Procedural Level Generation with Difficulty Level Estimation for Puzzle Games

Łukasz Spierewka[1], Rafał Szrajber[1 [0000−0003−2777−0251]], and Dominik Szajerman[1 [0000−0002−4316−5310]]

Institute of Information Technology, Lodz University of Technology, Łódź, Poland
dominik.szajerman@p.lodz.pl

Abstract. This paper presents a complete solution for procedural creation of new levels, implemented in an existing puzzle video game. It explains the development, going through an adaptation to the genre of game of the approach to puzzle generation and talking in detail about various difficulty metrics used to calculate the resulting grade. Final part of the research presents the results of grading a set of hand-crafted levels to demonstrate the viability of this method, and later presents the range of scores for grading generated puzzles using different settings. In conclusion, the paper manages to deliver an effective system for assisting a designer with prototyping new puzzles for the game, while leaving room for future performance improvements.

Keywords: Procedural content generation · Puzzle game · Difficulty level estimation.

1 Introduction

This paper presents a complete solution for procedural generation of levels in an existing puzzle video game called *inbento*. It is a mobile puzzle game released in September 2019 for the iOS and Android platforms. In this game the player is tasked with finishing over a hundred levels. They are introducing new concepts and gradually rising in difficulty.

The theme of the game is based on the idea of preparing bento: a type of Japanese cuisine where the meal is packed tightly in a container. Bento boxes can usually contain various ingredients: boiled rice, raw or cooked fish, prepared egg, vegetables, sandwiches or more [13].

1.1 Game rules

inbento is centered around the idea of preparing a complete bento box in accordance with a recipe given to the player in each level. The game view consists of three main elements: the recipe book, the bento box and the cutting board, each serving a distinct purpose. Figure 1 presents three example game views. The right side of the recipe book contains the desired end state of the puzzle – a solution that has to be replicated by the end user. In the middle of the game

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

view sits the bento box. Figure 1 on the left shows the initial state of the level. The goal of the game is making this box look exactly like the reference image shown in the recipe book.

Fig. 1. Various stages of trying to place a piece into the box; Left: initial state (not picked up); Middle: valid placement (indicated by black markers displayed in the box); Right: invalid placement.

On the very bottom sits the cutting board, serving as a collection of interactive pieces: composites of one or more blocks that can be placed into the bento box. Using all available pieces to complete a puzzle is one of the requirements of the game. Upon dragging the piece into the box, the game validates whether all of the blocks fit within the boundaries of the container. If the result is negative (e.g. Figure 1, right), the piece is sent back into the cutting board. If it is positive, piece blocks are placed into the box replacing previous content.

Boxes in the game can take on various sizes, from a single-cell grid (1×1) all the way up to a four-by-three grid (4×3). Similarly, the maximum size of the cutting board was limited to up to 8 pieces. The maximum piece size is 3×3 blocks. As seen on Figure 2, each piece can be rotated to appear in one of four different states which can later be placed into the box. Because of this, each piece can allow the creation of up to four times as many different states upon placement, which is utilized to increase puzzle complexity.

Despite the apparent simplicity of the game, it turns out that even small differences in the size of the box, the number and size of pieces, the size of cutting board translate into an exponentially growing number of combinations of solutions, which makes the design process difficult. Procedural level generation

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

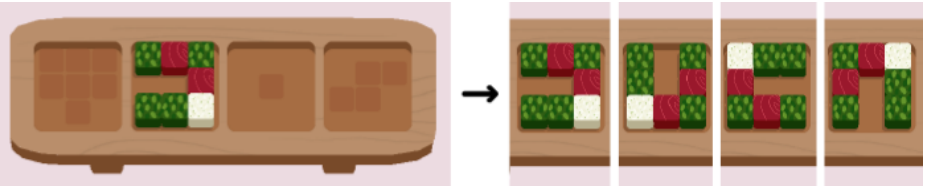

Fig. 2. Each piece can be rotated to 0, 90, 180 and 270 degrees.

and the testing of its results are also made difficult by this. This paper shows these issues and our way of addressing them.

Since finished bento boxes rarely contain empty space, all final solutions for every level in the game do not contain any empty blocks. It was closely followed during the design stage of *inbento* and preserving it for automatically generated levels is also desirable.

