An Angry Birds Level Generator with Rube Goldberg Machine Mechanisms

Febri Abdullah*, Pujana Paliyawan*†, Ruck Thawonmas*, Tomohiro Harada*, Fitra A. Bachtiar‡
*Intelligent Computer Entertainment Laboratory, Ritsumeikan University, Japan
†Research Organization of Science and Technology, Ritsumeikan University, Japan
‡Faculty of Computer Science, Brawijaya University, Indonesia
ruck@is.ritsumei.ac.jp

***Abstract*—This study proposes a method for generating Angry Birds-like game levels featuring a domino effect generated based on Rube Goldberg Machine (RGM) mechanisms, which allow them to be completed by one shot of a bird. The proposed method generates a level by selecting predefined segments consisting of several objects arranged in a way that creates a domino effect among them. To increase the variability of generated levels, the proposed method procedurally generates a varying structure on the top of certain blocks in a predefined segment. Our results show that the proposed RGM generator is comparable to two existing generators, including the winner of the 2018 AIBIRDS Level Generation Competition, in terms of stability while it outperforms both baseline generators with respect to running time and an expressivity metric called "dynamic" which is introduced in this work to measure the time period where moving objects, including a shooting bird, reside in a given level. In addition, from the perspective on the destructive power of a shot, the proposed generator can generate levels featuring a successful domino effect with a high probability, in particular for levels with three to four segments.**

***Index Terms*—Angry Birds, Rube Goldberg Machine, Procedural Content Generation**

I. Introduction

A Rube Goldberg machine (RGM) [1], [2] is a machine that uses a chain reaction to accomplish a simple task in a very complicated manner. The concept of this machine was invented by Rube Goldberg, an American cartoonist often referred to as the "father of invention [3]" for his idea of a machine that once it starts, it is practically impossible for those who watch it to peel themselves away from the anticipation of what is coming next. Passing throughout almost a century, this idea lives in pop culture; Rube Goldberg's name has become searchable, hash-taggable, and at best viral [4]. Goldberg contraptions have flooded cultures around the world in commercials, contests, movies, and TV shows [3]. Undoubtedly, the RGM is hilarious by its nature [4], and it is fun to watch a good RGM [3].

With a goal to create gameplay that entertains not only players but also spectators who watch it, we embraced RGM ideas and applied them to level generation in "Angry Birds" or the likes. Namely, we designed a method for generating levels that feature a domino effect, allowing them to be completed by one shot of a bird. This paper is an extension with more details on implementation, of our entry to the Angry Birds Level Generation Competition at CoG 2019, of our previous short papers [5], [6] whose details are given in II-C; a process called structure generation is introduced to increase the variability of generated levels.

II. Literature Reviews

A. Rube Goldberg machines in Academia

An RGM [2] is a machine intentionally designed to perform a simple task in an indirect and over-complicated fashion. Often, these machines consist of a series of simple devices that are linked together to produce a domino effect, in which each device triggers the next one, and the original goal is achieved only after many steps. This concept is very popular, and many RGM contests have been held, e.g., in early 1987, Purdue University started the annual National Rube Goldberg Machine Contest [7], organized by the Phi Chapter of Theta Tau, a national engineering fraternity, and in 2009, the Epsilon Chapter of Theta Tau established a similar annual contest [8] at the University of California, Berkeley.

An idea of RGMs is widely adopted in education, particularly engineering education, and it is known for its effectiveness in motivating students. For example, Lei et al. [9] described how an RGM project can be integrated to an introductory electrical engineering course to trigger and maintain students' motivations. They described that RGM has an innovative, humorous and unconventional nature, and their project created a social environment that encourages intellectual engagement[1].

Acharya [10] applied RGM mechanisms in engineering education. The main focus was given to engineering design and microcontrollers in RGM mechanisms. The author claimed that the Rube Goldberg project provided a unique engineering design experience for students. Those students successfully accomplished the task by strictly following the given guideline and the recommended detailed engineering design and development approach. All required work products were submitted and presented within the deadline. Positive student feedback was obtained during and after the learning experience.

O'Connor [11] applied RGMs to a course designed to teach concepts of design and development of Web-based instructional materials. The author created a project named Rube-o-

[1]https://en.wikipedia.org/wiki/Typical_intellectual_engagement

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

Rama for teaching K-12 students with the Rube Goldberg's concept. From the perspective of education outcomes, the author claimed that the overall learning experience for all participants was much richer than one would find in a traditional classroom or a traditional room-to-room videoconferencing setting for distance education.

There exist some video games mainly featuring RGMs, for example, *Rube Works: The Official Rube Goldberg Invention Game* that bridges the worlds of gaming and education. However, to the best of our knowledge, there exists no work, besides our work, that applies the concept of RGMs to game level generation. Although the studies mentioned above were not directly related to video games, we believe that they together with our previous findings demonstrate the potential of RGM mechanisms in entertaining people who watch RGM contents.

