Article

Procedural Video Game Scene Generation by Genetic and Neutrosophic WASPAS Algorithms

Aurimas Petrovas and Romualdas Bausys *

Department of Graphical Systems, Vilnius Gediminas Technical University, Sauletekio al. 11, LT-10223 Vilnius, Lithuania; aurimasp.petrovas@vilniustech.lt

* Correspondence: romualdas.bausys@vilniustech.lt

Abstract: The demand for automated game development assistance tools can be fulfilled by computational creativity algorithms. The procedural generation is one of the topics for creative content development. The main procedural generation challenge for game level layout is how to create a diverse set of levels that could match a human-crafted game scene. Our game scene layouts are created randomly and then sculpted using a genetic algorithm. To address the issue of fitness calculation with conflicting criteria, we use weighted aggregated sum product assessment (WASPAS) in a single-valued neutrosophic set environment (SVNS) that models the indeterminacy with truth, intermediacy, and falsehood memberships. Results are presented as an encoded game object grid where each game object type has a specific function. The algorithm creates a diverse set of game scene layouts by combining game rules validation and aesthetic principles. It successfully creates functional aesthetic patterns without specifically defining the shapes of the combination of games' objects.

Keywords: genetic algorithm; procedural generation; game scene; multicriteria decision making; WASPAS-SVNS

Citation: Petrovas, A.; Bausys, R. Procedural Video Game Scene Generation by Genetic and Neutrosophic WASPAS Algorithms. *Appl. Sci.* **2022**, *12*, 772. https://doi.org/10.3390/app12020772

Academic Editor: Rui Araújo

Received: 5 December 2021
Accepted: 7 January 2022
Published: 13 January 2022

1. Introduction

Today, researchers are discovering more and more new results in the artificial intelligence domain [1]. Increasing computing power, storage, and volumes of data creates new approaches to use Artificial Intelligence. Computational Creativity is one of the approaches that is gaining traction. It is used to solve creativity problems and is realized through computation-based systems that attempt to simulate creative work. Creativity definitions vary, and widely agreed upon definitions of creativity in engineering are not defined, but it is necessary to understand the essence of the concept to model a system according to creativity principles. The definition of creativity can be split into a few parts depending on how creative work is rated or created. Usually, there are four types of creativity modeling targets: person, process, product, and press. The most common machine learning targets for creative tasks are product and process [2]. The product target evaluates a completed work and attempts to replicate it by combining and expanding elements of the previous work. The process target tries to simulate logical loops, which are used to create work. The person target is rarely used, as it requires simulation of the creative agents or person. Press target is quite common when trying to filter creative and impactful work (i.e., Internet content scans). The product target is widely used in machine learning tasks, as most training data sets are made up of the available creative work—these systems usually create an independent logic loop of creativity directly unrelated to the original work process [3]. We are focusing on the process-related target, as it usually generates more example-independent results, which is one of computational creativity tasks. This means that the generated work differs more from the training data set.

To understand the structure of creativity, we can break it down into different classifications, which are important to understand when building a model. Creative value can be defined by these key terms: usefulness, aesthetics, originality, relevance to the task, and surprise [4]. These points are often referred to when trying to identify what makes the creative result valuable. Creativity involves a combination of expertise, chance, and intuition; adding these traits to a system generally makes the system more likely for its result to have more creative value [2]. Example approaches to cognitive creativity include concept combination, concept expansion, imagery, metaphor, and divergent thinking [5]. The cognitive approach is often compared to a heuristic search. Evaluation of results is an important task, as it defines creativity. Common evaluation methods are Classification, Regression, Predictive models, Generative models. These methods usually try to replicate creative content rather than exploring new spaces. Transformational creativity systems try to decide what is creative by themselves, autonomously using more abstract evaluation methods. The most effective evaluation is usually outside the system, feedback from other creative systems [6]. Transformational systems are not effectively realized or used today. Common challenges for creativity evaluation can be grouped into two categories: how to generate transformational creative content, which can add new value to existing parts of the results; another common challenge is that generated results may be quite similar between a few iterations of the result running on the same model. There are not that many creativity-oriented models. Creative models can generate artwork but lack contextual creative value [7].

There are a lot of possible criteria in the generative content ranging from functional to aesthetic; however, it is difficult to choose the criteria list for each task, and there is not a lot of research work done on creative fitness modeling. Ratios of selected criteria have a huge impact on the final results, and improper ratios may easily break the final result. Another important choice is the selection of the criteria itself. They can range from low to high levels. The low-level criteria define basic building block rules, and the high-level criteria define more abstract and specific tasks. Lower-level criteria usually increase the variety of generated results, while higher-level criteria can generate a specific result with fewer calculations. It is important to select criteria or create a criteria manipulation system to form a fitness function. There are various ways to realize a model using various algorithmic approaches. We are approaching our problem with a combination of procedural generation and machine learning methods.

One of the methods of content generation problem is procedural game content generation using machine learning models on existing content (PCGML) [8]. The use of procedural content generation is increasing in the game industry, and researchers are trying to find new ways to generate high-quality content. Generation assist levels can be categorized as partial, complete, autonomous, interactive, or guided. Game content is classified as functional and cosmetic. The main problems with the procedural generation machine learning approach include training on small datasets, lack of suitable data, parameter adjustment, and others [9]. Procedural content generation methods (PCG) usually lack evaluation, and objectives are created by designers. Use cases for PCGML are autonomous generation, artificial intelligence-assisted design, repair, analysis, and data compression. The proposed research is focused on autonomous generation, which creates game content without human interaction by combining the algorithm and the fitness function. Video games are a widely used form of multimedia that requires a broad scope of machine learning approaches. Game design generally requires the level of the game to be both playable and aesthetic [10]. At the same time, there is no common way to standardize datasets and evaluate performance for game design problems [8,11,12]. The objectives of PCG in game level generation are to make games more replayable, less demanding for creator time, reduce storage space, or enable particular aesthetics [10].

The fitness function for game design and computational creativity usually contains a subjective combination of criteria and is still in the early research state of its quantification [13]. There are no widely agreed upon definitions of how results should be compared. Game design can be broken into several parts, categorizing games by their objectives. Conversion to fitness criteria varies depending on the type of game. Patterns are elements that are present in levels across multiple games, rather than being a feature that is recurrent on the same game title. Patterns are usually categorized into several types. Guidance guides players in an intended direction. The safe zone is the area where players are not exposed to negative interactions. The foreshadowing hints at something that will happen later. Layering is the pattern of combining multiple objects to create a new experience. Branching provides players with multiple paths to reach their objectives. Pace breaking refers to the purpose of changing elements of the game to achieve a creative objective [14].

Most automated game design approaches follow reverse engineering principles, usually by using datasets generated by analyzing other games. Using this approach, fitness criteria for generations do not have to be defined [15]. The main problem with this approach is the lack of new concepts in the generated content [6]. The goal is to derive objective formulas from game design principles to generate game levels. Game flow strategy is one of the propositions to measure game design in the literature. This concept combines concentration, challenge, player skills, control, clear goals, feedback, immersion, and social interaction criteria [16]. Some authors try to measure game engagement by analyzing difficulty and applying constraints to make levels playable [17]. Current research is trying to quantify abstract creativity criteria so that they can be used in real-world digital applications. There are some examples in the field that use fuzzy logic to express criteria lists [18]. There is also a new emerging usage of neutrosophic sets combined with multicriteria decision making (MCDM), but this approach is not widely explored in the field of machine learning, but it can benefit the creativity of such models [19–22]. There are also not many MCDM algorithms used together with iterative optimization algorithms [23,24]. Some SVNS applications in the literature emphasize a greater focus on uncertainty [25]. In the following paragraphs, we add a more detailed explanation of related work, methodology, created framework, results, and conclusions.