1.2 Piece types and mechanics

In order to make the game more satisfying in later chapters, *inbento* continuously introduces new piece types (Figure 3) that affect the gameplay in various ways:

1. **Food Piece** is the most basic piece type. Consists of ingredient blocks that replace any blocks they're placed on top of.
2. **Swap Piece** – by using it the user is able to switch positions of two blocks inside the box. The piece needs to consist of exactly two non-empty blocks, and at least one of them needs to be placed on a non-empty block.
3. **Move Piece** – upon use of it, all affected blocks will be moved by one grid cell in accordance with directions of the arrows visible on each piece block. After a block has been moved it leaves behind an empty space.
4. **Grab Piece** – after placement all affected box blocks are taken out of the container and sent back into the cutting board, creating a new food piece mirroring the shape of the affected box blocks. At least one targeted block must be non-empty for this piece to work.
5. **Copy Piece** consists of two block types – the "source" block and "target" blocks. Upon placement, all target blocks (regardless of whether they are empty or not) are replaced with the source block's type. For this piece to work, the source type must not be empty and the piece itself needs to contain at least one source block and one target block.
6. **Rotation Lock** is a special mechanic that can be used together with all other piece types. When enabled, the affected piece cannot be rotated and has to be placed in the box as-is. Since using this mechanic reduces the number of available states generated from each placement, it was used mainly to lower the complexity of earlier levels in order to maintain a smoother difficulty curve across chapters.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

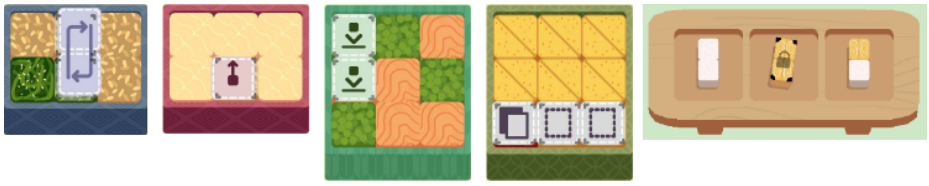

Fig. 3. Various piece types. From the left: swap, move, grab, copy, rotation lock.

2 Related work

2.1 Procedural content generation

With the complexity of games being on the rise since the start of the industry there has also been an increasing demand for titles that provide more content, and thus more playtime for the end user. While companies at the highest tiers – the so-called AAA developers, which is a term referring to games with the largest budgets for creation and marketing [8] – can fill their titles with more things to do by scaling up the workforce, smaller developers often do not have the money or the time that would allow them to catch up to their larger competitors. Thus, the need for being able to generate new content without hand-authoring it was born.

Through the employment of Procedurally Generated Content (PCG) [18], developers can stretch out their game length almost indefinitely. Instead of creating all high-quality content such as levels, art assets or even game mechanics or NPCs behavior by hand, programmers can instead define the boundaries of a system governed by algorithms, which takes in certain pre-programmed inputs in order to create entirely new outputs [16, 15].

There are various existing cases of using PCG to create large amounts of in-game content in titles both big and small.

"Rogue" is one of the best-known early examples of using generation to create unique maps on the spot, which resulted in a different playthrough for the player each time they launch the game. This mixture of theoretically infinite levels combined with challenging "permadeath" gameplay was so impactful that it launched an entirely new subgenre of games called roguelikes [14] featuring titles like "NetHack", "Ancient Domains of Mystery" and "Angband".

Another example of PCG can be found in No Man's Sky. This title is a space-exploration adventure game and one of the most recognizable releases of last few years, mainly due to the fact that the title's simulated universe contains 18 quintillion planets. Hand-crafting this amount of content would usually take an enormous amount of time, which is why the developers of the game have created a system that generates these planets based on an existing 64-bit seed value [11].

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

2.2 Difficulty estimation

When planning out player progression in a game, designers often strive for optimal results in terms of introducing a new user into the experience and ensuring that they are enjoying themselves during the entire duration of the title. This desire is tightly coupled with the game's difficulty presented through challenges that are meant to counter the player's abilities and knowledge that are growing with each minute spent with the simulation.

The concept of flow state named by Csíkszentmihályi [5] outlines that the optimal state of happiness for a person appears when they are presented with challenges matching their skill level. If the challenge is too low, the user experiences apathy or boredom; if it is too high they might feel anxious and worried.