B. Procedural Content Generation for Angry Birds

Procedural content generation (PCG) has increasingly gained attention as a promising solution that can reduce the cost of game development. Shaker et al. [12] described PCG as computer software that can create game contents on its own, or together with human players or designers. PCG techniques can be used for generating contents such as game levels or stages. Our work focuses on PCG for generating game levels on "Science Birds", a clone version of "Angry Birds" presented by Ferreira and Toledo [13] that has been widely employed for academic research.

"Science Birds" level generation is a popular theme of research, and an annual competition on this topic has been held by IEEE Conference on Games (CoG) since 2016 (then IEEE Conference on Computational Intelligence and Games) [14]. Ferreira and Toledo [13] proposed a level generator based on a genetic algorithm (GA), whose main objective is to minimize the total amount of object movement during simulation to create levels which do not fall. Kaidan et al. [15] modified the above GA-based generator such that it automatically adjusts its parameters according to the player's gameplay results and hence adapts to the player's skills. Stephenson and Renz proposed methods [16]–[18] that can create a wide variety of stable levels without the aid of predefined level components. From the perspective of system evaluation, these studies mainly focus on the stability and/or playability of generated levels while effects on the player's emotions was not much considered.

Our group has been focusing on building game levels that promote fun and relaxing gameplay. Jiang et al. [19] proposed a pattern-struct approach that generates each time a different appearance for an alphabetical letter; a level generator, called Funny Quotes, was our first work focusing on fun and was the winner of the Fun Track of the CIG2016 Angry Birds level generation competition. Yang et al. [20] proposed two implementations for promoting mental well-being through smiling: in the first implementation, the player must smile in order to erase fog that hides some parts of the game stage, while in the second implementation, a TNT (an in-game explosive) that can be exploded by the player's smile is introduced. Later on, Yang et al. [21] found that proper placement of a TNT in the game contributes to an increase in the interestingness of generated gameplay, so they presented and compared several techniques to optimize a TNT's placement to promote the spectator's emotions. Xu et al. [22] presented mechanisms for generating game levels in the form of pixel images and promoting players to smile to enhance the explosive power of a special bird; their system could not only increase the positive affect but also decrease the negative one of the players.

C. Angry Birds with RGM levels

RGM-based level generation is a recently established project in our group. This project is inspired by supporting evidence from non-game-related studies that RGMs have strong potential in entertaining people who watch them [9]–[11]. Former studies on PCG for Angry Birds did not emphasize evaluation of generated levels in terms of player emotions [13], [15]–[18]. On the other hand, previous studies targeting emotion promotion by our group evaluated systems only in terms of either playability or user evaluation. For example, Jiang et al. [19] evaluated the playability of generated levels and showed that levels with various difficulty (i.e., easy to hard) can be created, but user evaluation was not conducted, while Yang et al. [20], [21] and Xu et al. [22] focused on user evaluation, but overlooked the playability aspect. On the contrary, in this RGM project, we evaluate the effectiveness of RGM-based level generation in terms of both user evaluation (effects on the player's affect [5] and the player's working memory [6]) and playability (this paper).

For more details, our first work presenting RGM mechanisms was evaluated in terms of the effects of on spectators' emotions [5]. Two sets of gameplay videos were compared: one consisting of only perfect shots and the other of only imperfect shots, i.e., those that complete and those that do not complete a level in one shot, respectively. A self-report questionnaire called Positive and Negative Affect Schedule (PANAS) [23] was used to assess emotions of the spectators. The results showed that the set of perfect-shot videos leads to higher positive affect and lower negative affect of the spectators in the same series of generated RGM levels.

The second work [6] investigated whether and to which degree watching live streaming of Angry Birds gameplay with RGM levels could affect the spectator's working memory (WM). WM is measured using a task called the N-back task [24]. The results showed a possibility that watching Angry Birds featuring RGMs can increase the spectator's WM performance.

In both studies [5], [6], RGM levels were generated using a predefined set of objects called segment, each constructed by several types of primitive blocks, platforms (a type of surfaced object, apart from the ground of the level, whose gravity is not applicable and can not be destroyed), and TNTs arranged manually beforehand to achieve the RGM effect. Each segment has a specific object, called input object, that must be destroyed to trigger the collapse of the segment

and make another specific object, called output object, move in a particular direction to a specific destination. The next segment is then chained using the information on the previous segment's output destination. This process is repeated until the desired number of segments or the maximum boundary of a level is reached. In theory, this mechanism allows a level to be completed with just one shot of a bird by creating a domino effect chained from the first to the last segment.

This study extends the RGM level generator from our previous work [5] [6] by improving the variability of the generated levels, which was said to be an important factor for keeping the experience of players fresh [25]. To increase the content variability, we propose a method that generates a structure on each predefined segment. We define a structure as a set of objects procedurally generated on a segment.

III. RGM Level Generator

This section describes how RGM levels are generated. The first part of this section is about RGM segments as components of an RGM level. The second part explains the algorithm for level generation. The last part proposes the algorithm for generating a structure in a segment.