2. Related Work

There is a rising interest in automated game level generation. Machine learning algorithms are masters of specific computational tasks, but there is no perfect solution to mimic human creativity. The primary goal of this type of research is to identify creativity measurements and apply them to automated content generation. The current stage of results in this field is mostly exploratory and does not substitute for creative work in most cases, but it gradually increases assistance levels for the creator by overtaking simple creative tasks. PCGML is one of the assistance tools for work generation. There are 4 modeling steps: problem identification, solution, results, and application of generated results.

There are research examples that tackle the problem of computed creativity in the game design field. One of the examples in the industry is the generation of physical puzzle game levels with the objective of fitness of the feasibility and stability of objects [26]. Final fitness is calculated using an agent that plays the game. This method reduces the computational cost for this problem and adds new solutions to calculate the rewards of the genetic algorithm, not focusing too much on the penalty. Another example is a level generator for a Lode Runner-type game. It assesses playability and connectivity using the ‘A*’ algorithm. Generator uses an autoencoder with a multi-channel approach, analyzes 150 pre-made levels, and uses evolutionary algorithms. Levels are encoded into multichannel strings. This solution adds some unpredictability. Performance evaluation compares similarity to the original game levels [27]. There is also a framework for general 2D games (mostly top-down adventures) [28]. It evaluates levels for symmetry, balance, density, and reachability with a focus on aesthetics and difficulty. For final fitness, it derives 3 different fitness values and calculates the average value (Score Difference Fitness, Unique Rule Fitness, and Metric Based Fitness). It tries to apply the procedural video game generation problem to a variety of games with differing rules. Another example is focused on creative patterns [29]. The match 3 type game is used as a base for the generator. For evaluation, it uses visual pattern recognition and line symmetry. The results are judged by expert study analysis. It learns from existing content and uses pattern-aware PCGML, random Markov fields with symmetric positional information, and visual analysis. This research tries to generate larger structural patterns. The Pac-Man arcade-type game [30] evaluates playability, the spread of objects, ratios, and evolving levels using a genetic algorithm. It tries to generate unique levels with each iteration of the algorithm. Another more generalized example is the generation of verticality for mostly flat surfaces on grid-based surfaces [31].

Most of the examples in the literature uses 2D space for experimentation and games from the 1980s or simple game levels made specifically for the selected tasks. Usual objects for game levels are empty space, wall, player, goal, collectibles, and hazards. General evaluation criteria are guidance, progression, aesthetics, safe zones, and pace breaking. The current state of the computational creativity field in video games is quite young and has not yet been applied to a game structure for large and complex games. It is also difficult to model systems that can fully replicate manual creative work. As the result becomes more complex, it is easier to distinguish synthetic creativity.

3. Scene Layout Modeling and Optimization Algorithm

We propose a PCGML framework for automated game scene layout generation. Our mathematical model consists of the fitness function, which is used by a genetic algorithm to evaluate the population. The MCDM utility function is used as a genetic algorithm fitness function. We chose fixed criteria parameters for difficulty, playability, and size adjustments. One algorithm iteration populates a game level grid, which is also used for further evaluation. Evaluation calculates fitness for each game level grid and selects the best performing grids. This model generates varied and unexpected results because the generation seed is randomly selected and fine-tuned by the algorithm.

We combine the level design criteria measurements into a multi-criteria decision-making table to formulate the problem. The final fitness for the value of the game scene can be measured by combining the scores for each criterion. It can evaluate different generated scenes by using generated alternatives for one table axis and fitness scores for another table axis. Based on the table results, we can then choose which alternatives should be used as a base for further scene generations. By calculating the fitness score for each criterion and combining them into a table, we can assess and evolve a combined fitness score to generate game levels. Proposed research is focused on the process evaluation type, which studies what types of actions are made that make results creative. We combine the criteria values by converting them to fuzzy sets. Our approach is to use the weighted aggregated sum product assessment with a single-valued neutrosophic sets (WASPAS-SVNS) method to find solutions when multiple conflicting criteria are present [19–24]. Calculations are made with fuzzy logic using neutrosophic sets [32].

From a computational creativity standpoint, we are using a creative process approach combining usefulness, aesthetics, and chance to create our model. These aspects form the constraints and criteria set for the mathematical model. This framework designs and generates video game level layouts. It generates random levels, modifies them with a genetic algorithm, and evaluates them with weighted aggregated sum product assessment to find the best alternatives. The framework can also be expanded with additional requirements and fitness criteria, and most of the parameters can be altered to suit needs. We explain our realization in detail next, broken into four chapters: game scene modeling methodology, game scene procedural generation criteria list, proposed extension of genetic algorithm by WASPAS-SVNS, and application of WASPAS-SVNS utility function to calculate fitness function.

3.1. Game Scene Encoding Modeling

A common set of game objects is applied, which is selected based on game level design principles. There are several possible object types encoded in the matrix. Each number represents a different object type. Game scene layout is discretized into a grid, and one object can occupy one cell. The single-scene layout forms a single genetic algorithm chromosome. These are:

- Player (number 0)—represents the starting position of the subject, which is intended to play the game;
- Exit (number 1)—marks the location that the player should reach to finish the game;
- Empty space (number 2)—traversable and empty space, which can be used to navigate by the player;
- Wall (number 3)—object that blocks player navigation;
- Hazzard or enemy (number 4)—a traversable object, which is dangerous for the player;
- Collectible (number 5)—a desirable object that can be collected by the player;
- Ground—this object is not encoded in the chromosome matrix but is used during the 3D projection visualization step as a floor layer.

The data of a single chromosome is stored in a two-dimensional number grid (Figure 1). We use a 10 unit wide and 10 unit long matrix for our experiments. Each object type is encoded as a different number. The final results are projected into the 3D space by adding a ground layer beneath the grid and converting numbers into 3D objects on the main grid.

3	3	3	3	3	3	3	3	3	3
3	0	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	2	3
3	2	2	2	2	2	2	2	1	3
3	3	3	3	3	3	3	3	3	3

Figure 1. Single-chromosome data example.

3.2. Game Scene Procedural Generation Criteria List

After discretization, we choose a set of criteria that define our game layout requirements. We proposed to use 4 fitness criteria functions and 3 constraint functions, which use function results to calculate the total fitness value with the WASPAS-SVNS algorithm for each iteration of the genetic algorithm. Aesthetics are defined by the symmetry and empty-space balance criteria. Usefulness is defined by the safe zone and player exit distance criteria. Criteria were selected based on recurrence in the literature [26–30], game design principles, and creativity definitions [2,4,5]. If one of the constraint functions does not pass, the total fitness is multiplied by zero.