What sits in the middle of these two states is the flow channel. If the player skills are met with the right amount of challenge, they should feel stimulated and engaged by the game (Figure 4, left) [2].

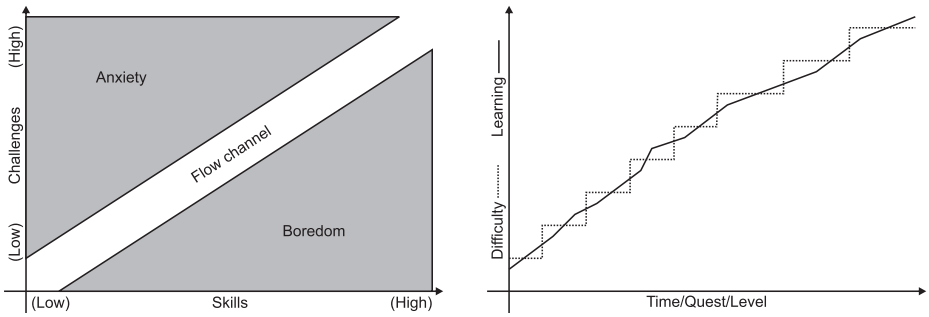

Fig. 4. Left: A visualization of the flow channel [5]; Right: A graph depiction of the difficulty and learning (player skill) curves; the sections when one of the curves rises above the other exemplifies maneuvering within the flow channel [2].

One tool that can help the designers with ensuring that the experience stays within the desired area for as long as possible is the concept of difficulty curves (Figure 4, right), which serve as a graphical representation of how the game's difficulty changes over course of the playthrough [6, 1]. Usually these are categorized into two main types: time-based (based on how long the user was playing) and distance-based (how much of the game has been finished by the user). Through mapping out the title's progression in this manner, the creators can aim to deliver an optimal first time user experience (FTUX) [7] – a concept that is especially important for games targeted for casual audience playing on platforms that do not garner long-term attention from users, such as smartphones.

The biggest issue in applying these methods comes down to the fact that often their data is supplied through user focus testing which can be time-consuming and costly. Difficulty curves are usually formulated by the designer and then refined through testing the game's content on potential players [17]. However,

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

there have been multiple papers with different approaches to difficulty estimation that is less reliant on user data; through genetic algorithms [3], constraint satisfaction problem (CSP) solving [10] or calculating common features in games into a single difficulty function [12] in order to lessen the burden on the creators. The last-mentioned method has been adapted in our work to evaluate dfficulty during the procedural level generation.

3 Method

3.1 Procedural level generation

PCG is usually based on random numbers. Our initial approach was to randomly select from available piece types and try to generate a solution. The method attempted to insert the piece into the box a specified number of times at various positions and rotations, in order to try and naïvely match it to the grid. If the function did not succeed within this possibility space, it was assumed that the piece could not be properly placed under the existing conditions and discarded. The main advantages of this method are its reliability and speed. Since the result is generated through a simulated act of regular play (placing the pieces in the box one-by-one), the end state is guaranteed to be achievable by the rules of the game which avoids the need to verify whether a solution exists.

Unfortunately, because of the simplicity of this approach the finished levels are usually of a poor quality. Final solutions often contain empty spaces and are not visually pleasing. Furthermore, the possibility space of the level is not fully explored during generation, the algorithm has no way of ascertaining whether the end result can be achieved using a smaller amount of pieces than the entire inventory available to the player. This directly contradicts the rules and warranted the search for an alternative approach.

While the previous algorithm only tested a singular path from the initial state to the solution, the second iteration fully examines the total possibility space from the beginning to all potential end states of the graph tree in order to later be able to remove unwanted solutions.

Given an initial state the algorithm tries to explore it, creating child nodes in the tree and later examining them one-by-one, until all non-discarded states have zero remaining pieces, after which undesired states are removed from the final list.

The function is mapping out the entire tree of states using an approach similar to how a BFS (Breadth First Search) algorithm works for undiscovered graphs [9]. The algorithm keeps track of remaining nodes, removes elements from it one by one and tries to map all of its children that can be explored further, until the results (states without remaining pieces) are achieved. By using a queue, the tree is traversed in linear fashion, level sequence after level sequence until the bottom of the graph is reached.