A. RGM Segments

A new parameter called "base blocks" is added to specify the block type, material type, and position of one or several blocks upon which a structure is procedurally generated. We apply a rule-based algorithm to generate not only a relatively stable structure, but also a structure that lies outside the perfect-shot projectile trajectory of the bird and does not reside outside of the boundary of its segment.

To achieve a desired domino effect, we employ several variables for each segment as follows:

1) Input direction: the direction of an incoming object (the shot bird for the first segment or the output object from the preceding segment) that will trigger the input object. There are four types of direction: "left", "right", "down", and "up".
2) Output direction: the direction of the output object's movement after it is triggered. It has four types of direction similarly to the input direction.
3) Output destination: the x and y coordinates representing the new position of the output object after it is triggered; in other words, it indicates the position where the output object will be moved to, which is the average value among multiple simulations each with a different structure, to trigger the input object of the subsequent segment.
4) Segment size: the maximum width and height of the segment box within which all its objects must reside before the input object is triggered
5) Base blocks: a block or set of blocks – newly introduced in this work – serving as the base for placing a procedurally generated structure

Note that a segment may not have base blocks, but no structure will be procedurally generated for this kind of segment.

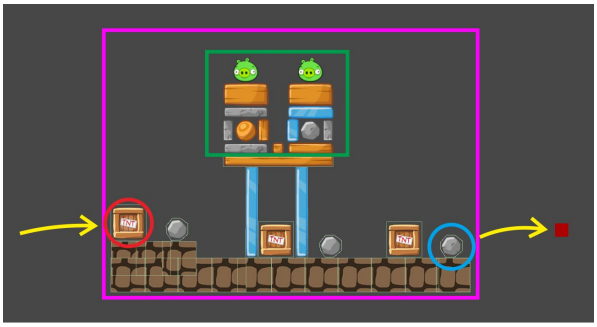


Figure 1: An example of an RGM segment with the "left" input and "right" output directions. A procedurally generated structure added to the current predefined segment is shown within the green rectangle. The segment's size is shown by the magenta rectangle. The input object (a TNT) is indicated by the red circle. The blue circle shows the output object. Several frames after the input object is destroyed, the output object will be relocated at the output destination (red box).

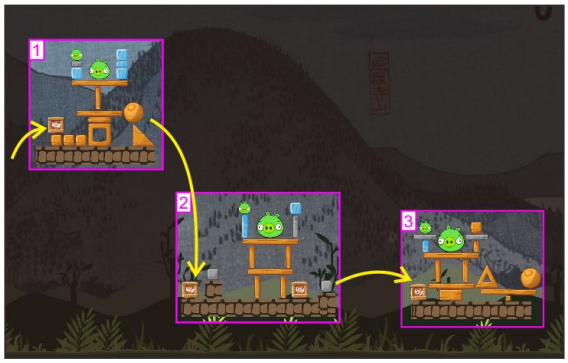
B. Level Generation

An RGM level is procedurally generated as a combination of segments containing various objects. As described earlier, a segment has a specific object (input object) that needs to be destroyed to trigger the RGM mechanism which eventually results in the movement of another object (output object). Based on this idea, we generate an RGM level by connecting several segments to create a domino effect among them. Algorithm 1 describes how an RGM level is generated.

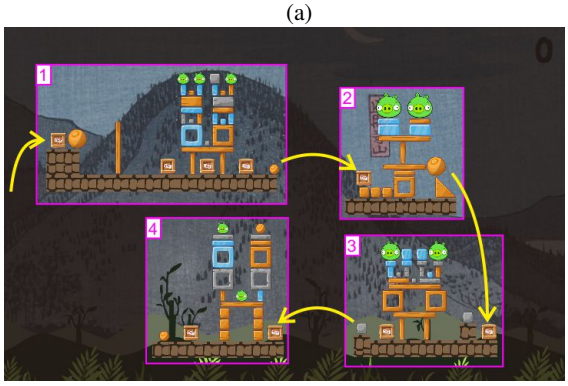
Algorithm 1 Level Generation Algorithm

```
1: inputDir ← left
2: while c < count do
3:     segment ← SelectSegment(inputDir)
4:     GenerateStructure(segment)
5:     inputDir ← Opposite(segment.outputDir)
6:     level.Add(segment)
7:     c ← c + 1
8: end while
9: Refine(level)
```

