An Evolutionary Approach to Metroidvania Videogame Design

Álvaro Gutiérrez Rodríguez, Carlos Cotta, Antonio J. Fernández Leiva
ETSI Informática, Campus de Teatinos, Universidad de Malaga, 29071 Málaga – Spain
Email:{alvarogr,ccottap,afdez}@lcc.uma.es

***Abstract*—Game design is a fundamental and critical part of the videogame development process, demanding a high cost in terms of time and effort from the team of designers. The availability of tools for assisting in this task is therefore of the foremost interest. These can not just speed up the process and reduce costs, but also improve the overall quality of the results by providing useful suggestions and hints. A conceptual system to approach the construction of this kind of tools is presented in this work. By using a learning component, the preferences and expertise of the designers can be modelled and to some extent simulated. This model is subsequently exploited by an optimization component that tries to create adequate game designs. A proof of concept of the system is provided in the context of level design in Metroidvania games. It is shown that the system can produce quality solutions and hints to the designer.**

I. Introduction

The development of a videogame encompasses different stages/phases and usually involves teams specialized on particular areas. Among all these phases, the design phase is crucial for the ultimate fate of the game: it is in this stage where it is decided how the game will be, what the requirements for subsequent development phases will be, and most importantly, which the source of fun in the game will be.

Unlike other development stages, design is not so commonly assisted by AI tools. This contrasts with the pervasive use of such tools in content generation. For example, Non-Player Characters (NPCs) are agents controlled by the computer whose behaviour must be believable (i.e., in accordance with the rol of that character) to keep the player's inmersion in the game. Developers also use procedural generation techniques to generate new content (e.g., maps[1], weapons[2], stories[3], etc.) during gameplay in order to diminish monotony. The use of content generation tools reduces the workload of designers and artists, and produce results that are generally well-received by the end users.

As anticipated before, the field of intelligent tools for game design is still nascent, but it does not mean that it has not yet been explored [1]–[6]. Indeed, previous works provide interesting and explanatory results, paving the way for further developments. While some of them focus on specific videogame genres and in stablished mechanics, a broader perspective is possible. In this sense, this work is directed to propose a tool aimed to create a complete game (leaving art and sound aside) for any genre, creating the mechanics, game rules, game elements, NPC behaviors and levels. For this, we propose the use of bioinspired algorithms for learning and optimization. More precisely, we pose the use of evolutionary algorithms to generate game contents (game rules, mechanics, etc.) and machine learning tools such as artificial neural networks to mimic the way the designer thinks. After briefly outlining others related works in Section II, this system is described for a general point of view in Section III. As an initial proof of concept, we have picked the case of Metroidvania games [7]. The deployment of the system on this context is then detailed in Section IV, and the results of an empirical examination are provided in Section V. We close this work with a summary of findings and an outline of future work in Section VI.

[1]http://spelunkyworld.com/index.html
[2]https://borderlandsthegame.com/
[3]https://www.shadowofwar.com/es/

II. Background

AI-assisted game design refers to the development of AI-powered tools supporting the game design and development process [6]. Combining these tools with Procedural Content Generation techniques (PCG) is a good approach to aid the game designer.

Liapis *et al.* [4], [8] generate new designs using genetic algorithms (GAs) as a PCG technique. Designers have a map design on which they are working, and the application derives new designs using this former as a seed. The GA uses different criteria to evolve solutions, such as game pace and player balance (see also [9]). The playability of the design is defined by considering if all resources and bases are reachable from any other base or resource. The application shows twelve derivations, six made with the evolution system commented and other six with novelty search. An important part of the experiments involved having professional designers use the tool. The feedback was positive, indicating the system was capable of providing interesting suggestions.

