actual training samples are the individual rooms in the dungeons, which are 16 (wide) × 11 (high) tiles in size. Many rooms are repeated within and across dungeons, so only unique rooms were included in the training set, which consisted of only 38 samples. The training samples are simpler than the raw VGLC rooms because the various tile types are reduced to a set of just three as shown in Table 2. The tile set was reduced to focus only on the functional aspects of tiles needed for the Rogue-like engine. Additionally, doors were transformed into walls because door placement is not handled by the GAN, but rather by a generative graph grammar which is used to combine separate GAN-generated rooms into playable dungeons (Section 4.4).

4.2 Interactive Evolution via Selective Breeding

The levels are evolved interactively using an interface similar to Picbreeder [32] and other interactive evolution systems (Fig. 2). A selective breeding algorithm using pure elitist selection is applied. First the user sees $N = 20$ images representing level sketches. Each level sketch is generated by a latent vector with a length dependent on the particular GAN model used to generate it. Complete Mario levels are depicted with the tile graphics from Table 1, and individual Zelda dungeon rooms are depicted with the tiles from Table 2.

The user selects $M < N$ individuals as parents for the next generation. These parents are directly copied to the next generation, and remaining slots are filled by their offspring. Selection continues in this fashion for as long as the user likes. There is a 50% chance of crossover for each offspring (single-point crossover on the vectors of real numbers). Independent of whether an offspring has two

[4]VGLC erroneously refers to impassable statues that occupy some rooms as enemies, but they are simply impassable objects. Other enemies are absent from VGLC



(a) Super Mario Bros. **(b) The Legend of Zelda**

Figure 2: Interactive Evolution Interfaces. Levels can be evolved for (a) Mario or (b) Zelda. Both interfaces display a 5×4 grid of level sketches. Multiple individuals can be selected before clicking the *Evolve* button to create the next generation. Mario levels can have their length adjusted, and can also be played. Zelda individuals are actually only rooms, but selecting multiple rooms and pressing *Dungeonize* generates a randomized dungeon from the rooms, which can then be played.

parents or is a clone, it then has a certain number of chances to mutate defined by a user-controlled slider ranging from 1 to 10. For each mutation chance, each real-valued number in the vector has an independent 30% chance of polynomial mutation [8]. The slider for the number of mutation chances allows users to control the amount of exploration done, by expressing a preference for few or many changes to the genome.

Sometimes, evolution can converge and get stuck in an area of the search space that is hard to escape [12]. This problem is especially common with small populations. One way around premature convergence is to simply restart the evolution process and hope for better results. Therefore, this system provides users with the option to replace selected genomes with new random ones via the *Randomize* button. The fact that only selected individuals are replaced allows users to maintain the best levels they've discovered so far while also looking for potential new starting points for a search. However, users can also explore the search space by changing vectors in other ways.

4.3 Manual Exploration of Latent Space

Users can explore the induced latent vector space more directly with the *Explore Latent Space* option (Fig. 3). This button can be clicked after selecting an individual from the population. A new window pops up with a visualization of the selected level and a number slider for each real number in the genotype. Next to each slider is an input box showing the currently selected value.

Users can change any individual real-valued component of the genotype by either manipulating the slider or entering a new number in the adjacent input box. The visualization updates accordingly. Users can even select a slider and use the left and right arrow keys to watch how the visualization of the level morphs with the changing inputs. Individual tiles gradually swap with others in a relatively smooth fashion, given that every location can only contain one tile. These sliders allow a user to explore the area of latent space surrounding a particular individual. Any modifications during exploration affect the genotype of the individual, thus allowing the modified vector to be selected for further evolution.