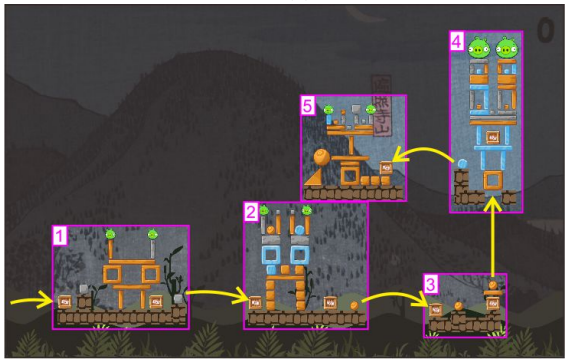
The algorithm starts with setting the initial value of *inputDir* to "left" (line 1). This is done because in this game typically a level places the slingshot at the left-most side. Our algorithm will randomly select a predefined segment having the input direction of *inputDir* from the set of predefined segments and assign the selected segment to the *segment* variable. The aforementioned segment selection is done by the *SelectSegment* function (line 3) subject to the constraint that a candidate segment does not overlap with the existing segments in the level. Note that the position of a segment of interest is determined by the output destination of the preceding one, namely, a selected segment must be positioned such that its input object's position is the same as the output destination of the preceding segment. To increase the variability of the



(a)



(b)



(c)

Figure 2: The top, middle, and bottom images are examples of RGM levels with 3, 4, and 5 generated segments, respectively.

selected segment, the *GenerateStructure* function (line 4) will generate a structure on it (more details of this algorithm are given in the next subsection). Then the value of *inputDir* will be updated. The algorithm will stop if the number of segments reaches a desired value (cf. Fig. 2 for two RGM levels with different numbers of segments). Finally, the *Refine* function will adjust the segments' position to prevent them from overlapping with the level's ground and ensure that they are not too close to the slingshot.

C. Structure Generation

Here we introduce the structure generation process to improve the segment variability of our previous RGM level generator [5] [6]. A structure is automatically generated on top of certain blocks in a segment. Algorithm 2 describes how such a structure is generated.

Algorithm 2 Structure Generation Algorithm

$minPos$ //minimum position for generating blocks
$maxPos$ //maximum position for generating blocks

1: $baseLayer \leftarrow segment.baseBlocks$
2: **while** !IsOutOfBounds($structure$) **do**
3: $availLayers \leftarrow$ GenerateMerged($baseLayer, minPos, maxPos$)
4: **if** $availLayers.count \leq 0$ **then**
5: $availLayers \leftarrow$ GenerateUnmerged($baseLayer, minPos, maxPos$)
6: **end if**
7: $selected \leftarrow$ GetRandom($availLayers$)
8: $structure$.Add($selected$)
9: $baseLayer \leftarrow selected$
10: **end while**
11: DistributePigs($structure$)
12: AddDecorations($structure$)

A structure is generated as a set of several layers stacked on top of each other. A layer consists of blocks of the same type, among those shown in Fig. 3, but with any kinds of available materials. The blocks in a layer are arranged symmetrically according to its base layer, the layer below it. The algorithm will stop if there is no space available for generating a new layer (line 2). Some primitive blocks available in the game are excluded in this process; more specifically, those that have shapes hard to stack or sizes too small, and henceforth they are called decorations (cf. Fig. 3). Figure 4 illustrates the generation process of a four-layered structure.

When a base layer of interest has more than one block, the *GenerateMerged* function (line 3) will try to merge them into a number of groups, each consisting of two blocks. The same function will search in the block database if there is any block whose length is greater than the distance of the two blocks for each group; if there exists multiple candidates, one of them will be randomly selected for being placed on top

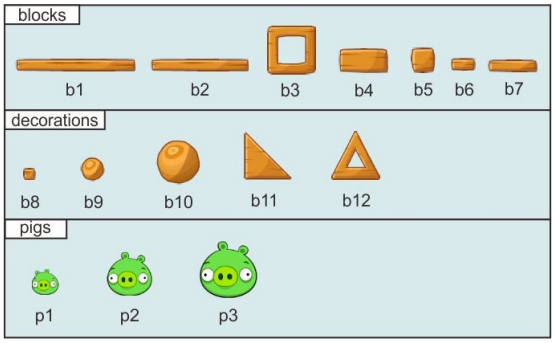
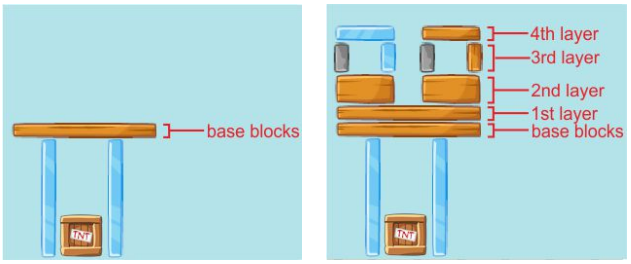


Figure 3: The primitive objects used as the components to generate a structure.



(a) (b)

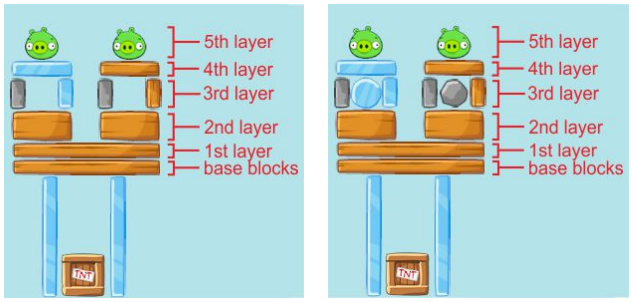
Figure 4: The left image illustrates a base segment. The right image shows the same base segment with a generated four-layered structure.

of the group. The function will be terminated if there is no such block. We use this function because we want to avoid the generated structure getting wider, which makes it more unstable. An illustration is shown in Fig. 4b where the four blocks in the 3rd layer are merged into two groups.