The results of the criteria are normalized to fit in the 0 to 1 range to have a reference point for different criteria metrics [17]. Zero represents the worst possible value, and 1 represents the best possible value. The final values for each criterion are also multiplied by 0.9 so they will not get too close to 1, as it may skew the results in the evaluation steps using neutrosophic sets. Scalar values are converted to single-valued neutrosophic sets during evaluation. The fitness functions are as follows:

- Symmetry calculation for aesthetic purposes. The chromosome grid is crossed with a horizontal and vertical slice to form 4 smaller 5×5 grids. Each object is checked to determine if it has an identical vertically and horizontally symmetrically matching object (Figures 2 and 3) in the 5×5 grid. The final results are calculated by dividing the symmetrical matches by the maximal possible matches (each object has two matching objects with touching 5×5 grids) (1). x and y represent the size of the grid, s is a binary value, the value of which is 0 if the object does not have a matching pair. Each object is measured twice for each axis.

$$m = \frac{\sum_{i=0}^{2xy} s}{2xy} \tag{1}$$

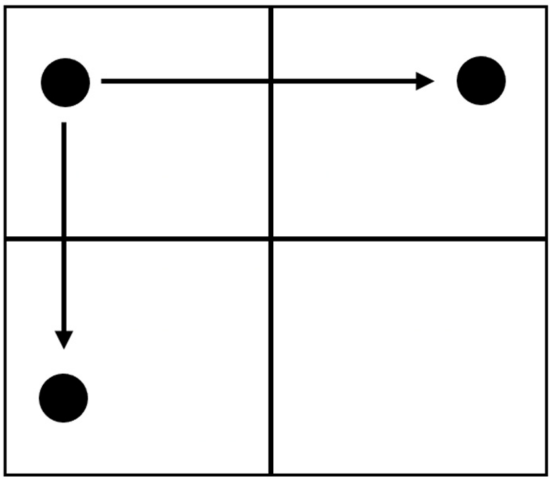


Figure 2. Symmetry calculation.

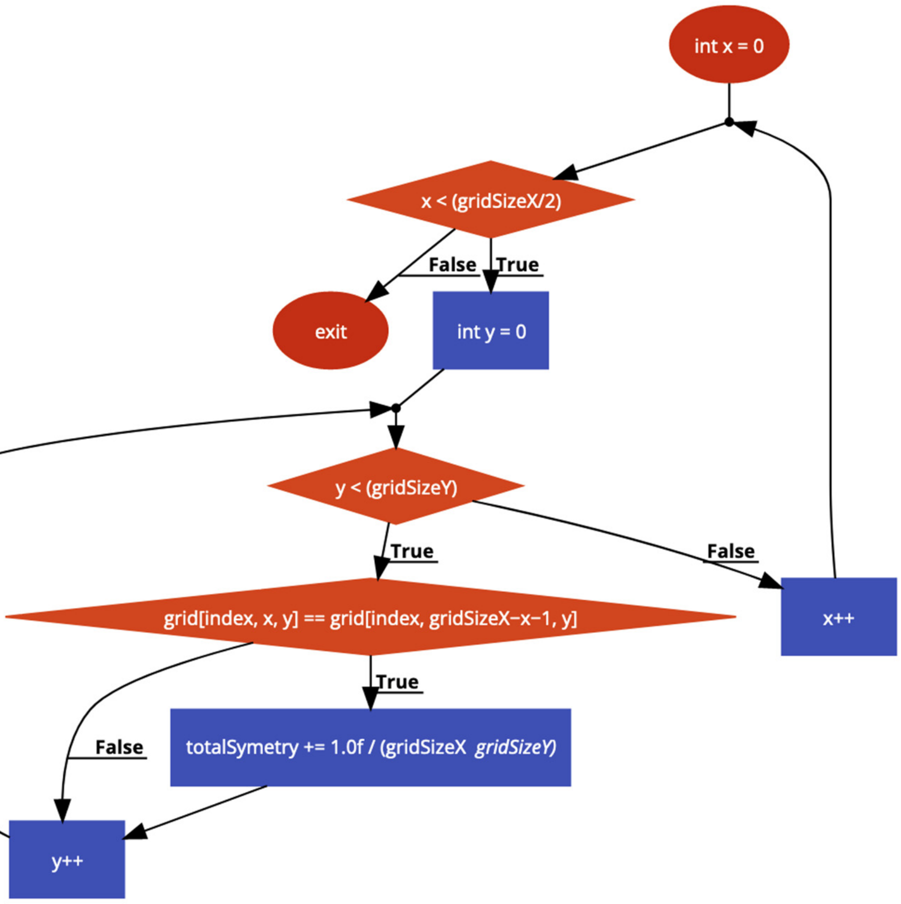

int x = 0
x < (gridSizeX/2)
False
True
exit
int y = 0
y < (gridSizeY)
True
False
x++
grid[index, x, y] == grid[index, gridSizeX−x−1, y]
True
False
totalSymetry += 1.0f / (gridSizeX *gridSizeY)*
y++

Figure 3. Symmetry calculation for single grid axis.

- Balance criteria for aesthetic purposes. Calculate how close to 50% is the ratio between empty game object count and total object count (Figure 4).

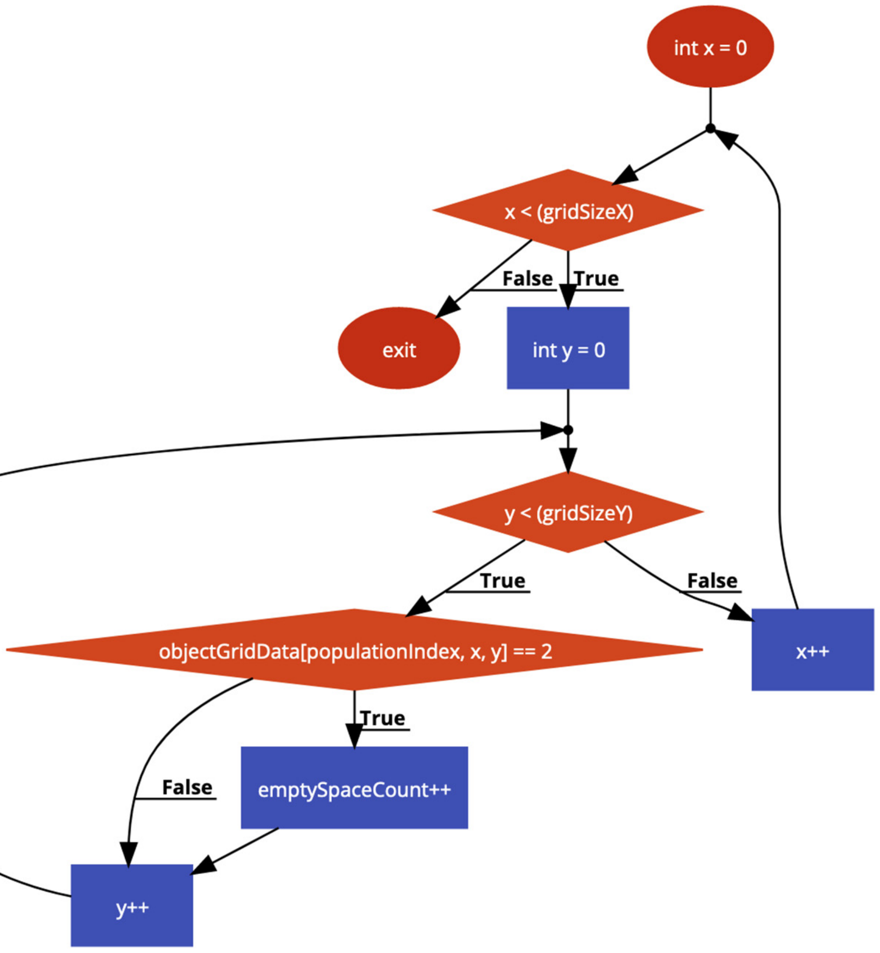


Figure 4. Empty-space balance.