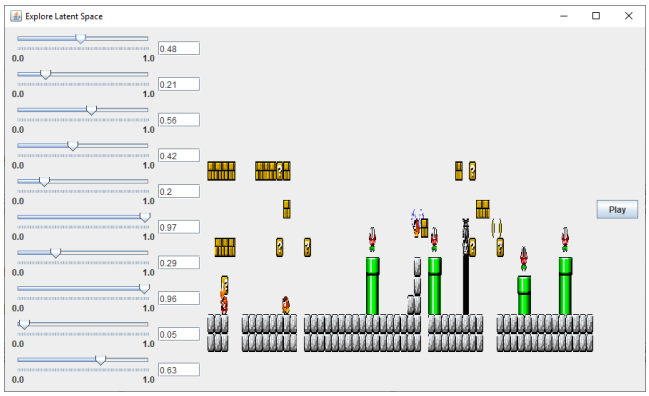


Figure 3: Interface for Manual Exploration of Latent Space. Direct modification of the 10 latent vector values on the left immediately updates the visualization of the Mario level, which can be played by the provided button. The Mario level has 10 latent variables because it consists of two segments, where each is generated by a latent vector of length 5.

Another way to explore the latent space is via the *Interpolate* button. This option requires two individuals to be selected, and opens a window containing one level on the left, and the other on the right. Between the levels is another visualization with a slider above it. The slider determines to what extent the center level is closer in latent space to either the left or right level (Fig. 4). Specifically, the slider traverses a line in high-dimensional space connecting the left level's vector to the right level's vector. Each increment along the slider moves each individual numerical component of the center vector closer to the corresponding component in either the left or right vector. The center level updates as the slider moves, and allows users to find a level that blends aspects of two other levels. In order to bring the newly created level into the population, it must replace either the left or right extreme, and the interface has buttons allowing either of the two levels to be replaced.

Exploring the search space with interpolation and direct modification of individual values allows for finer control of the generated

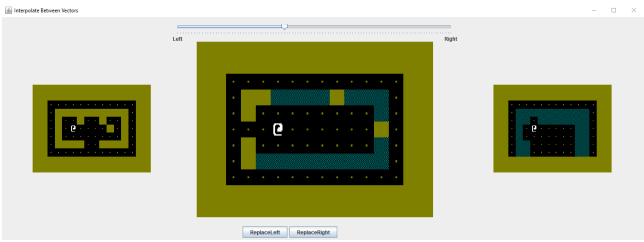


Figure 4: Interface for Interpolation Between Latent Vectors. The slider at the top adjusts the large Zelda room in the middle to be at different positions along the line in latent space connecting the latent vectors for the rooms on the left and right.

levels, but the various latent dimensions seldom have a direct relation to particular level components. Simply picking what looks appealing and evolving it can sometimes be more straightforward. However the space is searched, a way to play the levels is needed to properly evaluate them.

4.4 Interaction With Generated Content

Users can interact with content generated for each game.

Because a single latent vector represents a whole Mario level, the user can select a level and then click *Play* to launch the level in the Mario AI framework. Users can also modify the number of segments in the level. If the number is increased, then the underlying latent vector genome is extended by repeatedly copying the original latent vector values. Further evolution or direct manipulation can differentiate these sub-vectors from each other. To convert a lengthened vector into a level, it is split into sub-vectors of the appropriate length, which are sent to the GAN individually to create segments that are concatenated into one big level.

Latent vectors for Zelda models represent individual rooms, but Zelda dungeons consist of multiple rooms. Instead of a *Play* button, there is a *Dungeonize* button. The user first selects several rooms, and then clicks *Dungeonize* to see a preview of a dungeon generated from the selected rooms (Fig. 5). These dungeons are generated by a graph grammar [9]. Full details of how to combine a GAN with a graph grammar to generate dungeon levels for Zelda are discussed elsewhere [15], but some key points are mentioned here.

A graph grammar takes an initial *backbone* graph containing non-terminal symbols. An iterative process gradually replaces each symbol with sub-graphs until a graph with only terminal symbols remains. Each terminal symbol represents an individual room, and the rules of the grammar place various types of interesting challenges in the rooms: enemies, locked doors, keys for the locks, puzzle rooms, etc. This process also determines whether rooms are connected by a door, and the type of the door (normal, locked, puzzle-locked). However, the grammar does not define the layout of the individual rooms.