The *GenerateUnmerged* function (line 5) will undo the grouping by the *GenerateMerged* function and place two blocks on top of each block in the base layer. An illustration is shown in Fig. 4b where two blocks are placed on top of the 2nd layer.

The *DistributePigs* function (line 11) will exhaustively search for an empty space to place pigs on each generated layer. An illustration of the *DistributePigs* function can be seen from Fig. 5 where the pigs form the 5th layer.

As previously mentioned, we exclude decorations in the process of generating layers because of their shapes and sizes. They will serve as additional components to increase the variety of a structure. The *AddDecoration* function will search for any empty space left in the structure that can be used to place some of decorations. Figure 5b shows two decorations placed between the empty spaces left in the 3rd layer in Fig. 5a.



(a) (b)

Figure 5: The left image shows the additional 5th layer of pigs. The right one shows the same segment with generated decorations in the 3rd layer.

IV. Experiment and Results

We evaluated generated levels by using four factors: stability, running time, expressivity, and perfect-shot rate. We also compared the stability, running time, and expressivity of our generator with a baseline generator (Baseline), provided by the AIBIRDS COG 2019 Level Generation Competition[2], and IratusAves (MSGv2.0), the winning generator of the AIBIRDS 2018 Level Generation Competition developed by Stephenson and Renz[3] [18]. We set the parameters of all the generators as follows: 250 levels to generate, no forbidden block or material, and the time limit to generate levels of 30 minutes. For Baseline and IratusAves, the number of pigs is set from 3 to 10. We generated RGM levels using different numbers of segments: 3, 4, and 5 segments (denoted as RGM_3, RGM_4, and RGM_5, respectively), 250 levels each. In addition, Table I shows the system specification used in our experiment.

Table I: System specification used our experiment.

System Operation	Windows 10 Pro 64-bit (10.0, Build 17134)
Processor	Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50Ghz
Memory	16384 MB RAM
Graphic Card	NVIDIA Quadro K2000

A. Stability Analysis

The stability of a level was measured through a simulation process where no shooting takes place. The simulation runs for 10 seconds for each level and is started after there is no detected movement in any pig, block, or TNT. We classify a level as stable if there is no block, pig, or TNT being destroyed due to the in-game gravity during simulation. The stability of each generator is calculated as the percentage of stable levels in all 250 generated levels.

The results of the stability evaluation are shown in Table II. From this table, our RGM levels have high stability $(> 90\%)$ for all the settings (RGM_3, RGM_4, and RGM_5). In addition, RGM_3 has the stability of 97.60% higher than that of Baseline (95.60%) and nearly equal to that of IratusAves (98.80%). Note that the stability of our RGM level decreases as the number of segments increases because having an unstable segment in a level always causes the level to be unstable; the more segments the more chance a level will become unstable.

B. Running Time Analysis

The running time factor is used to measure how fast a generator generates a specific number of levels. We define running time as the time required (in seconds) for a generator to generate 250 levels. As shown in Table II, compared to Baseline (81.12s) and IratusAves (2003.91s), our RGM generator runs much faster for all the settings, i.e., 2.45s, 3.83s, and 6.13s for RGM_3, RGM_4, and RGM_5, respectively.

[2]https://aibirds.org/Level-Generation/LevelGeneratorBaseline-2.0.zip
[3]https://github.com/stepmat/IratusAves

Table II: Stability and running time comparisons between Baseline, IratusAves, RGM_3, RGM_4, and RGM_5.

	Stability	Running time
Baseline	95.60%	81.12s
IratusAves	98.80%	2003.91s
RGM_3	97.60%	2.45s
RGM_4	94.40%	3.83s
RGM_5	92.40%	6.13s

Due to the structure generation process on each segment, the running time of the proposed generator increases as the number of segments increases.

C. Expressivity Analysis

Expressivity metrics have been used to describe how the generated contents are expressed in several aspects [13], [16]–[18]. We use four existing expressivity metrics [16], [17], i.e., frequency, linearity, density, and leniency, together with a metric called "dynamic" newly introduced in this study. Below are results for each of the five metrics in use.

1) Frequency: The frequency metric is used to express block distribution in a generated level. For each of the 12 types (cf. blocks and decorations in Fig. 3), its block ratio is calculated over all blocks on a level. The frequency of a block or a decoration is defined as the mean value of the block ratios for all 250 generated levels. Its value ranges from 0 to 1; 0 indicates that a block type of interest is not used on any levels, and 1 indicates that all the levels only consist of it.