During exploration the algorithm attempts to place each piece remaining in the state into the box using the placement sets. These sets represent pre-

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

computed arrays of grid cells affected during piece placement (Fig. 5). By preparing this data beforehand for each piece the algorithm is able to perform faster compared to a version where during each iteration for a given piece it would have to search for all valid placements.

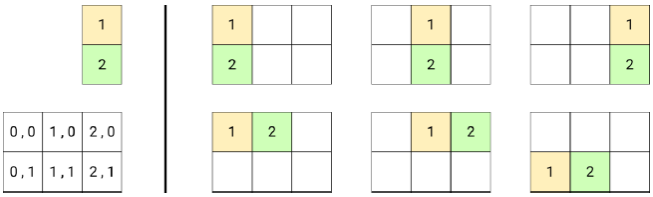

Fig. 5. Left: an example piece, and a 3x2 grid with coordinates in each cell; Right: example placement sets for different positions and rotations of the example piece.

If the placement was unsuccessful – for example, if the piece could not be placed in this location due to piece rules detailed in subsection 1.2 or because the move did not generate any changes in the box (i.e. replacing blocks with exact same ones) – the state is discarded.

However, if placement was successful the piece is removed from this newly created state and based on the number of remaining pieces it is either added back into the queue (if there are any pieces left) or added to the results (if there are none remaining). If the newly generated result has already been achieved, the duplicate counter of its predecessor is incremented in order to use this information later.

Once the queue is emptied, unwanted duplicates – states that have been found at earlier graph levels – are discarded in order to ensure that the final solution can only be achieved through using up all available pieces.

In order to get the final results a filtering and sorting algorithm has been implemented. This method takes in all gathered results, removes all solutions containing any empty fields, and finally orders the list starting with solution states containing the smallest number of duplicates.

Through testing it has been found that levels with the least amount of alternative paths towards the solution are usually the most interesting. The function also outputs all potential alternatives for the designer to select the desired end state.

The exhaustive nature of this solver-assisted generation method ensures that any final state can only be found at the requested depth and is able to filter the selection of results through the lens of solution uniqueness.

However, all of that comes at the expense of speed. The computational requirements of this BFS-inspired algorithm increase exponentially based on initial settings. Simple 2×2 levels with up to 4 pieces are usually generated in less than a second, but raising the amount of pieces above that number brings in a roughly tenfold increase in waiting times depending on pieces that are initially generated.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

As the tool is not meant for consumer use and was designed mostly to assist game developers, it was assumed that this was an acceptable compromise in exchange for the improved end results.

3.2 Difficulty level estimation

In this work, a combination of all categories of difficulty measures [12] has been adapted through metrics which have been outlined:

1. **Search depth** v_{sd} is a metric of deductive iterations required to solve a puzzle [4], and was calulated from the maximum number of states s that can be generated for given puzzle:
$$v_{sd} = \min(1, \frac{\log_{100}(s)}{5}) \tag{1}$$
2. **Palette** v_{pa} is determined by the number of different food types t available in the level:
$$v_{pa} = \min(1, \frac{\exp(t)}{50}) \tag{2}$$
3. **Piece types** v_{pt} is based on the variety of different u mechanics present in the puzzle:
$$v_{pt} = \min(1, \frac{u}{4}) \tag{3}$$
4. **Extra surface** v_{es} compares the total surface area a_p of available pieces (the number of non-empty blocks) with the level area a_l in order to determine how many supplementary tiles are contained in the user's inventory:
$$v_{es} = \min(1, \log_{64}(\max(a_p - a_l))) \tag{4}$$
5. **Duplicate solutions** calculated using the BFS-inspired solver; first, the tree of possible results is fully explored in order to determine the number of paths (duplicates) d leading to the desired solution; afterwards, this value is compared with the total number of states s in order to calculate a perceived difficulty:
$$v_{ds} = (1 - \log_s(d)) \cdot \min(1, \frac{s}{1000}) \tag{5}$$

The calculated values are later combined using a weighted linear function (eq. 6) where each value is treated independently in order to achieve a result that is simple to understand.

$$\text{difficulty}(L) = \sum_{i \in \{sd,pa,pt,es,ds\}} (w_i \cdot v_i(L)) + w_0 \tag{6}$$

In order to ascertain the difficulty grade for a level L in *inbento*, a number of variables V_i with values ranging from 0 to 1 have been multiplied by their respective weight values w_i, and summed together with a base weight w_0.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

There are games and factors where a different relationship can be considered – quadratic or even exponential. However, in the case of *inbento*, after a few attempts, it turned out that the linear relationships are both simple and sufficient (section 4). Moreover, the methods of calculating the coefficients v_i themselves contain non-linearities resulting from their characteristics.