When mapping the final graph of terminal symbols to a 2D layout of rooms, specific rooms are randomly chosen from the group selected by the user before clicking *Dungeonize*. The number of rooms in the dungeon is determined by the graph grammar and random chance, so user-selected rooms may appear multiple times, or not at all (unlikely unless a large number of rooms is selected). If the user does not like the resulting dungeon, then clicking the

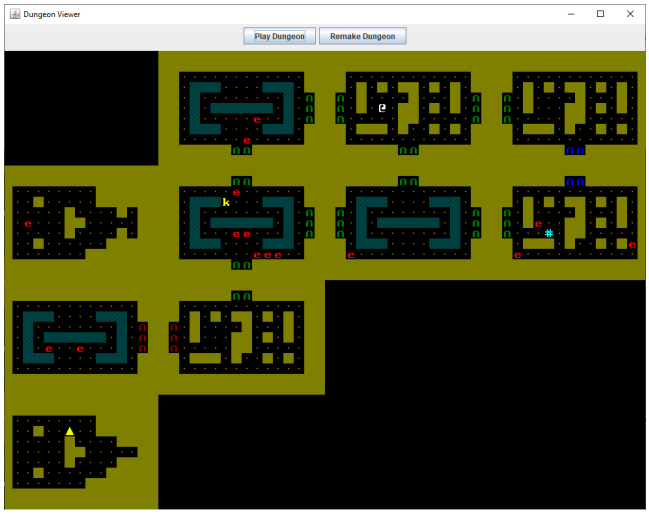


Figure 5: Dungeon Generated By Graph Grammar From GAN Rooms. This dungeon was generated by selecting three rooms and clicking *Dungeonize*. Its layout is random, but based on a graph grammar. The user-selected rooms are randomly placed. There are several symbols: @ = player avatar, e = enemy, k = key, # = raft, and △ = Triforce/goal. Doors can be green (normal), red (locked), or blue (opens after enemies are defeated). Some of the walls are bombable, but these are not marked. The user can either play this dungeon, or choose to generate a new one using the same rooms.

Remake Dungeon button will generate a new dungeon with the same user-selected rooms. Generated dungeons can be played using a game engine for Rogue-like style games that uses simple ASCII graphics. This Rogue-like engine is also described elsewhere [15].

5 HUMAN SUBJECT STUDY

A small user study was conducted to gauge user preference for interactive evolution vs. direct latent space exploration. User quotes also give insight into the strengths and weaknesses of the system.

5.1 Procedure

The software was tested by convenience sampling 22 individuals in China and Denmark with varying game-playing experience. Participants were instructed to design the "coolest" Mario/Zelda level possible using the available options. Each user had their own subjective definition of what "cool" means. Participants were invited to evaluate three variations of the tool using the procedure below:

(1) (design and play) Design levels using only evolution and randomize options. (5 minutes)
(2) (design and play) Design levels using only exploration of latent space, interpolate, and randomize options. (5 minutes)
(3) (survey) Indicate which method led to cooler levels, and explain why.
(4) (design and play) Design levels with all options: evolution, exploration, interpolate, and randomize. (5 minutes)
(5) (survey) Indicate if combining evolution, explore, interpolate, and randomize leads to even better levels, and explain why.

Table 3: Human Subject Study Results. The top two rows show preferences in the first survey, after using the evolution and exploration systems. Note that one Mario user neglected to answer this question. The lower three rows show user perception of the combined system. The main result is that the study participants sligthly preferred the exploration interface over the pure evolution interface and preferred the combination of both the most.

	Mario	Zelda	Total
Prefers Evolution	3 (33.33%)	5 (41.67%)	8 (38.10%)
Prefers Exploration	**6** (66.67%)	**7** (58.33%)	**13** (61.90%)
Combination Worse	1 (10.00%)	0 (0.00%)	1 (4.55%)
Combination Equal	3 (30.00%)	4 (33.33%)	7 (31.82%)
Combination Better	**6** (60.00%)	**8** (66.67%)	**14** (63.64%)

To avoid ordering effects, the order of the first two sessions alternated for different subjects. This study gives insight into what the system is capable of, and which aspects of it should be further developed in the future.