Fig. 6 shows the frequency of each block type for each generator where the names of blocks are shown in Fig. 3. Compared to IratusAves which frequently selected "b6" and "b8", our RGM generator rarely selected these types of blocks and preferably selected bigger blocks such as "b1" and "b5". We also found that all the generators rarely selected "b10", "b11", and "b12". In our case, the proposed RGM generator rarely selected "b8", "b10", "b11", and "b12" because they were excluded from the layer generation process of a segment due to their shapes and sizes. However, "b9" has a high frequency, even though it was classified as a decoration because

Table III: Expressivity comparisons (mean and standard deviation) over 250 generated levels between Baseline, IratusAves, RGM_3, RGM_4, and RGM_5.

	Density	Leniency	Linearity
Baseline	$24.21\% \pm 5.14\%$	78.02 ± 21.34	0.07 ± 0.08
IratusAves	$32.37\% \pm 4.67\%$	94.28 ± 23.69	0.04 ± 0.05
RGM_3	$13.19\% \pm 3.56\%$	46.56 ± 12.94	0.30 ± 0.25
RGM_4	$11.31\% \pm 3.46\%$	62.30 ± 15.13	0.31 ± 0.24
RGM_5	$10.53\% \pm 3.26\%$	78.33 ± 16.53	0.30 ± 0.23

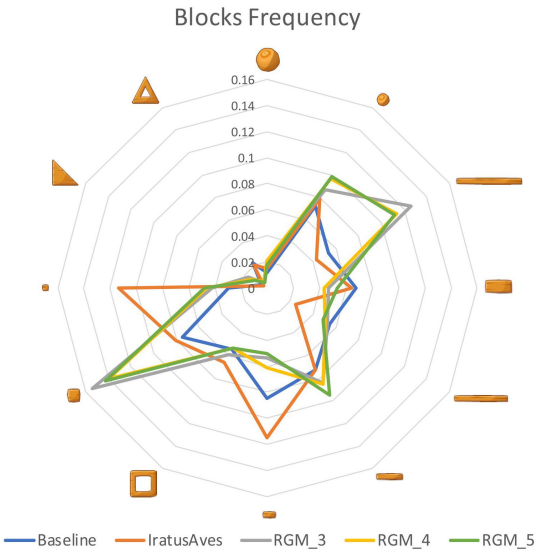


Figure 6: The frequency of each block type for each generator.

most of our predefined segments contain this type of block. In addition, even though our proposed generator has a capability of generating a varying structure on each segment, it still relies on predefined segments, which makes the frequency of a block type of interest roughly the same regardless of the number of segments.

2) Density: The density metric expresses a level's compactness. A level having low density indicates that it has more amount of empty space than other levels with a higher density. We define density of a level as the percentage of the amount of space occupied by all objects in the level's whole space. Table III shows that our RGM generator has lower mean density than Baseline and IratusAves. This is because a generated segment needs more empty space to ensure that its output object can move in a clear path, which also explains why the density value decreases as the number of segments increases.

3) Leniency: Leniency is used to express the difficulty of a level. We define leniency using the numbers of pigs (p) and blocks (b) as follows:

$$Leniency = 2p + b \tag{1}$$

As shown in Table III, our RGM generator shows less mean leniency for all the settings compared to IratusAves. This is because space provided in its predefined segments for placing blocks, to form a structure, and pigs is more limited. However, RGM_5 has a slightly higher value than Baseline. In addition, the leniency value of the proposed RGM generator increases as the number of segments increases since a level with a higher number of segments has more pigs and blocks.

4) Linearity: Linearity expresses the variety of each generated content. Previous studies defined this metric in slightly different ways. Ferreira and Toledo [13] defined linearity as the average height of all the blocks that are generated in a level and represented as columns. Stephenson and Renz [16] defined linearity based on a level's width and height. Another

study conducted by them [17], [18] measured linearity by performing a linear regression which takes the positions of all the generated blocks, platforms, and pigs (plus TNTs in the latter work) as data points. We also use the linearity metric discussed in the latter work of Stephenson and Renz [18] where the resulting R^2 value, obtained by applying a linear regression analysis to the said data points, is used to express the linearity. A low value of R^2 indicates that a generated level has low linearity or, in other words, all generated objects' positions in the level do not form a straight line. From the results in Table III, our RGM generator in all the settings has a higher R^2 mean value than both Baseline and IratusAves. These results indicate that our generated RGM levels have higher linearity due to a constraint imposed by the input and output directions of the predefined segments.

5) Dynamic: Findings from one of our previous studies [21] indicate that objects' movement or the duration of the movement caused by a TNT explosion plays an important role in helping spectators lower their negative mood. However, such an effect can not be directly expressed by the existing four expressivity metrics. Hence, we propose a new one called dynamic in the following.

We define dynamic of a level as the time duration (in seconds) during the timing when a bird is shot and the timing when moving objects are no longer detected in the level. We consider all TNTs and pigs in a level as the shooting targets for this measurement. And we run several simulations where the targets are aimed one by one—by utilizing the bird's starting position, target position, and game's gravity—and take the maximum duration as the dynamic value.