ANGELINA is a cooperative coevolutionary system for automating the process of videogame design. There have been several different versions of ANGELINA in the past [10]–[12]. Focusing on the last version [5], it can create every content of a game: mechanics, game rules, programming, sound, art and levels. The creation of a game starts with a semantic derivation of a sentence or word; after this derivation, free contents (e.g., models, textures, sounds, etc) in agreement with the derivation are searched in the web. Later, mechanics, game

rules and, finally, levels in agreement with those mechanics and game rules are created. During the process, a game is represented as a map which defines passable and non-passable areas in a two-dimensional grid, a layout which describes the arrangement of player and non-player characters in the map, and a ruleset which describes the effects of collisions between game entities, as well as movement types for the NPCs, and time and score limits for the game [12]. A collection of EAs run concurrently, each of them aimed to optimize a different component; in order to have a complete vision and optimize their individual objectives, they share information on the game (board state, rules, mechanics) during the evolutionary process and use the fitness of individual components to increase the overall fitness of the finished artifact[4].

III. An Automated System for Game Design

The design process is difficult due to the different abstract facets it encompasses. Consider for example the design process of a character: the designer receives the story of the character and some features of their personality (e.g., heroic, brave, etc.); then, the designer creates several different designs, all of which can be functional, well drawn, and using an appropriate color palette. But which of them is the best design? The designer could arrange all different designs in a screenshot of the movie or videogame to see which one fits better but how is this decision taken? Designers need to use their experience and creativity to choose the fittest design.

If we now think about level design in a Metroidvania videogame, the main issue is the same: designers knows what kind of experience they want to create with certain mechanics, game rules and level elements specified but, what is the best combination and order of all the elements? Some designs could be created, tested and then the most convincing design (according to the designer's creative mind and subjective opinion) could be refined.

The examples above tackle different elements of a game but they are solved in the same way: through the designer's expertise. In both cases the different designs are evaluated by the mind of the designer, so we need to recreate this cognitive process. To this end, we propose a framework that orchestrates the use of bioinspired algorithms for learning and optimization – see Fig. 1. A typical configuration would involve the use of artificial neural networks (ANNs) for learning and evolutionary algorithms (EAs) for optimization.

Within this framework, EAs will be used to recreate the different ideas that a designer can have during the design process. This is done by trying to optimize an objective function that mimics the designer's goals and preferences. This function is provided by the learning component of the framework. This component is initially seeded with a collection of learning cases, namely examples that can be either positive (goals, appealing features of solutions, etc.) or negative (undesirable features of solutions, traps to be avoided, etc.). Using these, a first model of the designer is built and used by the EA to generate potentially admissible solutions.

[4]Two games produced by this system were presented in the Ludumdare 28 GameJam – http://ldjam.com/ .

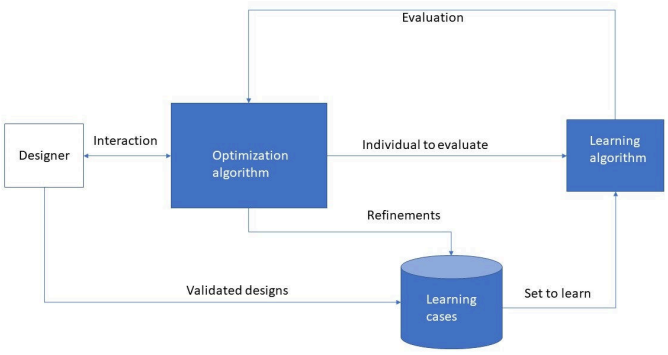


Figure 1: System scheme

During the process, the designer can interact with the system by providing additional examples or clues to refine the model and hence bias the search process in specific directions.

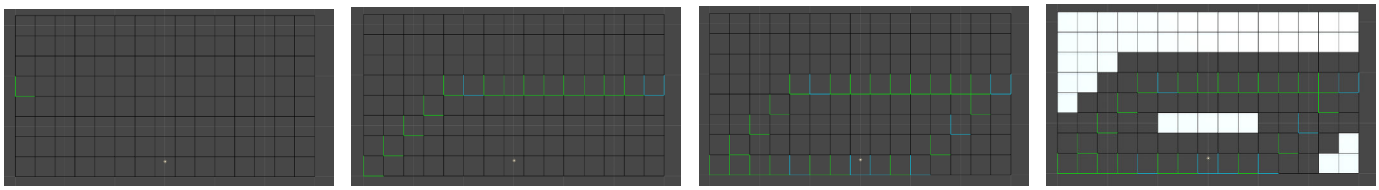
In line with the work by Sorenson and Pasquier [13], in which the fitness function calculate the feasibility and fun of the design, we aim to customize this function to each designer. To this end, the learning component needs to learn to think like the designer and will validate the different individuals (ideas) of the EA like the designer would. A system such as this one can be used to create a complete game of any genre. As an initial step, in this work we have focused on Metroidvania games, whose complexity is more amenable for a study under controlled conditions. This is detailed in next section.