5.2 User Responses

Participant responses are summarized in Table 3. Sample sizes are too small to draw statistically significant conclusions, but results are consistent across the two games. There is a small preference for direct exploration of latent vectors over evolution, but a majority also believes that the combination of approaches is better than either one in isolation. This preference is not surprising, given that extra features generally should not detract from the usefulness of the system.

Only one individual produced worse levels with the combined system than with their individual preference, which was direct exploration. Those that thought the combined system was only as good as their individual favorite seemed to be primarily focusing on the approach they liked more. One user that preferred evolution said, "I ended up focusing on evolution and randomization only [because] I was not aware how changing the numbers [could] improve my level and clicking two buttons (evolve, random) somehow [gave] me interesting levels." Recall that the *Randomize* button was available in all interfaces. A user that preferred exploration stated, "I didn't feel [that] evolution added much, and the explore latent space [option] was the more interesting method. [Combining methods] didn't negatively affect the process, but it wasn't better than the previous version."

However, some individuals found that combining evolution and exploration opened up new possibilities, and gave quotes like, "I was able to bootstrap the characteristics I wanted the level to have by fine-tuning two [levels'] latent vectors. After that, evolution 'was aware' that I wanted e.g. more green pipes," and "It takes the advantages of the previous two [systems] and it added the [functionality] of generating better levels. The 'evolve' and 'interpolate' [buttons] help to discover better levels." These quotes indicate that a sophisticated user can take full advantage of the system to produce better results, but other quotes indicate that combining these features can be challenging: "It [requires] skill to combine [these] two [methods]. [It] is difficult."

Figure 6: Mario Level Generated By Study Participant. To pass the area marked by the red rectangle, Mario needs to be in his *small* state, which the user found to be an unusual limitation.

Some user complaints had more to do with the GAN or other aspects of level generation than with the particular exploration method. Users were surprised that some levels were simply not beatable, or perhaps not beatable without unusual behavior. For example, in some of the generated Super Mario levels, the only way to pass is to injure Mario so that he shrinks, allowing Mario to pass through tight spaces (Fig. 6). A similar issue that affected some Zelda levels is that enemies and items will occasionally appear in unreachable areas of rooms. Even when such placement does not affect whether the level is beatable, users typically find it unusual. Users also noted that isolated water or wall tiles could be generated, which did not seem very useful. However, some rooms in the original game are empty except for two standalone blocks, so such generated rooms do not differ greatly from levels in the original game.

Many participants commented that it would be more explainable and easier to design levels if the effects of changes to the latent vector were correlated with particular level features. One said, "The [system] using explore latent space is not explainable and quite random when changing the values of [a] latent vector." Another said, "It was, however, quite hard to go in a direction, like 'I want more enemies'." The tile-based nature of the GAN output probably makes working toward a specific goal difficult. Enemies and other tiles suddenly appear from nothing as latent value sliders are manipulated.

Some of the participants expected the functionality of direct editing of blocks, via selecting a tile and then changing it to an object (e.g. brick or empty) to fix or improve a generated level. For instance, this functionality would be useful to fix the isolated wall issue mentioned previously.

Despite these minor complaints, the users seemed to enjoy the experience. Participants described their levels with words such as "interesting," "exciting," "diversity," and "innovation." The fact that they were able to discover such levels within short five-minute sessions indicates that the GAN helped them focus on levels within a high-quality region of design space.

6 DISCUSSION AND FUTURE WORK

The combination of interactive evolution with latent variable evolution makes it easy to explore a space of high-quality solutions. Furthermore, the discrete space and small size of the tile-based levels makes use of small latent vectors reasonable. In turn, this makes exploration via direct manipulation of latent vector values manageable. Exploring latent space in this way is particularly useful for video game levels, because a user's personal preferences may not be easily quantifiable as an objective function. In fact, in a single session a user's goals may shift as different parts of the latent space are explored. This latent exploration combined with the ability to play the levels to test them out makes the search process itself an entertaining game experience.