Mathematically, it can be expressed in these steps (2), where e is the total empty space ratio normalized from 0 to 1, t is the sum of the empty objects (they are reversed if they exceed 50% of the grid), x and y represent the grid size, and s is a binary value, which value is 1 if the object is empty. t can be calculated by counting all empty space objects (3) and then reversing the value if it exceeds 50%.

$$e = \frac{t}{\frac{t}{\frac{1}{2}xy}} \tag{2}$$

$$\begin{cases} t = \sum_{i=0}^{n} s \\ t = \frac{1}{2}xy - t_1 - \frac{1}{2}xy \end{cases} \tag{3}$$

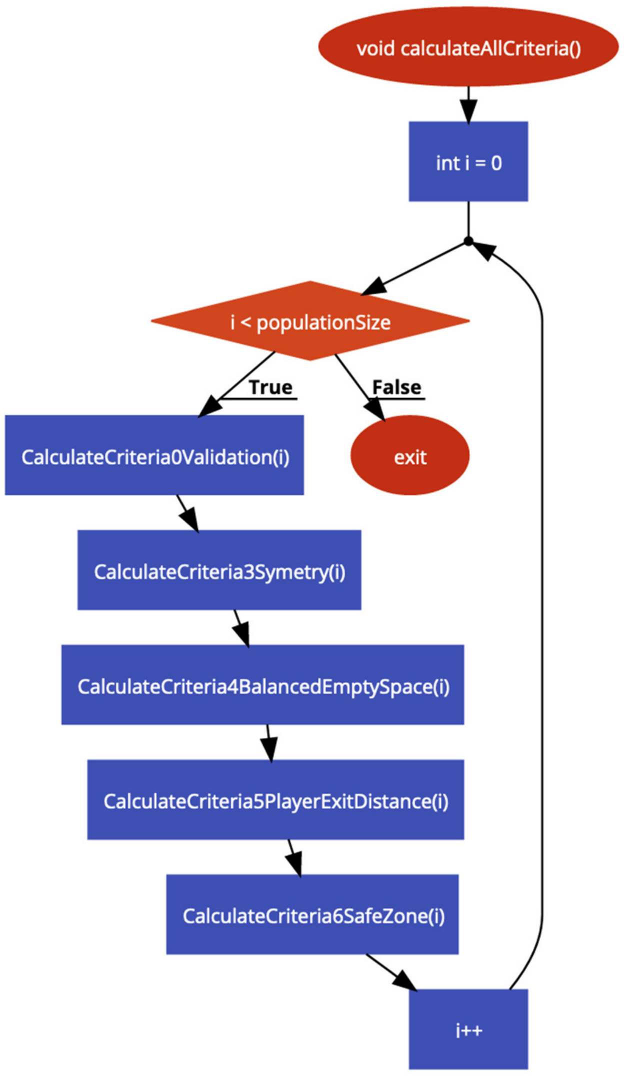
- Distance between player and exit game objects. x and y represent the coordinates of the player and exit (4). This rule makes sure that the player can see as much of the generated scene as possible while traveling to the exit point;

$$d = \sqrt{(x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1)} \tag{4}$$

- The safe zone criteria calculate the amount of Hazzard-type objects in a defined square around the Player and divide the result by the total area of this square (5).

$$z = \frac{x_1 y_1}{x_2 y_2} \tag{5}$$

Criteria are calculated for each member of the population (Figure 5) and can be modified on demand. The criteria calculations are the building blocks of the fitness function.



void calculateAllCriteria()
int i = 0
i < populationSize
True
False
CalculateCriteria0Validation(i)
exit
CalculateCriteria3Symetry(i)
CalculateCriteria4BalancedEmptySpace(i)
CalculateCriteria5PlayerExitDistance(i)
CalculateCriteria6SafeZone(i)
i++

Figure 5. Criteria list.

Constraint functions are:

- Scan the chromosome grid and check if Player object exists;
- Scan chromosome grid and check if an exit object exists;
- Pathfinding algorithm to check if there is a passable way between Player and Exit.

3.3. Application of WASPAS-SVNS in Genetic Algorithm

In the evaluation step, we combine all the fitness results of the criteria functions using the modified WASPAS-SVNS algorithm (Figure 6) [19]. Most previous use cases for this algorithm were tested with single iterations [19–22]. This research focuses on an iterative process with WASPAS-SVNS, so there were tweaks made for it to work together with the genetic algorithm. These are the main steps of the final evaluation and explanations about how it was joined with our procedural generator:

1. Combining criteria evaluation data into matrix X where one dimension represents the index of a chromosome, and another dimension represents the index of the criteria (6);

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \tag{6}$$

2. The original algorithm normalizes the data here, inside the WASPAS-SVNS algorithm, but for the iterative process it does not work because the local min–max and global min–max values are not the same, so we need to define boundaries before this step [33]. Normalization is made in the criteria functions to fit in the range of 0 to 1 (7). v represents current criteria value and v_{max} is the highest possible value for that criterion for the selected matrix size. $\widetilde{x}_{ij}$ is a normalized index ij criteria value of matrix X;

$$\widetilde{x}_{ij} = \frac{v}{v_{max}} \tag{7}$$

3. Neutrosophication step. In this step, we convert results from our normalized criteria function results into neutrosophic sets. The neutrosophic set consists of three numbers: truth (t), intermediary (i), and falsehood (f). For this, we map criteria results with neutrosophic numbers, but we do a linear conversion as even the slightest non-proportional shifts can make a huge error in the long evolutionary run. N represents a neutrosophic number and S represents a scalar number (8);

$$N(t,i,f) = \begin{cases} S \\ 1 - S \\ 1 - S \end{cases} \tag{8}$$

4. Sum of the total relative importance of the alternative (single evolutionary iteration chromosome);
5. Total relative importance of the product of the alternative;
6. A joint generalized criterion for the ranking alternatives (step 4 and step 5) (9);

$$\widetilde{Q}_i = 0.5\widetilde{Q}_i^{(1)} + 0.5\widetilde{Q}_i^{(2)} \tag{9}$$

7. Neutrosophic numbers (truth, intermediacy, and falsehood) are converted to scalar numbers using this formula and then used for chromosome evaluation in the genetic algorithm (10);