IV. A Case Study: Metroidvania Game Design

When a game designer starts the design of a new game, one of the starting points is selecting the genre of the game. Each genre has several predefined mechanics that define it. When the designer select the genre, new mechanics are created and mixed. In our case, we focus on the Metroidvania genre [14] due to its diversity of mechanics. This genre is famous for mixing the Metroid and Castlevania games series. Games in this genre feature a large interconnected map through which the player can move, having to obtain objects, weapons or abilities to unlock the different locked areas. The map is composed of different areas, each of which is in turn composed of different rooms (including secret rooms). Rooms are where the different enemies, objects, new abilities, are placed.

Typically, a Metroidvania game is a side-scrolling platform game whose creation can be accomplished by adequately defining the following elements [7]:

- **Mechanics:** all actions that the player or character can do, changing the state of the game as a consequence.
- **Game rules:** these indicate the results of an action, i.e., how an action has modified the state of the game. Rules are composed by a set of previous states of game elements, a set of actions and a set of new states (not necessarily the states of the elements involved in the action). For example: stateOf(boxA) + action(moveBoxAToPositionX) = stateOf(doorA, open).
- **Level elements:** all objects included in a certain level (phase, stage, or screen) of the game. We distinguish



(a) (b) (c) (d)

Figure 2: (a) Space partition (b) Main path of the level (c) Joining the main path and the secondary one (d) Complete design of the level.

between two types: interactive and non-interactive. Interactive elements are those whose state can be changed by an action. Conversely, non-interactive elements are those whose state is not affected by player's actions.

- **Objects:** every game object that the player collect, changing the player's state as a result. Such a modification will be defined by the game rules, and can be a new ability or updated stats, just to give two examples.
- **Characters:** these are agents controlled by the computer whose goals are defined by the game rules and whose behaviors are defined by game mechanics.
- **Levels:** This is the scenario in which players use their mechanics and the different enemies, objects and level elements created for the game are placed. The game can have one or several levels but the goal in all of them is defined by the game rules.

We are specifically going to focus on the generation of levels. To do so, for each level to be created we pick a width and height, and generate an empty space of these dimensions, discretized into cells whose size doubles the player's size. Each cell is a possible point of the path to complete the level. Once we have this grid of cells, the designer (or the automated system) creates the path and the optimization algorithm finds a combination of mechanics to use in the so-defined path. The cells that are not part of the path are merely "atrezzo" of the level (e.g., solid walls, pipings, skeletons, etc.). See Fig. 2 for an example of a level created by the system.

The path was calculated by choosing two random points on the left and right sides of the grid and connecting them. Subsequently, a secondary path is created using the same procedure, picking two randoms points in the grid at Manhattan distance $\geqslant 2$ of the main path.

A. Learning process

As stated in the previous section, the learning component is responsible of modelling and simulating the cognitive process of the designer regarding how good a particular level design is. In this work, we have opted for using a neural network for this purpose, although arguably other machine learning methods could be used as well. The role of this component is to serve as a judge capable of assessing the creations of the optimization component in the same way that the designer would do. To this end, we need a reference set composed of good and bad solutions, as dictated by the preferences/knowledge of the designer. We shall use in the following the notation Ref_{good} and Ref_{bad} to refer to the collection of good and bad solutions