Although this interactive search tool allows users to gain a better understanding of latent level design space, it lacks some practical features that would make it more useful to actual level designers. The most obvious feature to add is the ability to directly edit levels by assigning particular tiles to particular locations in a level. If this feature were added, then designers could explore latent space until they stumbled across a level that is close to what they want, and then make small tweaks to perfect it.

Such direct edits to the level could even be incorporated back into the genome. One option to do this is by using a hybrid encoding [17]: genomes would consist of both a latent vector (indirect encoding) and a vector with one slot for every tile in the level (direct encoding). The direct part could act as a filter, overriding the GAN output where specified. These edits could remain fixed, or also be subject to further evolution. Instead of a hybrid encoding, explicit tile edits could also be recorded by searching for a latent vector that matches the edited level (as closely as possible).

Another adjustment that would make the tool more usable is if the latent dimensions actually corresponded to level features meaningful to a human user. Although adjusting the latent dimension sliders results in small local changes, those changes can introduce totally unexpected features, such as changing a staircase of blocks into pipes in Mario, or a maze of water into a thick wall in Zelda. If specific sliders affected specific portions of the level, or directly corresponded to the presence of certain features (e.g. number of pipes, denseness of water tiles), then humans could make easy use of them. Such control over latent space could be possible if the GAN had either a special architecture or training regime that made certain latent dimensions more responsible for training loss associated with certain output regions or tile types. In addition, it could be useful to integrate related work on scaling the response of control parameters for content generation in order to relate better to user expectations [6].

Another adjustment to the tool that could improve usefulness would be to extend it towards a mixed-initiative system [22] or a system in which IEC is combined with novelty search [20] and objective-based search [23, 48]. This approach could allow users to quickly move toward different regions of the search space without having to manually lead the system there.

Adding most of these practical features to this particular system should be straightforward. Once the system is enhanced, a larger human subject study with a broader sample of participants can be conducted to see if human evaluation reveals statistically significant impacts from particular interface components.

Further development of this system also makes sense because of how general it is. The exact same GAN code was used to train models for both Mario and Zelda. The only game-specific knowledge that the system's code needs is awareness of the output dimension for individual training samples, and a way to display and play the generated levels. Therefore, the authors hope that data for other games will be used to train GAN models and evolve new levels using this system.

An additional benefit of providing a human-in-the-loop method of exploring the latent space of GAN-generated content is the great potential of additional insights into the genotype-to-phenotype mapping encoded in GANs. This is an under-explored area, in particular for phenotypes with direct encodings. Data from user-interaction could thus help develop new, better suitable optimization algorithms for these types of problems [44, 45].

7 CONCLUSIONS

As Generative Adversarial Networks (GANs) continue to be applied as an indirect genotype-to-phenotype mapping for evolution, practitioners will need tools for effectively exploring the inscrutable latent spaces they produce. This paper presents a tool for applying interactive latent variable evolution to GAN models that produce video game levels. This allows users to quickly explore level design space without any a priori objectives in mind. The tool also provides users with more direct ways of controlling the output, namely the features to directly edit latent variables, and to interpolate between points in latent space. The tool is evaluated based on its application to level generation for Super Mario Bros. and The Legend of Zelda. A majority of participants in a human subject study found that the combination of these features was more powerful than either one in isolation, and gave feedback that will allow this system to be developed further into a general purpose tool for game level design for other video games.

ACKNOWLEDGMENTS

The authors would like to thank the Schloss Dagstuhl team and the organisers of Dagstuhl Seminars 17471 and 19511 for hosting productive seminars. This work was supported by the National Natural Science Foundation of China (Grant No. 61906083), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. JCYJ20190809121403553) and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