$$S\left(\widetilde{Q}_i\right) = \frac{3 + t_i - 2i_i - f_i}{4} \tag{10}$$

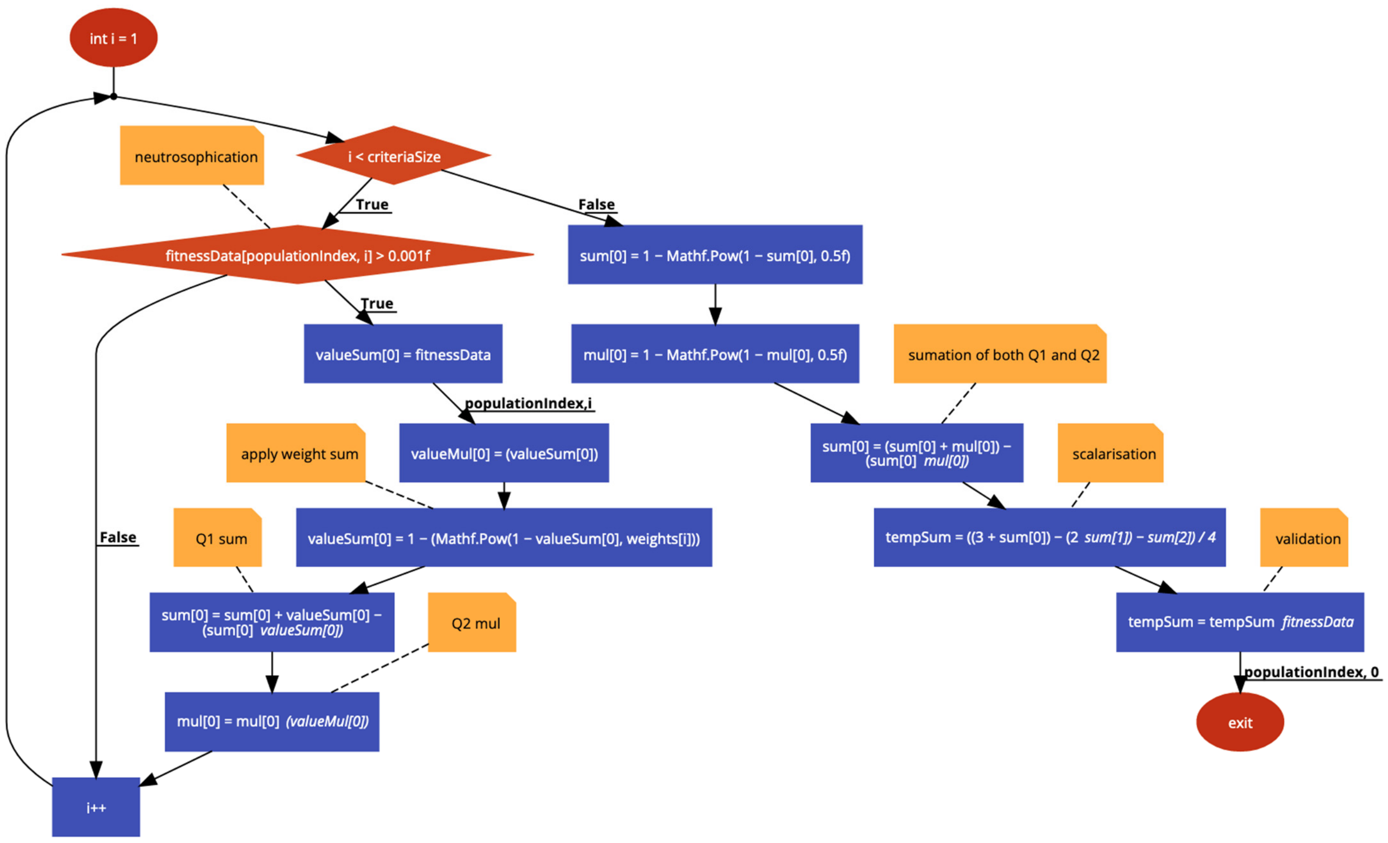

int i = 1
neutrosophication
i < criteriaSize
True
False
fitnessData[populationIndex, i] > 0.001f
sum[0] = 1 − Mathf.Pow(1 − sum[0], 0.5f)
True
valueSum[0] = fitnessData
mul[0] = 1 − Mathf.Pow(1 − mul[0], 0.5f)
sumation of both Q1 and Q2
populationIndex,i
apply weight sum
valueMul[0] = (valueSum[0])
sum[0] = (sum[0] + mul[0]) − (sum[0] mul[0])
scalarisation
False
Q1 sum
valueSum[0] = 1 − (Mathf.Pow(1 − valueSum[0], weights[i]))
tempSum = ((3 + sum[0]) − (2 sum[1]) − sum[2]) / 4
validation
sum[0] = sum[0] + valueSum[0] − (sum[0] valueSum[0])
Q2 mul
tempSum = tempSum fitnessData
populationIndex, 0
mul[0] = mul[0] (valueMul[0])
exit
i++

Figure 6. Fitness algorithm by the MCDM approach.

3.4. Proposed Extension of Genetic Algorithm

Genetic algorithms are used to solve optimization problems using natural selection [34]. It is convenient for us, as it can iterate in many different local maximums because of its random nature. It iteratively modifies the population of individual game level grids. At each generation, random grids are selected and modified. With time, the population evolves toward aesthetic and functional solutions. Genetic algorithms allow for finding good solutions without designing them. Transformative genetic algorithm operators are applied to a small set of grid cells. The main advantages from other optimization algorithms are: non-linear convergence, more than one solution can evolve in parallel and best solutions are kept, uses a lot of random numbers so it is not deterministic, and each run of the algorithm proposes different solutions. The fitness function is based on the WASPAS-SVNS algorithm and is used to find the best individuals using several criteria. One grid represents a single solution to a problem. Population defines the total concurrent grid pool. Best fitness shows the best grid designs in the current population. Two concurrent snapshots are used during calculations: parent and child generation.

The level layout is trained with the genetic algorithm (Algorithm 1), and the evaluation criteria for each iteration are combined with the WASPAS-SVNS algorithm to calculate the single fitness value. The population size is set to 50, and the algorithm runs for 2000 iterations. We are using selection and mutation operators to filter and repopulate the population. For initial data, we create empty chromosomes and fill them with random data where each object is coded with integer numbers from 2 to 6 (all possible objects except player (number 0) and exit (number 1)), and then add 1 Player and 1 Exit object. For each iteration, we calculate the median fitness value for all populations and split chromosomes into two temporal arrays, which store chromosomes below and above the median value. Chromosomes below the median value are replaced with chromosomes from the above median array, and then 5% of this new array data is mutated with new random values (Figure 7).

Algorithm 1. Genetic algorithm.

```
InitializeRandomPopulation:
DoFullEvolution:
  for amountOfEvolutionCycles
    CalculateAllCriteria
      for populationSize
        Validation
          PlayerExists
          ExitExists
          PathBetweenPlayer-ExitExists
        Symetry
        EmptySpaceBalance
        Player-ExitDistance
        SafeZone
    FindUnderperformersAndPerformers
      for populationSize calculateFitness
          WASPAS-SVNS
    EvolveUnderperformersWithGeneticAlgorithm
DrawGrid(best fitness):
```

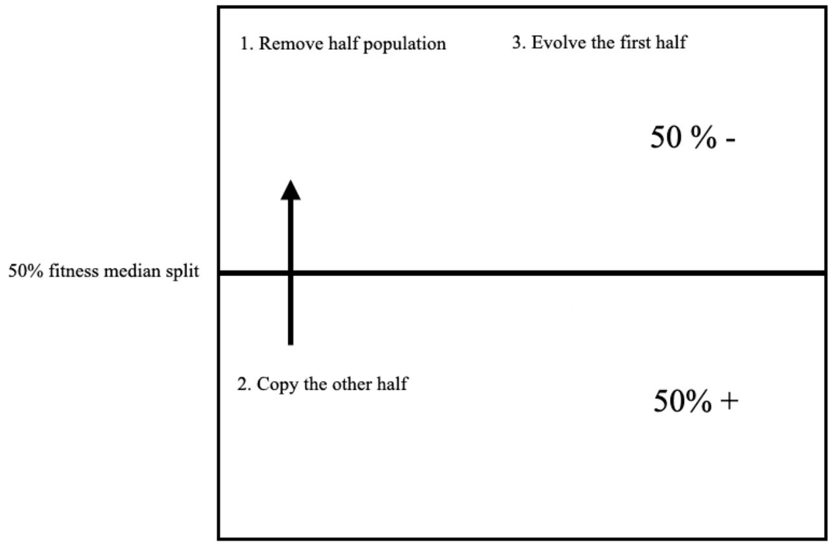


Figure 7. Data evolution.