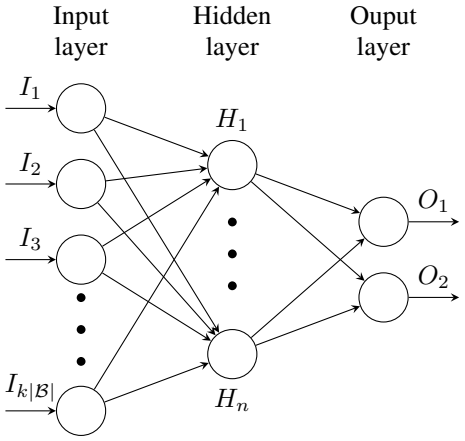


Figure 3: ANN structure used to evaluate motifs

respectively. This is the initial input to the system, and will be used the get the ball rolling in the automated design process.

In order to model the designer's preferences, the next step is extracting the necessary information from the reference sets. For this purpose, the solutions (i.e., game designs) contained in these are scanned in order to identify particular motifs, which are subsequently classified as good or bad, depending on their presence in good/bad solutions. Notice that this can lead to contradictory inputs to the classifier, since a given motif can appear in both good or bad solutions. The particular learning algorithm used must therefore be able to discern whether the motif is really a significant indicator of goodness/badness or it is simply an irrelevant piece of information.

In the particular context we are considering in this work, solutions are the description of the main path of the level (the same process can of course be applied to the secondary path). We assume we have a collection $\mathcal{B}$ of possible building blocks for this path, each of them representing the contents of a certain grid cell (i.e., a corridor, a staircase, a moving platform, etc.). The whole path is therefore a sequence of blocks. Let m be the length of this sequence. In order to extract motifs, we have opted for a very simple strategy which amounts to identifying contiguous subsequences of a certain length k. Thus, if we have a solution $[b_1, b_2, \ldots, b_m]$, we can extract $m - k + 1$ motifs from it: $\langle b_1, \ldots, b_k \rangle$, $\langle b_2, \ldots, b_{k+1} \rangle$, up to $\langle b_{m-k+1}, \ldots, b_m \rangle$. This is repeated for each solution in either reference set, and each motif is assigned a label *good/bad* depending on its provenance.

Once the set of motifs is available, it is fed to an artificial neural network such as the one depicted in Fig. 3. The input level has $k|\mathcal{B}|$ neurons, meaning that each motif is encoded as the concatenation of k bitstrings, each of them representing a block in the motif. To be precise, each of these bitstrings has as many bits as block types are, and only one bit is set to 1 (the bit corresponding to the block in the corresponding position of the motif). As to the output level, it has two neurons, corresponding to each of two classes considered (*good*/*bad*). After training the ANN, any given motif can be assessed numerically with a value ranging from -1 to 1 by subtracting the output (which is between 0 and 1) of these two output neurons. We use this procedure in order to handle contradictory information, which can be present as stated before.

B. Optimization process

The optimization process has the role of generating tentative solutions. As stated before, each solutions indicates the different mechanics to be used in the main path of the level. We represent them with an integer array of length m. Each position in the array represents a cell in the main path, and each integer value $0, \ldots, |\mathcal{B}| - 1$ indicates a certain type of block. In order to evaluate a given solution, it is scanned and the motifs it contains are extracted, much like it was described in the previous subsection. Given that we are considering substrings of length k as motifs, it is convenient to define a function

$$\mathcal{M}^{(k)} : \mathcal{B}^m \to \mathbb{N}^{|\mathcal{B}|^k} \tag{1}$$

that computes how many times each motif appears in a certain solution (note that each solution contains $m - k + 1$ motifs, some of them possibly repeated, among $|\mathcal{B}|^k$ different potential motifs).