The mean value of dynamic for all 250 generated levels from each generator is shown in Table IV. It can be seen that the generated RGM levels have a higher mean value than both Baseline and IratusAves. This result is caused by the collapse of each segment which triggers a domino effect among them. It can also be noted that the dynamic value increases as the number of segments increases because more segments will collapse one after another resulting in extending the duration of the whole level's collapse.

Table IV: Dynamic comparisons (mean and standard deviation) over 250 generated levels between Baseline, IratusAves, RGM_3, RGM_4, and RGM_5.

	Dynamic
Baseline	$12.48s \pm 2.10s$
IratusAves	$12.20s \pm 2.44s$
RGM_3	$12.78s \pm 1.88s$
RGM_4	$13.46s \pm 2.06s$
RGM_5	$14.10s \pm 2.06s$

D. Perfect-shot Evaluation

We define a perfect shot[4] as a single shot that completes a given level by destroying all the pigs therein. The idea behind using this factor is to evaluate the performance of generated RGM levels by featuring a domino effect among segments. We evaluated the perfect-shot rate on 500 generated RGM levels for each setting. For each level, we calculate the angle and power of bird-shooting to always aim and shoot at the input object of the first segment. A level is considered "perfect" if it can be completed with just one shot of a bird; note that in the experiment, the game always proceeded to the next level regardless of whether or not the current level is completed after the first shot.

Table V shows the perfect-shot rate result of each setting. Although there is room for improvement, e.g., employing a simulation process to ensure each generated level can be completed with a perfect shot, we consider that our RGM levels have a high perfect-shot rate on both RGM_3 (82.6%) and RGM_4 (74.6%). Note that, as mentioned in III-A, since the output destination for each segment is derived beforehand from multiple samples (each with a different procedurally generated structure in a given segment), there exists an error in the actual position where the output object will be moved to; this kind of error accumulates, and hence the perfect-shot rate drops as the number of segment increases.

Table V: The perfect-shot rate of RGM levels in each setting.

	Perfect-shot rate
RGM_3	82.8%
RGM_4	74.6%
RGM_5	68.6%

V. Conclusions and Future Work

From the experimental results, our RGM levels have high stability comparable to both existing generators although we plan to increase the stability of each RGM level by employing beforehand a simulation process to dismiss any unstable levels or more accurate calculation to estimate the stability of each generated structure. Our proposed generator performs much faster in generating levels. In terms of the four existing expressivity metrics, our proposed generator still has room for improvements in the following.

First, from the frequency results, the proposed generator tends to has the same block distribution regardless of the number of segments employed. Second, from the density perspective, as the number of segments increases, the density value decreases because our proposed generator needs more empty space to ensure the output object of a segment has a clear path to destroy its subsequent segment. Third, the leniency results show that our proposed method has a lower mean value compared to the latest winning generator due to having more limited space, for placing blocks and pigs, in the predefined segments. Fourth, according to the linearity

[4]https://youtu.be/WBKsBG5s-o0

results, the RGM levels have higher linearity due to a limited number of combinations of the input and output directions of the predefined segments. For future improvements, we plan to generate RGM levels without relying on predefined segments because almost all unsatisfying results with respect to the existing expressivity metrics apparently stemmed from the use of predefined segments.

However, the results from our newly introduced dynamic metric show that our generated RGM levels have a higher mean value than both of the existing generators. These results indicate that the domino effect featured in the proposed generator has successfully extended the interval of having moving objects in generated levels. For future improvements, we plan to propose better methods to accurately predict the movement of the output object of a segment. By doing this, we assume that the empty space between a pair of connected segments can be minimized, thus a larger number of segments can be placed in a given level; noting that the dynamic value increases as the number of segments increases.

From the perfect-shot evaluation, those generated with the RGM_3 and RGM_4 settings have a high perfect-shot rate, indicating that there is a high probability to create a domino effect among their segments. We plan to improve the perfect-shot rate by performing the same method as in our plan to increase the level's stability. We assume that simulating each segment's destruction process or accurately predicting objects' movement can guarantee a high perfect-shot rate on each RGM level.

ACKNOWLEDGMENT

This research was supported in part by Grant-in-Aid for Scientific Research (C), Number 19K12291, Japan Society for the Promotion of Science, Japan. Special thanks are also given to Zhou Fang, Junjie Xu, and Dawang Chen for their amazing work in designing the base segments in this study.

REFERENCES

[1] Rube Goldberg machine, en.wikipedia.org/wiki/Rube_Goldberg_machine (accessed 4 June 2019)

[2] MF. Wolfe, R. Goldberg, Rube Goldberg: Inventions!, Simon and Schuster, 2000 Nov 20, pp. 1-195.