4. Results

For this research, we developed a framework from scratch using the Unity game engine and visual game object assets from the Unity Asset Store. The results are generated with a custom C# script. Tests were performed with a 2.4 GHz 8-Core Intel Core i9 CPU. Procedural generator with neutrosophic evaluation generates quickly rising scores for the first 100–200 generations under current conditions compared to summation of individual criteria fitness scores, but generator usually requires more time to make symmetrical and visually balanced scene layouts while making sure that game rules apply. The final fitness score usually sets at around 0.75–0.85. It is important to have lots of local maxima for the game scene generation, as results must be unique and differ from each other. There are many possible solutions based on the random initial seed and mutations. Fitness examples with different seeds of random initial data and 500 generations (Figure 8). Note that close to 1 fitness is not possible, as the criteria conflict with each other. Fitness usually starts to converge after about 500–2000 generations. It takes about 21 s, on average, to calculate one 10×10 grid level with 2000 generations. As the initial population (50) is relatively low compared to the total possible scene layouts (7 to power of 100) and mutations are set to

5%, initial seeds usually define how wide the final fitness range is. We choose the lower population to have a wider number of possible solutions. The goal is not to optimize the algorithm for one solution, but to generate a diverse set of level layouts that satisfy the creativity and game design criteria.

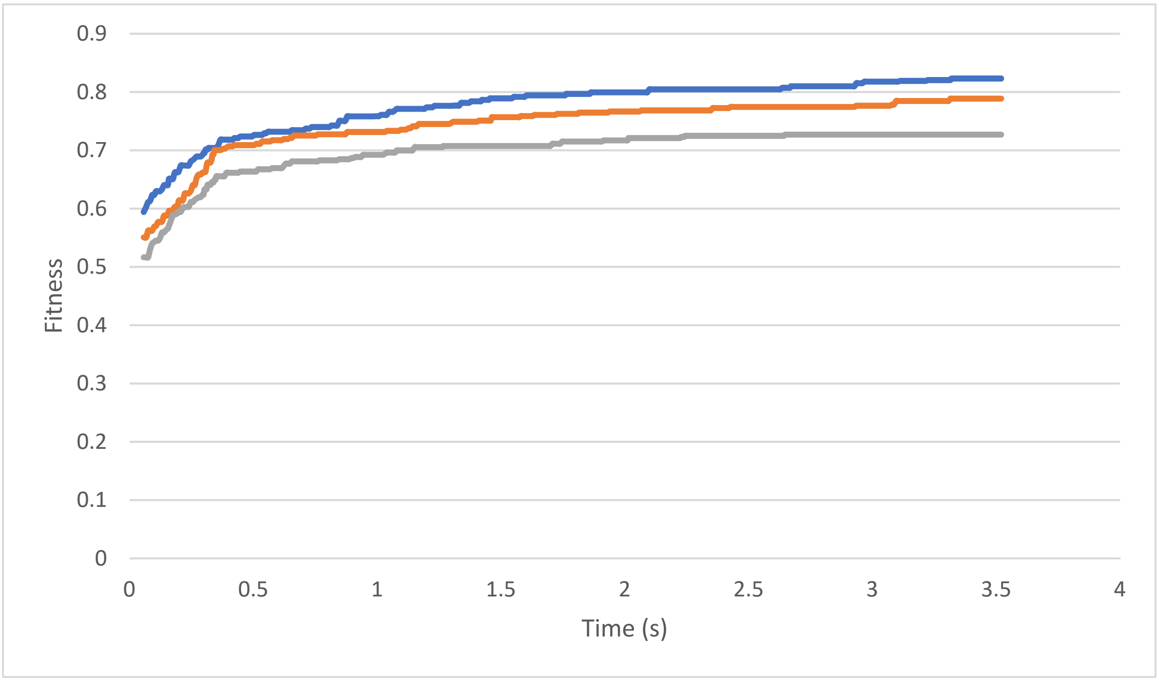


Figure 8. Fitness evolution examples.

We can also observe visual results, which generate aesthetically appealing game scenes. It has many elements of symmetry and space balance. Some examples are: room-like shape without a specific code that defines what a room is (Figure 9), as symmetry is conflicting with other criteria and is not strictly defined, we can also observe semi-symmetric shapes (Figure 10), a smaller room inside the scene with lots of coins/rewards (Figure 11), game scene without lots of walls (Figure 12). We can see that the generator can create many different aesthetic shapes.

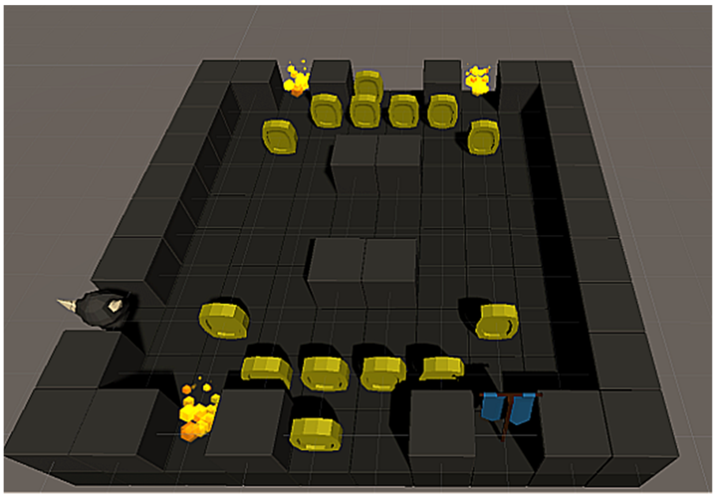


Figure 9. Room-like generated scene.

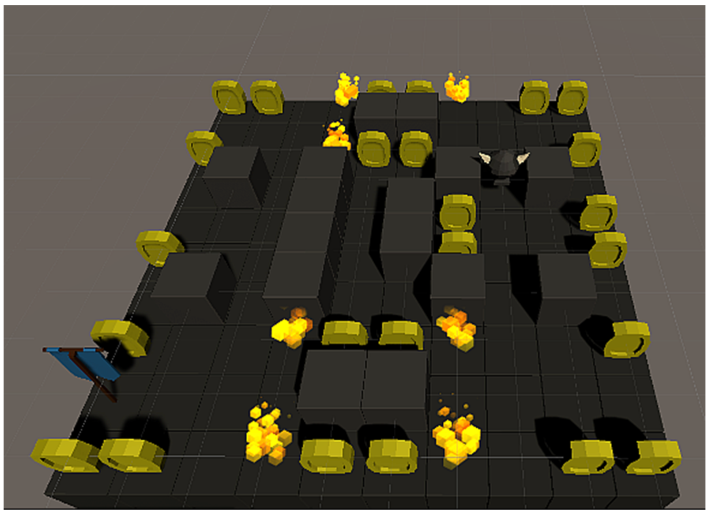


Figure 10. Semi-symmetric results.

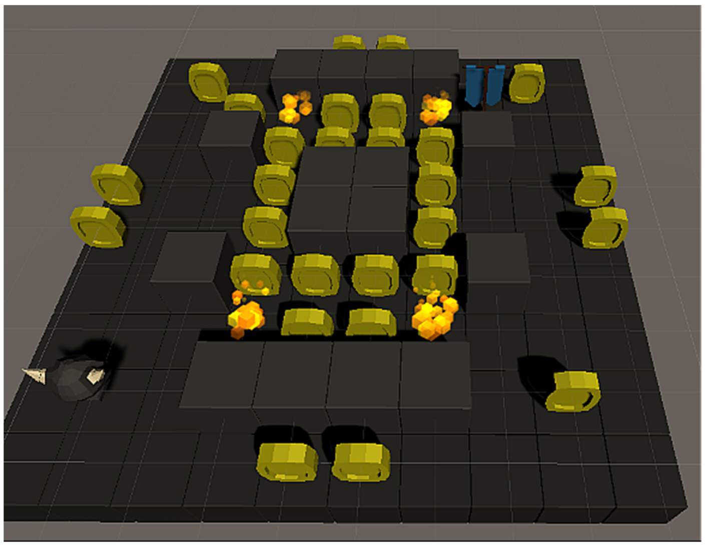


Figure 11. Small room inside the scene.