Now, we resort to the assessment of the ANN in order to evaluate a particular collection of motifs. More precisely, let S be a solution (a length-m sequence of blocks). Then, $\mathcal{M}^{(k)}(S)$ would be an array with the frequency of each motif in S. We will use the notation $\mathcal{M}^{(k)}(S)_{i_1 \ldots i_k}$ to refer to the element in this frequency array corresponding to motif $\langle i_1 \ldots i_k \rangle$, i.e., the count number of this specific motif in this solution S. The quality value $f(S)$ of this solution would be then:

$$f(S) = \sum_{i_1 \ldots i_k \in \{1,\ldots,|\mathcal{B}|\}} T^{\text{ANN}}_{i_1 \ldots i_k} \mathcal{M}^{(k)}(S)_{i_1 \ldots i_k} \tag{2}$$

where $T^{\text{ANN}}_{i_1 \ldots i_k}$ is the assessment provided by the ANN for that specific motif $\langle i_1 \ldots i_k \rangle$. Thus, the objective function would compute the sum of the value attributed by the ANN to each motif (which ranges from 1 for very desirable motifs down to -1 for highly undesirable solutions, with all the range of intermediate values for motifs of more or less uncertain desirability) present in the solution, weighted by the corresponding frequency of the motif.

The search engine used is an elitist genetic algorithm (GA) with binary tournament selection, uniform crossover and random mutation. The objective of the search would be providing designers different suggestions that mimic their preferences, which could be in turn refined online by tagging particular solutions as good or bad and retraining the ANN.

V. Experiments and Results

The system described in the previous sections has been put to test in order to obtain a proof of concept of its functioning. The results obtained will be described later on in this section. Previously, let us detail the configuration of the experiments.

A. Experimental setting

The game design task considered in this work consists of constructing the level of a Metroidvania game using a collection $\mathcal{B}$ of 7 different blocks. The length of the main path is in this case $m = 50$, thus resulting in a search space whose size $|\mathcal{B}|^m \geqslant 10^{42}$ makes a brute force exploration be out of question. The motifs considered are substrings of length $k = 3$. Hence, the ANN utilized in the learning phase has an input layer of $k|\mathcal{B}| = 21$ neurons. We have chosen $n = 63$ neurons in the intermediate layer (that is, thrice the size of the input layer), and have 2 output neurons as described in Section IV-A. All neurons have a sigmoid activation function, and the ANN is trained using backpropagation (learning rate $\delta = 0.2$, momentum $\alpha = 0.33$; run until the error is less than 0.1 or a maximum of 100,000 learning epochs is reached). As to the GA utilized in the optimization component has a population size of $\mu = 50$ individuals, crossover probability $p_c = 0.9$, mutation probability $p_m = 1/m = 0.05$, and is run for *maxevals*$= 5000$ evaluations (we created different sets of parameters, tested them with the same training set and those mentioned before obtained the best results).

We have considered three test cases to evaluate the behavior of the system. Each of these test cases is constructed by creating an initial payoff table for each of the motifs. Once adequate reference sets Ref_{bad} and Ref_{good} are defined (by providing solutions that aim to maximize or minimize the resulting sum of payoffs according to the initial table), the whole system tries to (i) discover the usefulness of each motif (which is quantitatively unknown to the system, and can only be inferred via the appearance of these motifs in good or bad solutions) and (ii) construct good solutions by combining appropriately these motifs (not straightforward in general, given the fact that all motifs in a solution are in partial overlap with each other).

The first test case is given by a payoff matrix

$$T_{xyz} = \begin{cases} 1 & (x = y) \vee (x = z) \vee (y = z) \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

i.e., motifs with 3 different elements are undesirable, whereas motifs with 2 or 3 identical elements are good. This is an easy test case that can nevertheless provide interesting information in terms of the bias of the optimizer towards particular directions. The second test case is actually a variant of the previous one, and is given by the following payoff table:

$$T_{xyz} = \begin{cases} 0 & x = y = z \\ 1 & x \neq y \neq z \\ 3 & \text{otherwise} \end{cases} \tag{4}$$