The weights of this function have been selected in the process of optimizing the function through a set of levels hand-crafted by the game's design team. As these puzzles have been hand-crafted and tested in the months leading up to the game's release and iterated post-launch in order to bring the players up to speed with *inbento*'s mechanics, it is believed that this set serves as a decent example for the algorithm.

4 Results

In this section, an overview of the results of the method will be presented, starting with grading existing, hand-crafted levels to measure how close the results were to the intent of the original designer. This will be followed by grading of puzzles created by the generator in order to analyze the quality of the final solution.

First, the difficulty grades for man-made existing levels from Chapters 1 through 4 of the game (36 levels in total) were measured.

The guiding principle behind the design of these levels was to present the player with an optimal FTUX [7] that would ease them into the game and explain mechanics without overly large increases in difficulty.

Each chapter in *inbento* contains 9 levels, only 7 of which need to be completed to be able to proceed into the next set. Furthermore, a chapter can also contain multiple "tutorial" levels that purposefully lower their difficulty in order to focus on explaining a new game system.

Table 1 shows that the difficulty grade does indeed gradually rise between each chapter, showing large dips for tutorial levels that are deliberately simpler in order to teach users new concepts. One outlier in that trend is the existence of two levels: 2-1 and 2-2, which have been specifically designed as more complex in order to highlight the difference between regular blocks, and ones with the rotation lock mechanic.

The method showed promising results when applying the algorithm to puzzles created by a human designer. This part focuses on applying it to a collection of levels generated using the second (BFS-based) algorithm.

To validate the solution a sample of initial generator settings has been selected. For each collection of settings, 20 complete levels (meaning that the level can be solved in at least one way) have been generated and the resulting grades have been presented in Table 2. The final grade is a median of all resulting scores and has been presented along with a median generation time for the chosen settings.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

Table 1. Difficulty grading for Chapters 1-4. The charts are showing final weighted difficulty scores for each level. Green color indicates tutorial levels which in most cases are accompanied by a drastic drop in difficulty.

Level index	1-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9
Tutorial stage	yes	-	-	yes	-	yes	-	-	-
Search depth score	0	0.06	0.12	0	0.12	0.09	0.09	0.18	0.27
Palette score	0.02	0.05	0.05	0.02	0.05	0.05	0.05	0.05	0.02
Piece types score	0	0	0	0	0	0	0	0	0
Extra surface score	0	0	0	0	0	0	0	0	0.26
Solution score	0	0.01	0.01	0	0.01	0.01	0.01	0.08	0.47
Final score	0.02	0.12	0.18	0.02	0.18	0.15	0.15	0.31	1.04
Final weighted score	0.03	0.35	0.64	0.03	0.64	0.49	0.49	1.00	2.22

Level index	2-1	2-2	2-3	2-4	2-5	2-6	2-7	2-8	2-9
Tutorial stage	yes	yes	-	yes	yes	-	-	-	-
Search depth score	0.21	0.21	0.18	0.06	0.12	0.09	0.27	0.3	0.46
Palette score	0.05	0.05	0.15	0.05	0.15	0.15	0.15	0.15	0.15
Piece types score	0	0	0	0	0	0	0	0	0
Extra surface score	0.26	0.26	0.17	0	0.17	0.33	0.17	0.26	0.17
Solution score	0.37	0.27	0.25	0.01	0.03	0.04	0.77	0.69	0.62
Final score	0.9	0.8	0.75	0.12	0.46	0.61	1.35	1.04	1.39
Final weighted score	1.85	1.73	1.56	0.35	0.99	1.09	2.62	2.08	3.33