REFERENCES

[1] David Adams. 2002. *Automatic generation of dungeons for computer games.* Technical Report. University of Sheffield. Bachelor thesis.
[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning.*
[3] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep Interactive Evolution. In *European Conference on the Applications of Evolutionary Computation.*
[4] Philip Bontrager, Julian Togelius, and Nasir Memon. 2017. DeepMasterPrint: Generating Fingerprints for Presentation Attacks. (2017). arXiv:cs.CV/1705.07386
[5] Nathan Brewer. 2017. Computerized Dungeons and Randomly Generated Worlds: From Rogue to Minecraft [Scanning Our Past]. *Proc. IEEE* 105, 5 (2017), 970–977.
[6] M. Cook, S. Colton, J. Gow, and G. Smith. 2019. General Analytical Techniques For Parameter-Based Procedural Content Generators. In *Conference on Games.* 1–8. https://doi.org/10.1109/CIG.2019.8848024

[7] Barbara De Kegel and Mads Haahr. 2019. Procedural Puzzle Generation: A Survey. *IEEE Transactions on Games* (2019).

[8] Kalyanmoy Deb and Ram Bhushan Agrawal. 1995. Simulated Binary Crossover For Continuous Search Space. *Complex Systems* 9, 2 (1995), 115–148.

[9] Joris Dormans. 2010. Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. In *Procedural Content Generation in Games*.

[10] A. E. Eiben and J. E. Smith. 2015. *Interactive Evolutionary Algorithms*. Springer, Berlin, Heidelberg, 215–222.

[11] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. 2019. Searching the Latent Space of a Generative Adversarial Network to Generate DOOM Levels. In *Conference on Games*. IEEE, 1–8. https://doi.org/10.1109/CIG.2019.8848011

[12] David E Goldberg. 1987. Simple Genetic Algorithms and the Minimal, Deceptive Problem. *Genetic Algorithms and Simulated Annealing* (1987), 74–88.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*. 2672–2680.

[14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*. 5769–5779.

[15] Jake Gutierrez and Jacob Schrum. 2020. Generative Adversarial Network Rooms in Generative Graph Grammar Dungeons for The Legend of Zelda. In *Congress on Evolutionary Computation*. IEEE.

[16] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. 2009. Evolving Content in the Galactic Arms Race Video Game. In *Computational Intelligence and Games*. IEEE, 241–248.

[17] Lucas Helms and Jeff Clune. 2017. Improving HybrID: How to best combine indirect and direct encoding in evolutionary algorithms. *PLOS ONE* 12, 3 (2017).

[18] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications* 9, 1 (2013).

[19] Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. 2016. Autoencoders for Level Generation, Repair, and Recognition. In *ICCC Workshop on Computational Creativity and Games*.

[20] Joel Lehman and Kenneth O. Stanley. 2011. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation* 19, 2 (2011).

[21] Antonios Liapis, Christoffer Holmgård, Georgios N Yannakakis, and Julian Togelius. 2015. Procedural personas as critics for dungeon generation. In *European Conference on the Applications of Evolutionary Computation*. Springer, 331–343.

[22] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-Aided Game Level Authoring. In *Foundations of Digital Games*. 213–220.

[23] Mathias Löwe and Sebastian Risi. 2016. Accelerating the Evolution of Cognitive Behaviors Through Human-computer Collaboration. In *Genetic and Evolutionary Computation Conference*. 133–140.

[24] Simon M. Lucas and Vanessa Volz. 2019. Tile Pattern KL-Divergence for Analysing and Evolving Game Levels. In *Genetic and Evolutionary Computation Conference*. ACM, 170–178. https://doi.org/10.1145/3321707.3321781

[25] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. 2016. Adversarial Autoencoders. In *International Conference on Learning Representations*.

[26] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating Search Spaces by Mapping Elites. (2015). arXiv:cs.AI/1504.04909

[27] Anh Nguyen, Jason Yosinski, Yoshua Bengio, Alexey Dosovitskiy, and Jeff Clune. 2016. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space. (2016). arXiv:cs.CV/1612.00005

[28] Peter Thorup Ølsted, Benjamin Ma, and Sebastian Risi. 2015. Interactive evolution of levels for a competitive multiplayer FPS. In *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1527–1534.