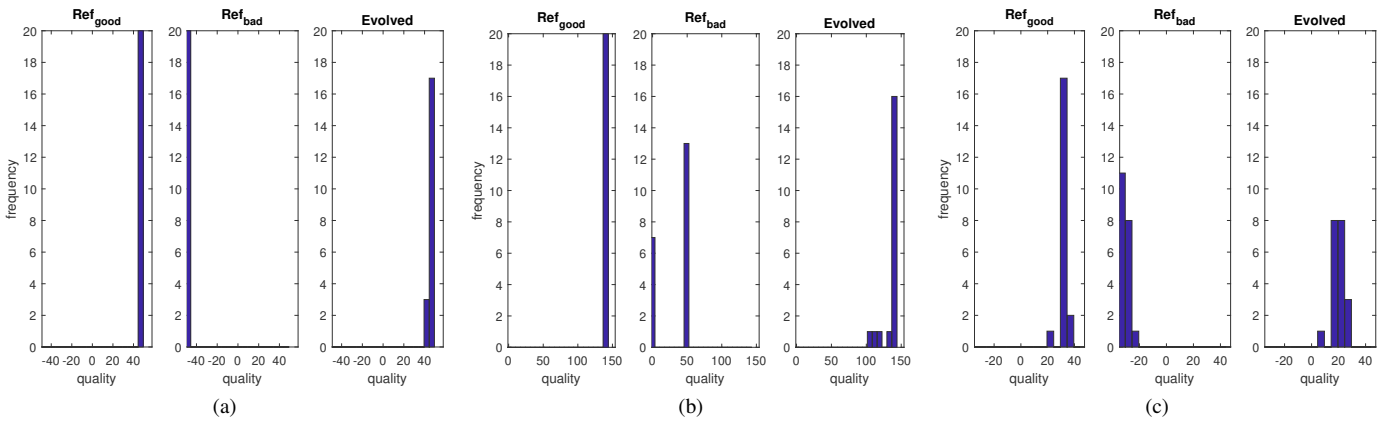


Figure 4: Histogram of objective values according to the underlying payoff table for reference solutions (Ref_{good} and Ref_{bad}) and solutions evolved by the system. (a) Test case #1 (b) Test case #2 (c) Test case #3

In this case, motifs with 3 identical elements are highly undesirable. motifs with 3 different elements are better, yet suboptimal; the best motifs are those with exactly 2 identical elements. Finally, the third test case is defined via random payoffs: T_{xyz} is uniformly drawn at random from the interval $[-1, 1]$ for each particular motif $\langle x, y, z \rangle$.

In the first two test cases, perfect reference sets are constructed by picking at random solutions only comprising good or bad motifs (in the second test case, suboptimal motifs are just included in bad solutions). In the third case, we have resorted to an GA analogous to that used in the optimization component aiming to maximize/minimize the objective function defined by the corresponding payoff table. In all cases, both Ref_{bad} and Ref_{good} comprise 20 solutions each.

For each test case, the ANN has been trained using the motifs extracted from the reference sets and the GA has been run 20 times, keeping the best solution found in each run. All algorithms have been implemented on the videogame engine Unity using C# as programming language. The AForge library[5] has been used to support the ANN.

B. Results

A first glimpse of the experimental results obtained is provided in Fig. 4. Therein, the objective values of both reference solutions and evolved solutions is shown for each test case. As indicated in previous subsection, the reference solutions for the first two test cases are perfect, in the sense they only contain desirable or undesirable motifs. Thus, the mass of objective values is concentrated on the left end of the histogram for bad solutions and on the right end for good solutions. The reference solutions for the third test case were empirically obtained and hence there is more variety of objective values (note that this may to some extent constitute a more realistic setting, in which solutions provided by the designer are not ideal flawless prototypes but can rather have an inherently noisy structure). In either case, notice how the solutions provided by the EA also tend to cluster towards the right end of the histogram, indicating that they are objectively good according to the hidden criterion used for defining goodness/badness.