Level index	3-1	3-2	3-3	3-4	3-5	3-6	3-7	3-8	3-9
Tutorial stage	yes	-	-	-	-	-	-	-	-
Search depth score	0.03	0.11	0.23	0.29	0.29	0.28	0.3	0.34	0.49
Palette score	0.05	0.05	0.05	0.15	0.15	0.15	0.15	0.4	0.4
Piece types score	0	0	0	0.25	0.25	0.25	0.25	0	0
Extra surface score	0	0	0	0.17	0	0.26	0	0	0
Solution score	0	0.01	0.05	0.46	0.7	0.69	0.82	0.41	0.56
Final score	0.08	0.17	0.33	1.31	1.39	1.63	1.52	1.15	1.45
Final weighted score	0.2	0.6	1.18	2.71	2.81	3.09	3.01	2.59	3.05

Level index	4-1	4-2	4-3	4-4	4-5	4-6	4-7	4-8	4-9
Tutorial stage	yes	-	-	-	-	yes	-	-	-
Search depth score	0.22	0.42	0.47	0.59	0.62	0.38	0.51	0.51	0.59
Palette score	0.05	0.15	0.15	0.4	0.15	0.15	0.15	0.4	0.15
Piece types score	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
Extra surface score	0	0	0	0	0	0	0	0	0
Solution score	0.04	0.78	0.95	0.82	0.76	0.8	0.87	0.92	0.91
Final score	0.56	1.06	1.81	2.06	1.78	1.58	1.78	2.08	1.09
Final weighted score	1.05	3.51	3.94	4.67	4.43	3.37	4.04	4.43	4.45

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

Table 2. Median grades and generation times for each collection of generator settings, averaged from 20 solvable, generated levels for each set.

Level size	2×1	2×1	2×2	2×2	2×2	3×2	3×2	3×2
Case	1	2	1	2	3	1	2	3
Food type count	2	2	2	2	4	2	2	2
Piece count	2	2	2	2	3	3	3	4
Piece type count	1	3	1	3	1	1	3	1
Median grade	0.42	0.82	0.38	0.96	1.83	0.97	2.51	2.05
Med. generation time [in ms]	0.02	0.03	0.14	0.22	2.47	4.43	7.37	168.07

Level size	3×2	3×2	3×2	3×3	3×3	3×3	3×3
Case	4	5	6	1	2	3	4
Food type count	4	4	3	2	2	4	4
Piece count	3	4	5	3	3	3	4
Piece type count	1	1	3	1	3	1	3
Median grade	2.63	3.64	4.69	2.59	3.42	3.34	4.67
Med. generation time [in ms]	6.99	240.98	16674.81	30.19	100.69	61.3	3169.38

5 Discussion

The results show that as has been anticipated, increasing level complexity through modifying the settings results in a rise in perceived puzzle difficulty. Different settings contribute to the final results in various ways. For example, while 2×1 and 2×2 levels have very similar perceived difficulty levels for basic settings (two types of food, two food pieces each), using a different set of values for the same sizes results in a much more noticeable difference that only grows as the scale of the puzzle grows (e.g. cases 1 and 2 for 3×2 level).

The largest jumps in difficulty grade usually occur with increasing the number of available pieces – as each additional option available to the player contributes greatly to the search depth measure, levels with more pieces usually score much higher compared to others. This can be seen in the comparison of cases 1 and 3 for level 3×2.

An interesting observation in the analysis of the 3×3 level is a similar jump in difficulty grade when comparing case 1 to case 2, to case 3 and to case 4, respectively. In each of these cases, only one of the three initial parameters was increased.

In turn, comparing the differences between cases 1 and 2 and the differences between cases 1 and 3 for 3×3 levels shows that it is possible to compensate for a change in one of the initial parameters by changing the other so that the final grade remains approximately the same. This allows the designer to experiment with the level parameters despite the need to set it at specific (increasing) levels.

Table 2 also includes median generation times for the levels with given parameters. It is obvious that the piece count parameter has a very large influence on the generation time. Unfortunately, in the case of using an algorithm that searches the entire solution space, which gives accurate results, it is inevitable. There are two possible solutions to this problem if it were to become significant.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

Heuristic approach, available as an option or technical based on parallel or GPU assisted processing.

Examples of the levels from each generator settings have been shown in Figure 6.