[3] The best Rube Goldberg machines, www.digitaltrends.com/cool-tech/best-rube-goldberg-machines/ (accessed 4 June 2019)

[4] The official Rube Goldberg website, www.rubegoldberg.com/rube-the-artist/ (accessed 4 June 2019)

[5] F. Abdullah, C. Yang, P. Paliyawan, R. Thawonmas, T. Harada, F.A. Bachtiar, "Promoting Emotion With Angry Birds-like Gameplay on Rube Goldberg Machine Levels," IEEE ICCE-Asia 2019, 2019 Jun 12-14 (accepted)

[6] F. Abdullah, C. Yang, P. Paliyawan, R. Thawonmas, T. Harada, F.A. Bachtiar, "Effect of Angry Birds-like Live Streaming on Working Memory," 2019 SEGAH, 2019 Aug 5-7 (accepted)

[7] Rube Goldberg Contest - Rube Goldberg Machine, www.purdue.edu/newsroom/rubegoldberg/index.html (accessed 4 June 2019)

[8] Theta Tau, en.wikipedia.org/wiki/Theta/_Tau (accessed 4 June 2019)

[9] CU. Lei, HK. So, EY. Lam, KK. Wong, RY. Kwok, CK. Chan, "Teaching Introductory Electrical Engineering: Project-based Learning Experience," 2012 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), 2012 Aug 20-23, pp. H1B-1, IEEE.

[10] S. Acharya, A. Sirinterlikci, "Introducing Engineering Design Through an Intelligent Rube Goldberg Implementation," Journal of Technology Studies, 2010 Oct 1, vol. 36, no. 2, pp. 63-72.

[11] D. O'Connor, "Application Sharing in K-12 Education: Teaching and Learning With Rube Goldberg," TechTrends, 2003 Sep 1, vol. 47, no. 5, pp. 6-13.

[12] N. Shaker, J. Togelius, M. J. Nelson, "Procedural Content Generation in Games: A Textbook and an Overview of Current Research", Berlin: Springer, 1st edition, 2016 Oct 19, pp. 1-237.

[13] L. Ferreira, C. Toledo, "A Search-based Approach for Generating Angry Birds Levels," 2014 IEEE Conference on Computational Intelligence and Games, 2014 Aug 26, pp. 1-8, IEEE.

[14] MJ. Stephenson, J. Renz, X. Ge, LN. Ferreira, J. Togelius, P. Zhang, "The 2017 AIBIRDS Level Generation Competition," IEEE Transactions on Games, 2018 Jul 12, pp. 1-10.

[15] M. Kaidan, T. Harada, CY. Chu, R. Thawonmas, "Procedural Generation of Angry Birds Levels With Adjustable Difficulty," 2016 IEEE Congress on Evolutionary Computation (CEC), 2016 Jul 24, pp. 1311-1316, IEEE.

[16] M. Stephenson, J. Renz, "Procedural Generation of Complex Stable Structures for Angry Birds Levels," 2016 IEEE Conference on Computational Intelligence and Games (CIG), 2016 Sep 20, pp. 1-8, IEEE.

[17] M. Stephenson, J. Renz, "Procedural Generation of Levels for Angry Birds Style Physics Games," The Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), 2016 Sep 19, pp. 225-231.

[18] M. Stephenson, J. Renz, "Generating Varied, Stable and Solvable Levels for Angry Birds Style Physics Games," 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017 Aug 22, pp. 288-295, IEEE.

[19] Y. Jiang, T. Harada, R. Thawonmas, "Procedural Generation of Angry Birds Fun Levels Using Pattern-Struct and Preset-Model," 2017 IEEE Conference on Computational Intelligence and Game (CIG), 2017 Aug 22, pp. 154-161, IEEE.

[20] C. Yang, Y. Jiang, P. Paliyawan, T. Harada, R. Thawonmas, "Smile With Angry Birds: Two Smile-interface Implementations," 2018 Nicograph International (NicoInt), 2018 Jun 28, p. 80, IEEE.

[21] C. Yang, P. Paliyawan, T. Harada, R. Thawonmas, "Blow Up Depression With In-Game TNTs," 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE), 2018 Oct 9, pp. 820-821, IEEE.

[22] J. Xu, Y. Okido, S. Sae-Lao, P. Paliyawan, R. Thawonmas, T. Harada, "Promoting Emotional Well-being With Angry Birds-like Gameplay on Pixel Image Levels," 2019 SEGAH, 2019 SEGAH, 2019 Aug 5-7 (accepted).

[23] D. Watson, LA. Clark, A. Tellegen, "Development and Validation of Brief Measures of Positive and Negative Affect: the Panas Scales," Journal of personality and social psychology, 1988 Jun, vol. 54, no. 6, pp. 1063-1070.

[24] MJ. Kane, AR. Conway, TK. Miura, GJ. Colflesh, "Working Memory, Attention Control, and the N-Back Task: a Question of Construct Validity," Journal of Experimental Psychology: Learning, Memory, and Cognition, 2007 May, vol. 33, no. 3, pp. 615-622.

[25] P. Lopes, A. Liapis, GN. Yannakakis, "Sonancia: Sonification of Procedurally Generated Game Levels," The 1st computational creativity and games workshop, 2015 Mar 18, pp. 1-6.