[5]http://www.aforgenet.com/

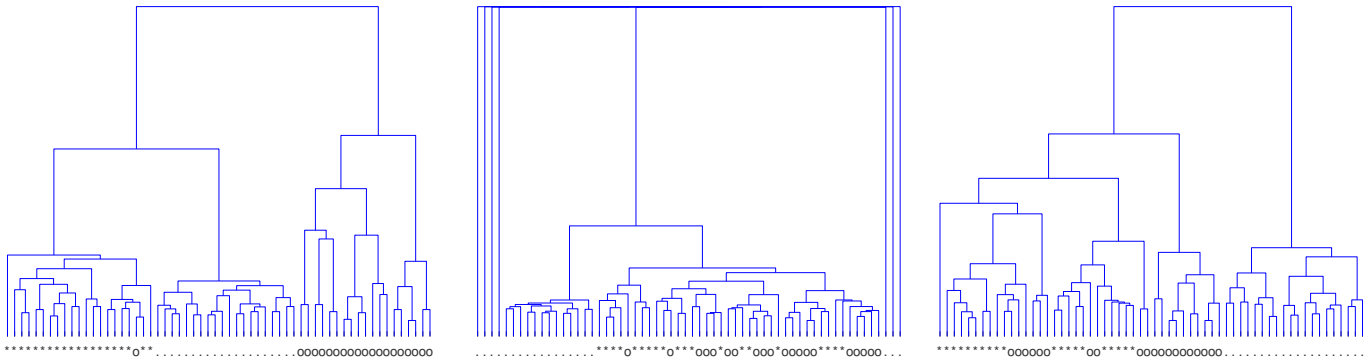
Table I: Structural difference between solutions in the test case #1. The number of motifs of each type (based on the number b of identical blocks in it) is shown for each data set

	b=2	**b=3**	**b=0**
Ref_{good}	897	63	0
Ref_{bad}	0	0	960
Evolved	533	414	13

Some further insight is obtained if we take a look at the structure of evolved solutions and try to compare these with solutions in either reference set. To this end, we compute for each solution S the associated motif-frequency array $\mathcal{M}^{(k)}(S)$. Subsequently, we can compute the structural distance $D(S, S')$ between two solutions S and S' as the Euclidean distance between the corresponding motif-frequency arrays, i.e.,

$$D(S, S') = \sqrt{\sum_{i_1 \ldots i_k} \left(\mathcal{M}^{(k)}(S)_{i_1 \ldots i_k} - \mathcal{M}^{(k)}(S')_{i_1 \ldots i_k}\right)^2} \quad (5)$$

Once this is done for each pair of solutions, the resulting distance matrix can be used to perform a hierarchical cluster analysis. We have done this on the solutions available for each test case, using Ward's minimum variance method to guide the agglomerative clustering process [15]. As can be seen in Fig. 5, in both test cases #2 and #3, evolved solutions have some affinity among them but tend nevertheless to mix with clusters of solutions in Ref_{good}. Together with the good objective quality of these solutions discussed before, this indicates that the system did faithfully capture the features of interest in solutions and could combine them appropriately. Test case #1 is also interesting: evolved solutions are objectively good, but they tend to cluster together, mostly apart from solutions in either reference set. If we perform a quick structural analysis, we obtain the results shown in Table I. Note that a few bad motifs (composed of three different blocks; $b = 0$) have slipped into these solutions, although this is not very significant. It is far more interesting to note that evolved solutions tend to have much more motifs composed of identical blocks. This may be an artifact of the learning process, that



(a) (b) (c)

Figure 5: Clustering of solutions in the reference sets and evolved by the system. A dot ('.') represents solutions in Ref_{bad}, a star ('*') solutions in Ref_{good}, and a circle ('o') evolved solutions. (a) Test case #1 (b) Test case #2 (c) Test case #3

might marginally favor some particular motif, or a byproduct of the search dynamics of the EA, whereby it may be easier for it to create good solutions by exploiting this kind of motifs. Of course, at this point the loop could be closed by having the designer inspect these solutions, possibly providing complementary preferences in order to support or discard this kind of solutions.

VI. Conclusions and Future Work

We have presented the concept of an AI-assisted videogame design system, aimed to help designers create games by suggesting new ideas based on a model of their preferences and knowledge. To this end, we propose the orchestrated use of machine learning techniques and optimization methods, the former to capture the designers' expertise, and the latter to exploit this expertise in a systematic (and hopefully creative) way. The experimental proof of concept has shown that not only quality solutions analogous to those provided as reference can be generated, but also different non-anticipated biases can appear, providing interesting hints to the designer.