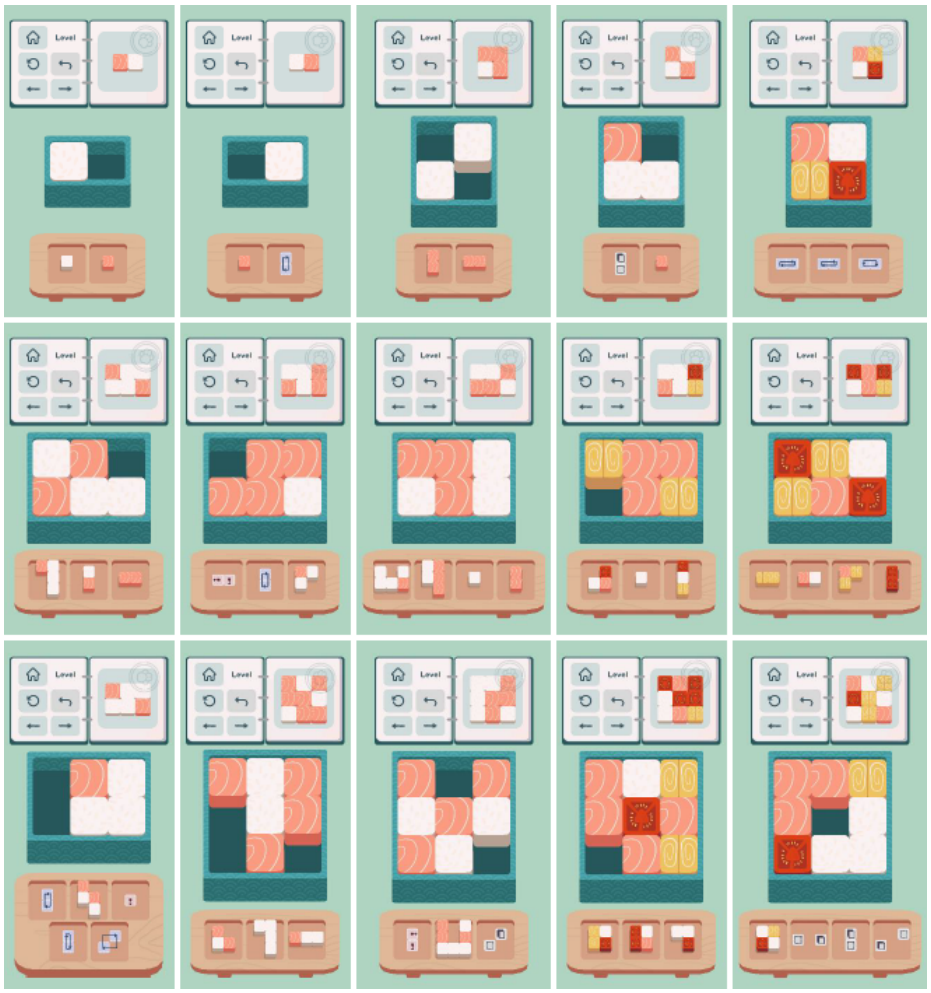

Fig. 6. Example levels generated for each collection of settings from Table 2

.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

6 Conclusions and future work

This work has proposed a complete solution for generating new puzzles for *inbento* and grading their perceived difficulty levels. The final result allows the designer to generate levels for the game using an easily modifiable set of parameters and receive information about the difficulty grade for that level.

Because the rules of the game and selected difficulty variables require full exploration of the possibility space of a given level, the algorithm's computation time increases exponentially as puzzle complexity goes up. Further work could focus on increasing the performance of this solution through data layout optimization and parallelizing the solver in order to cut down on the computation time. A best-case result here would allow for this solution to be used in the actual game as a separate mode that would allow players themselves the access to new, well-designed stages within a fraction of a second.

The future work could also include a comparison of the applied method with alternative methods for procedural generation. Modern methods using machine learning, for example, in LSTM or GAN networks, however, require much more expenditure on manual making of levels to prepare training and test datasets.

While the selection of variables constituting the final equation for weighted difficulty scoring looks satisfactory, there is space for deeper exploration of less obvious metrics such as the different effects of combining multiple mechanics in a single level and the relationship between the initial and final level states.

Another main conclusion of this study that could be learned from the process shown is to highlight places where the processing time is not crucial. While the runtime parts of game engines are thoroughly optimized, the tool side, which supports the designer's work, should rather focus on user convenience and maximum adjustment of the output data. PCG tools can take on as many operations as possible so that the runtime part should perform as few tasks as possible in order to run with the highest possible efficiency.