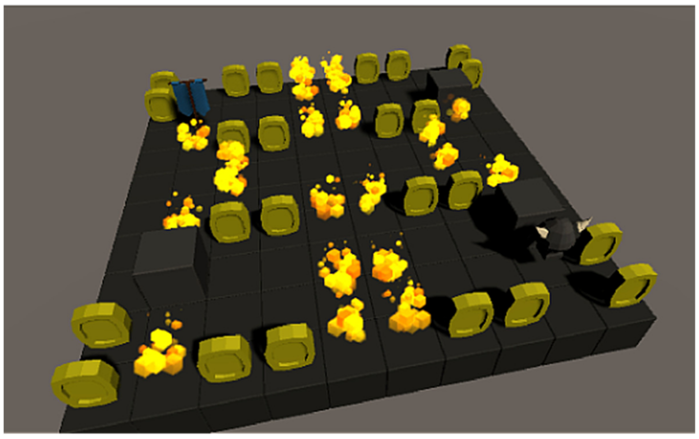


Figure 12. Game scene with almost no walls.

An example of intermediate evolution results can be observed (Figure 13). The grid is printed every 100 generations. We can observe a chaotic layout and quick progression early on and fine-tuning in the later generations.

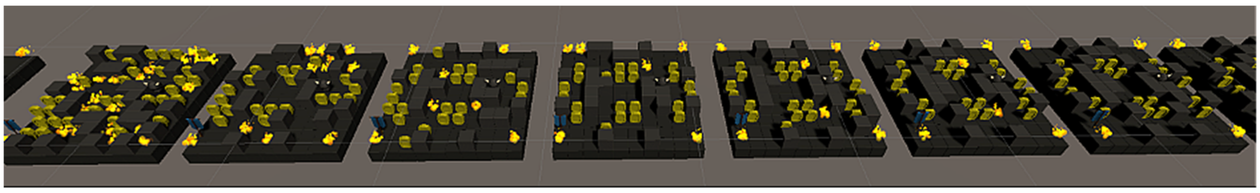


Figure 13. Intermediate evolution results (left to right).

On closer inspection, the realization of the aesthetic criterion can be seen in the visual examples (Figure 14) (symmetry and balance of the empty space balance). At the same time, game design requirements, such as pathfinding, are realized.

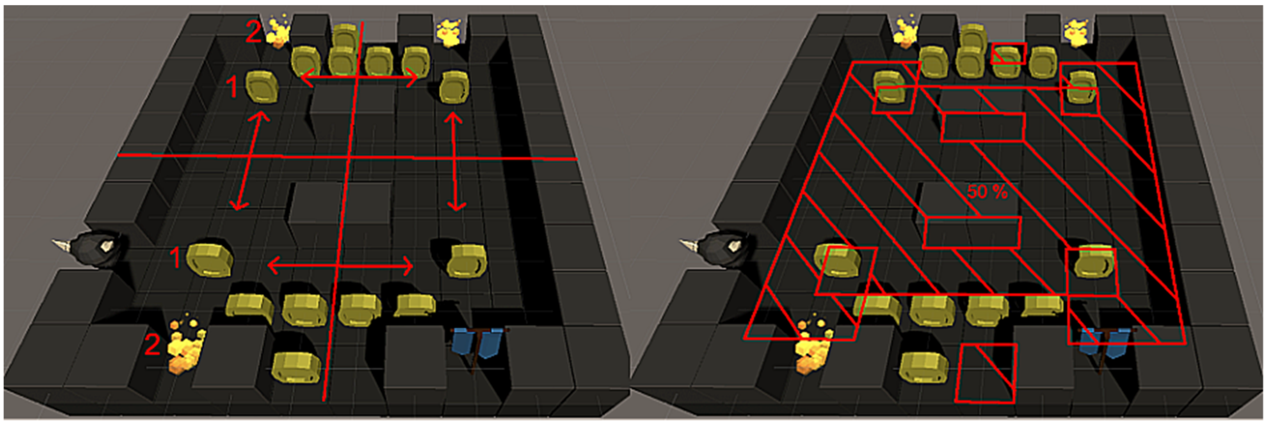


Figure 14. Symmetry and empty space balance. The numbers '1' and '2' represent the corresponding symmetrical objects. '50%' represents the balance between the number of objects and the empty space.

Each result is a different local maximum from many possible final results. It is a very small chance to generate an identical game level given the nature of the algorithm randomness and the number of possible solutions. It would also be possible to draw a crude initial room and then let the algorithm fine-tune it to satisfy aesthetic and playability criteria. The optimal solution for this task is total satisfaction with the proposed criterion under a given random initial seed. Compared to other similar research in the field, the proposed framework generates more visually noticeable aesthetic traits on small object resolutions while maintaining an above-average object pool. This approach to procedural generation has the potential to make creative work faster and easier.

5. Conclusions

The main problem that the proposed method solves is how to increase unique and not repetitive amounts of levels with several runs of the same algorithm. Observing the presented results, it can be concluded that our levels generate interesting game scene layouts, which differ with each run. It can also generate both aesthetic- and functional-level layouts at the same time. Visual representations of game assets can also be interchanged by a developer. The WASPAS-SVNS algorithm enables the evaluation of conflicting criteria. The proposed approach is realized by breaking down design principles into primary elements and defining them with the proposed criteria list. The algorithm generates a random shape and then sculpts a functional and aesthetic game level around that shape. The random nature of the genetic algorithm ensures surprise elements for the levels. It is also important to find a balance between different criteria weights and number of criteria that defines a certain objective to generate a coherent final level. Overabundance or lack of features may numb some of the game design elements. Creativity assistance algorithms can save time for game designers and developers, but at the moment, most commercial games use only light game design assistance tools, seeded procedurally generated or handcrafted levels. This work can be expanded by combining it with an algorithm, which can break down design elements from hand-crafted game levels and then use it as a base of criteria list.