As future work, we plan to deploy the system on a more realistic environment to test its capabilities. Needless to say, closing the loop and having the designer introduce dynamic preferences is another improvement of the foremost interest.

Acknowledgments

This work is supported by Spanish Ministerio de Economía, Industria y Competitividad under projects EphemeCH (TIN2014-56494-C4-1-P) and DeepBIO (TIN2017-85727-C4-1-P), and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

[1] R. Hunicke, M. Leblanc, and R. Zubek, "MDA : A formal approach to game design and game research," in *AAAI Workshop on Challenges in Game AI*, vol. 4. San Jose: Press, 2004, pp. 1–5.

[2] N. Sorenson and P. Pasquier, "The evolution of fun: Automatic level design through challenge modeling," in *International Conference on Computational Creativity*, D. Ventura, A. Pease, R. P. y Pérez, G. Ritchie, and T. Veale, Eds. Lisbon, Portugal: Department of Informatics Engineering, University of Coimbra, 2010, pp. 258–267.

[3] A. Liapis, G. N. Yannakakis, and J. Togelius, "Designer modeling for personalized game content creation tools," in *The 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Boston, USA: AI Access Foundation, 2013, pp. 11–16.

[4] ——, "Sentient sketchbook: Computer-aided game level authoring," in *ACM Conference on Foundations of Digital Games*, G. N. Yannakakis, E. Aarseth, K. Jørgensen, and J. C. Lester, Eds. Chania, Crete: Society for the Advancement of the Science of Digital Games, 2013, pp. 213–220.

[5] M. Cook and S. Colton, "Ludus ex machina: Building a 3d game designer that competes alongside humans," in *International Conference on Computational Creativity*, S. Colton, D. Ventura, N. Lavrač, and M. Cook, Eds. Ljubljana, Slovenia: computationalcreativity.net, 2014, pp. 54–62.

[6] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 317–335, 2015.

[7] K. Pearson, "Guide to making metroidvania styled games – part 1," PlatformerPower, 2017, https://platformerpower.com/post/160130225259/guide-to-making-metroidvania-styled-games-part-1 [accessed 28-Jun-18].

[8] A. Liapis, G. N. Yannakakis, and J. Togelius, "Designer modeling for sentient sketchbook," in *2014 IEEE Conference on Computational Intelligence and Games*. Dortmund, Germany: IEEE, 2014, pp. 1–8.

[9] R. Lara-Cabrera, C. Cotta, and A. J. Fernández-Leiva, "On balance and dynamism in procedural content generation with self-adaptive evolutionary algorithms," *Natural Computing*, vol. 13, no. 2, pp. 157–168, Jun 2014.

[10] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, S. Cho, S. M. Lucas, and P. Hingston, Eds. Seoul, South Korea: IEEE, 2011, pp. 289–296.

[11] M. Cook, S. Colton, and A. Pease, "Aesthetic considerations for automated platformer design," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, M. Riedl and G. Sukthankar, Eds. California, USA: The AAAI Press, 2012, pp. 124–129.

[12] M. Cook, S. Colton, and J. Gow, "Initial results from co-operative co-evolution for automated platformer design," in *Applications of Evolutionary Computation – EvoApplications 2012*, ser. Lecture Notes in Computer Science, C. Di Chio *et al.*, Eds., vol. 7248. Berlin, Heidelberg: Springer Verlag, 2012, pp. 194–203.

[13] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *Applications of Evolutionary Computation - EvoApplications 2010*, ser. Lecture Notes in Computer Science, C. Di Chio *et al.*, Eds., vol. 6024. Berlin Heidelberg: Springer Verlag, 2010, pp. 131–140.

[14] C. Nutt, "The undying allure of the metroidvania," Gamasutra, 2015, https://www.gamasutra.com/view/news/236410/The_undying_allure_of_the_Metroidvania.php [accessed 29-Jun-18].

[15] J. H. Ward Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.