Acknowledgment

This work was supported by The National Centre for Research and Development within the project "From Robots to Humans: Innovative affective AI system for FPS and TPS games with dynamically regulated psychological aspects of human behaviour" (POIR.01.02.00-00-0133/16). We thank Mateusz Makowiec, Marcin Daszuta, and Filip Wróbel for assistance with methodology and comments that greatly improved the manuscript.

References

1. Andrzejczak, J., Osowicz, M., Szrajber, R.: Impression curve as a new tool in the study of visual diversity of computer game levels for individual phases of the design process. In: Lecture Notes in Computer Science, pp. 524–537. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-50426-7_39

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9

2. Aponte, M.V., Levieux, G., Natkin, S.: Measuring the level of difficulty in single player video games. Entertainment Computing **2**(4), 205–213 (jan 2011). https://doi.org/10.1016/j.entcom.2011.04.001
3. Ashlock, D., Schonfeld, J.: Evolution for automatic assessment of the difficulty of sokoban boards. In: IEEE Congress on Evolutionary Computation. IEEE (jul 2010). https://doi.org/10.1109/cec.2010.5586239
4. Browne, C.: Metrics for better puzzles. In: Game Analytics, pp. 769–800. Springer London (2013). https://doi.org/10.1007/978-1-4471-4769-5_34
5. Csikszentmihalyi, M.: Flow: The psychology of optimal experience. Harper & Row **45**(1), 142–143 (jan 1990). https://doi.org/10.1176/appi.psychotherapy.1991.45.1.142
6. Diaz-Furlong, H.A., Solis-Gonzalez, C.A.L.: An approach to level design using procedural content generation and difficulty curves. In: 2013 IEEE Conference on Computational Inteligence in Games (CIG). IEEE (aug 2013)
7. Feng, L., Wei, W.: An empirical study on user experience evaluation and identification of critical UX issues. Sustainability **11**(8), 2432 (apr 2019). https://doi.org/10.3390/su11082432
8. Hillman, S., Stach, T., Procyk, J., Zammitto, V.: Diary methods in AAA games user research. In: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems. ACM (may 2016). https://doi.org/10.1145/2851581.2892316
9. Holdsworth, J.J.: The nature of breadth-first search. Tech. rep., School of Computer Science, Mathematics and Physics, James Cook University (1999)
10. Jefferson, C., Moncur, W., Petrie, K.E.: Combination. In: Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11. ACM Press (2011). https://doi.org/10.1145/1982185.1982383
11. Kaplan, H.L.: Effective random seeding of random number generators. Behavior Research Methods & Instrumentation **13**(2), 283–289 (jan 1981). https://doi.org/10.3758/bf03207952
12. van Kreveld, M., Loffler, M., Mutser, P.: Automated puzzle difficulty estimation. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG). IEEE (aug 2015). https://doi.org/10.1109/cig.2015.7317913
13. Nishimoto, H., Hamada, A., Takai, Y., Goto, A.: Investigation of decision process for purchasing foodstuff in the "bento" lunch box. Procedia Manufacturing **3**, 472–479 (2015). https://doi.org/10.1016/j.promfg.2015.07.210
14. Parker, R.: The culture of permadeath: Roguelikes and terror management theory. Journal of Gaming & Virtual Worlds **9**(2), 123–141 (jun 2017). https://doi.org/10.1386/jgvw.9.2.123_1
15. Rogalski, J., Szajerman, D.: A memory model for emotional decision-making agent in a game. Journal of Applied Computer Science **26**(2), 161–186 (2018)
16. Sampaio, P., Baffa, A., Feijo, B., Lana, M.: A fast approach for automatic generation of populated maps with seed and difficulty control. In: 2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). IEEE (nov 2017). https://doi.org/10.1109/sbgames.2017.00010
17. Sarkar, A., Cooper, S.: Transforming game difficulty curves using function composition. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. ACM (may 2019). https://doi.org/10.1145/3290605.3300781
18. Togelius, J., Kastbjerg, E., Schedl, D., Yannakakis, G.N.: What is procedural content generation? In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11. ACM Press (2011). https://doi.org/10.1145/2000919.2000922

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_9