[29] Kyungjin Park, Bradford W. Mott, Wookhee Min, Kristy Elizabeth Boyer, Eric N. Wiebe, and James C. Lester. 2019. Generating Educational Game Levels with Multistep Deep Convolutional Generative Adversarial Networks. In *Conference on Games*. IEEE, 1–8. https://doi.org/10.1109/CIG.2019.8848085

[30] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations*.

[31] Sebastian Risi, Joel Lehman, David B D'Ambrosio, Ryan Hall, and Kenneth O Stanley. 2015. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games* 8, 3 (2015), 244–255.

[32] Jimmy Secretan, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T. Folsom-Kovarik, and Kenneth O. Stanley. 2011. Picbreeder: A Case Study in Collaborative Evolutionary Exploration of Design Space. *Evolutionary Computation* 19, 3 (2011), 373–403.

[33] Noor Shaker, Miguel Nicolau, Georgios N Yannakakis, Julian Togelius, and Michael O'neill. 2012. Evolving Levels for Super Mario Bros Using Grammatical Evolution. In *Computational Intelligence and Games*. IEEE, 304–311.

[34] Noor Shaker, Julian Togelius, and Mark J Nelson. 2016. *Procedural Content Generation in Games*. Springer.

[35] Noor Shaker, Julian Togelius, Georgios N. Yannakakis, Ben George Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter A. Mawhorter, Glen Takahashi, Gillian Smith, and Robin Baumgarten. 2011. The 2010 Mario AI Championship: Level Generation Track. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 4 (2011), 332–347.

[36] Adam Summerville and Michael Mateas. 2016. Super Mario as a String: Platformer Level Generation via LSTMs. In *International Joint Conference of DiGRA and FDG*.

[37] 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. *IEEE Transactions on Games* 10, 3 (2018), 257–270.

[38] Adam J Summerville, Morteza Behrooz, Michael Mateas, and Arnav Jhala. 2015. The Learning of Zelda: Data-driven Learning of Level Topology. In *FDG Workshop on Procedural Content Generation in Games*.

[39] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón. 2016. The VGLC: The Video Game Level Corpus. In *Procedural Content Generation in Games*. ACM.

[40] Hideyuki Takagi. 2001. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE* 89, 9 (2001).

[41] Sarjak Thakkar, Changxing Cao, Lifan Wang, Tae Jong Choi, and Julian Togelius. 2019. Autoencoder and Evolutionary Algorithm for Level Generation in Lode Runner. In *Conference on Games*. IEEE, 1–4. https://doi.org/10.1109/CIG.2019.8848076

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

[43] R. van der Linden, R. Lopes, and R. Bidarra. 2014. Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 1 (March 2014), 78–89. https://doi.org/10.1109/TCIAIG.2013.2290371

[44] Vanessa Volz. 2019. *Uncertainty Handling in Surrogate Assisted Optimisation of Games*. Ph.D. Dissertation. TU Dortmund University, Germany.

[45] Vanessa Volz, Boris Naujoks, Pascal Kerschke, and Tea Tušar. 2019. Single- and Multi-Objective Game-Benchmark for Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference*. ACM, 647–655. https://doi.org/10.1145/3321707.3321805

[46] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam M. Smith, and Sebastian Risi. 2018. Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. In *Genetic and Evolutionary Computation Conference*. ACM. http://hdl.handle.net/11214/217

[47] Brian G Woolley and Kenneth O Stanley. 2011. On the deleterious effects of a priori objectives on evolution and representation. In *Genetic and Evolutionary Computation Conference*. ACM, 957–964.

[48] Brian G Woolley and Kenneth O Stanley. 2014. A Novel Human-Computer Collaboration: Combining Novelty Search With Interactive Evolution. In *Genetic and Evolutionary Computation Conference*. 233–240.

[49] Georgios N. Yannakakis and Julian Togelius. 2018. *Artificial Intelligence and Games*. Springer. http://gameaibook.org.

[50] Nicola Zaltron, Luisa Zurlo, and Sebastian Risi. 2020. CG-GAN: An Interactive Evolutionary GAN-based Approach for Facial Composite Generation. *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.