Author Contributions: Conceptualization, A.P. and R.B.; methodology, A.P. and R.B.; software, A.P.; validation, A.P. and R.B.; formal analysis, A.P. and R.B.; investigation, A.P.; resources, A.P.; data curation, A.P.; writing—original draft preparation, A.P.; writing—review and editing, A.P. and R.B.; visualization, A.P.; supervision, R.B.; project administration, R.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used to support the findings of this research are available upon request to the authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dick, S. Artificial Intelligence. *Harv. Data Sci. Rev.* **2019**, 1–8. [CrossRef]
2. Lamb, C.; Brown, D.G.; Clarke, C.L.A. Evaluating Computational Creativity: An Interdisciplinary Tutorial. *ACM Comput. Surv.* **2018**, *51*, 1–34. [CrossRef]
3. Carballal, A.; Fernandez-Lozano, C.; Rodriguez-Fernandez, N.; Castro, L.; Santos, A. Avoiding the Inherent Limitations in Datasets Used for Measuring Aesthetics When Using a Machine Learning Approach. *Complexity* **2019**, *2019*, 4659809. [CrossRef]
4. Ventura, D. Mere generation: Essential barometer or dated concept? In Proceedings of the Seventh International Conference on Computational Creativity ICCC 2016, Paris, France, 27 June–1 July 2016. Available online: http://www.computationalcreativity.net/iccc2016/wp-content/uploads/2016/01/Mere-Generation.pdf (accessed on 7 July 2021).
5. Cook, M.; Colton, S.; Gow, J. *Nobody's A Critic: On the Evaluation of Creative Code Generators—A Case Study in Video Game Design*; ICCC: Sydney, Australia, 2013.
6. Toivonen, H.; Gross, O. Data mining and machine learning in computational creativity. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2015**, *5*, 265–275. [CrossRef]
7. Franceschelli, G.; Musolesi, M. Creativity and Machine Learning: A Survey. *arXiv* **2021**, arXiv:2104.02726 2021.
8. Summerville, A.; Snodgrass, S.; Guzdiaz, M.; Holmgård, C.; Hoover, A.K.; Isaksen, A.; Nealen, A.; Togelius, J. Procedural content generation via machine learning (PCGML). *IEEE Trans. Games* **2018**, *10*, 257–270. [CrossRef]
9. Togelius, J.; Champandard, A.J.; Lanzi, P.L.; Mateas, M.; Paiva, A.; Preuss, M.; Stanley, K.O. Procedural content generation: Goals, challenges and actionable steps. *Dagstuhl Follow-Ups* **2013**, *6*, 61–75.
10. Risi, S.; Togelius, J. Increasing generality in machine learning through procedural content generation. *Nat. Mach. Intell.* **2020**, *2*, 428–436. [CrossRef]
11. Isaksen, A.; Gopstein, D.; Togelius, J.; Nealen, A. Discovering Unique Game Variants Computational. In Proceedings of the Creativity and Games Workshop at the 2015 International Conference on Computational Creativity, Park City, UT, USA, 29 June–2 July 2015.
12. Liapis, A.; Yannakakis, G.N.; Togelius, J. *Limitations of Choice-Based Interactive Evolution for Game Level Design*; AAAI: Stanford, CA, USA, 2012.
13. Cook, M.; Colton, S.; Pease, A.; Llano, M.T. *Framing in Computational Creativity—A Survey and Taxonomy*; ICCC: Charlotte, NC, USA, 2019.
14. Khalifa, A.; de Mesentier Silva, F.; Togelius, J. Level Design Patterns in 2D Games. In Proceedings of the 2019 IEEE Conference on Games (CoG), London, UK, 20–23 August 2019; pp. 1–8. [CrossRef]
15. Togelius, J.; Schmidhuber, J. An experiment in automatic game design. In Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, Perth, WA, Australia, 15–18 December 2008; pp. 111–118.
16. Sweetser, P.; Wyeth, P. GameFlow: A model for evaluating player enjoyment in games. *Comput. Entertain.* **2005**, *3*, 3. [CrossRef]
17. Sorenson, N.; Pasquier, P. Towards a Generic Framework for Automated Video Game Level Creation. In *European Conference on the Applications of Evolutionary Computation*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 131–140.
18. Lara-Cabrera, R.; Cotta, C.; Fernández-Leiva, A.J. On balance and dynamism in procedural content generation with self-adaptive evolutionary algorithms. *Nat. Comput.* **2014**, *13*, 157–168. [CrossRef]

19. Lescauskiene, I.; Bausys, R.; Zavadskas, E.K.; Juodagalviene, B. VASMA weighting: Survey-based criteria weighting methodology that combines ENTROPY and WASPAS-SVNS to reflect the psychometric features of the VAS scales. *Symmetry* **2020**, *12*, 1641. [CrossRef]
20. Morkunaite, Z.; Bausys, R.; Zavadskas, E.K. Contractor Selection for Sgraffito Decoration of Cultural Heritage Buildings Using the WASPAS-SVNS Method. *Sustainability* **2019**, *11*, 6444. [CrossRef]
21. Bausys, R.; Juodagalviene, B.; Žiuriene, R.; Pankrasovaite, I.; Kamarauskas, J.; Usovaite, A.; Gaižauskas, D. *The Residence Plot Selection Model for Family House in Vilnius by Neutrosophic WASPAS Method*; VGTU Press: Vilnius, Lithuania, 2020; Volume 24, pp. 182–196.
22. Bausys, R.; Kazakeviciute-Januskeviciene, G. Qualitative rating of lossy compression for aerial imagery by neutrosophic WASPAS method. *Symmetry* **2021**, *13*, 273. [CrossRef]
23. Semenas, R.; Bausys, R.; Zavadskas, E.K. A novel environment exploration strategy by m-generalised q-neutrosophic WASPAS. In *Studies in Informatics and Control*; National Institute for R&D in Informatics, ICI Bucharest: Bucharest, Romania, 2021; Volume 30, pp. 19–28.
24. Bausys, R.; Kazakeviciute-Januskeviciene, G.; Cavallaro, F.; Usovaite, A. Algorithm Selection for Edge Detection in Satellite Images by Neutrosophic WASPAS Method. *Sustainability* **2020**, *12*, 548. [CrossRef]
25. Ali, J.; Bashir, Z.; Rashid, T. WASPAS-based decision making methodology with unknown weight information under uncertain evaluations. *Expert Syst. Appl.* **2020**, *168*, 114143. [CrossRef]
26. Pereira, L.T.; Toledo, C.; Ferreira, L.N.; Lelis, L.H. Learning to Speed up Evolutionary Content Generation in Physics-Based Puzzle Games. In Proceedings of the 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI), San Jose, CA, USA, 6–8 November 2016; pp. 901–907.
27. Thakkar, S.; Cao, C.; Wang, L.; Choi, T.J.; Togelius, J. Autoencoder and Evolutionary Algorithm for Level Generation in Lode Runner. In Proceedings of the IEEE Conference on Games (CoG), London, UK, 20–23 August 2019; pp. 1–4.
28. Zafar, A.; Mujtaba, H.; Beg, M.O. Search-based procedural content generation for GVG-LG. *Appl. Soft Comput.* **2020**, *86*, 105909. [CrossRef]
29. Volz, V.; Justesen, N.; Snodgrass, S.; Asadi, S.; Purmonen, S.; Holmgård, C.; Togelius, J.; Risi, S. Capturing Local and Global Patterns in Procedural Content Generation via Machine Learning. In Proceedings of the 2020 IEEE Conference on Games (CoG), Osaka, Japan, 24–27 August 2020; pp. 399–406.
30. Safak, A.B.; Bostanci, E.; Soylucicek, A.E. Automated Maze Generation for Ms. Pac-Man Using Genetic Algorithms. *Int. J. Mach. Learn. Comput.* **2016**, *6*, 226–230. [CrossRef]
31. Petrovas, A.; Bausys, R. Automated Digital Terrain Elevation Modification by Procedural Generation Approach. In Proceedings of the 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 25 April 2019; pp. 1–5. [CrossRef]
32. Stanujkić, D.; Karabašević, D.; Popović, G.; Pamučar, D.; Stević, Ž.; Zavadskas, E.K.; Smarandache, F.A. Single-Valued Neutrosophic Extension of the EDAS Method. *Axioms* **2021**, *10*, 245. [CrossRef]
33. Padhye, N.; Deb, K. Multi-objective optimisation and multi-criteria decision making in SLS using evolutionary approaches. *Rapid Prototyp. J.* **2011**, *17*, 458–478. [CrossRef]
34. Whitley, D. A genetic algorithm tutorial. *Stat. Comput.* **1994**, *4*, 65–85. [CrossRef]