DALHOUSIE UNIVERSITY

Breadth First Search in Procedural Content Generation for Angry Bird Games

by

Chang Ye

A thesis submitted in fulfillment for the degree of Bachelor of Engineering

in the
Faculty of Computer Science

April 2018

Declaration of Authorship

I, Chang Ye, declare that this thesis titled, Breadth First Search in Procedural Content Generation for Angry Bird Games and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

DALHOUSIE UNIVERSITY

Abstract

Faculty of Computer Science

Bachelor of engineering

by Chang Ye

Ever since computer games were invented, the number of person-months that go into the development of a successful commercial game has increased. It is now common for a game to be developed by hundreds of people over a period of a year or more. Humans are expensive and slow, and this leads to a situation where fewer games are profitable, and fewer developers can afford to develop a game, leading in turn to less risk-taking and less diversity in the games marketplace.

Many of the costly employees necessary in this process are designers and artists rather than programmers. Procedure content generation could help companies to produce games faster and cheaper while preserving quality. Moreover, it could augment the creativity, create content-rich games.

The purpose of this thesis is to find a way to create diverse content in the famous game "Angry Birds" but reduce time and cost. In this thesis, we implement Procedural Content Generation by using breadth first search and K-means clustering. The breadth first search is used to search all possible blocks. K-means clustering uses these candidates to select most distinct structures. As a result, we found that we could generate various fun levels with a predefined silhouette in a much less time and cost.

Acknowledgements

I am grateful to Dr. Malcolm Heywood, Dr. Andrew McIntyre in the Faculty of Computer Science. I am extremly thankful and indebted to them for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

Contents

Declaration of Authorship **i**

Abstract **ii**

Acknowledgements **iii**

List of Figures **vi**

1 Introduction **1**

2 Background **2**

2.1 Angry Birds 2

2.2 The Angry Birds AI Competition 3

2.3 Previous studies 4

2.4 Goal 6

3 Methodology **7**

3.1 Method 7

3.1.1 Breadth first search 7

3.1.2 K-means clustering 8

3.2 Structures generation 8

3.2.1 Define silhouette 8

3.2.2 Generating Row Candidates 9

3.2.3 Compatible check 9

3.2.3.1 Stability check 9

3.2.3.2 Limitation and Overlap check 11

3.2.4 Candidates pruning 11

3.2.4.1 Vectorization 11

3.2.4.2 Clustering 12

3.2.5 Generate overall structure 13

4 Experiment results **14**

4.1 Variation in one silhouette 14

4.2 Variation in same silhouette with different section size 16

4.3 Variation in different silhouette 18

4.4 Result analyze 19

5 Conclusion and future work 21

List of Figures

2.1 Screenshot of a level from the Angry Birds game 2
2.2 The thirteen different block type available 3
2.3 Example of a level xml file 4

3.1 Breadth first search 7
3.2 Example of a silhouette 9
3.3 Example of stable blocks 1 10
3.4 Example of stable blocks 2 10
3.5 Example of stable blocks 3 11
3.6 Example of stable blocks 4 11
3.7 Example of section vectorization 12
3.8 Mean silhouette value in different K value 13

4.1 Result of generation 1. 14
4.1 Result of generation 1. 15
4.2 Result of generation 1. 16
4.3 Result of generation 2. 17
4.4 silhouette 2 18
4.5 Result of generation 3. 18
4.6 Mean silhouette value in different K value 19
4.7 Example of row quantification 19

Introduction

Nowadays, due to continuing rise of costs and scale expansion in game development, procedural content generation (PCG) has gained much attention as a promising area that its findings can solve such issues. PCG can be used for generating contents such as maps or models in a much less time.

According to Shaker et al. [1], PCG refers to computer software that can create game content on its own, or together with one or many human players or designers. There are three approaches to procedural content generation: constructive, simple generate-test and search-based[2]. A constructive algorithm generates the content once and is done with it. It needs to ensure that the content is correct or at least "good enough" during performing operation or sequence of operations. A generate-and-test algorithm incorporates both a generate and a test mechanism. After a candidate content instance is generated, it is tested according to some criteria. If the test fails,all or some of the candidate content is discarded and regenerated,and this process continues until the content is good enough

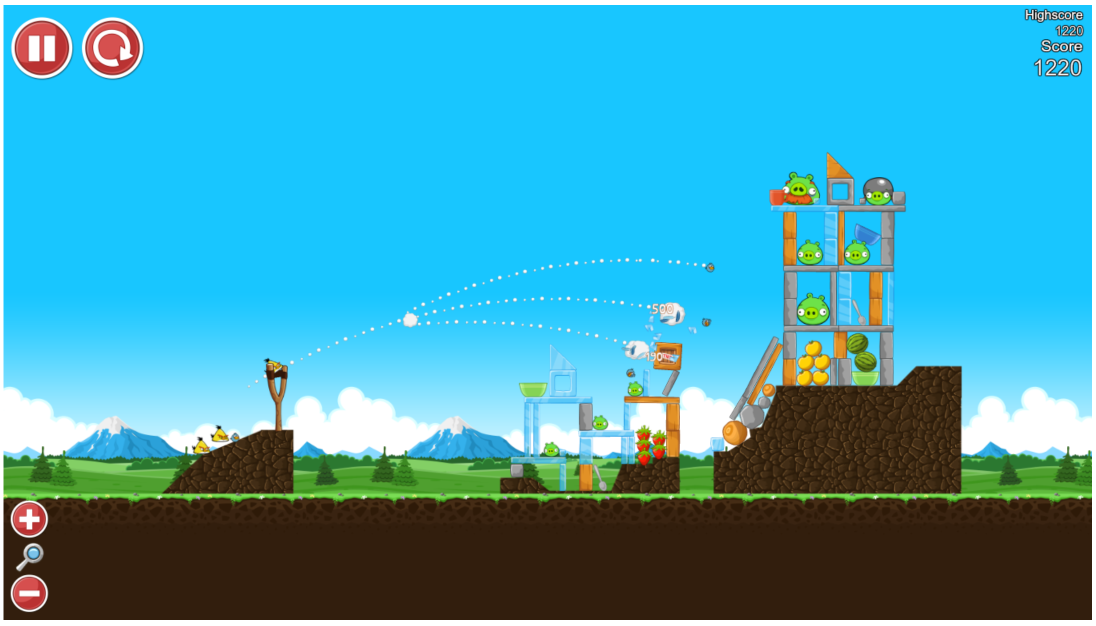
In this study, we aim to design a generator can automatically generate fun levels for the famous game Angry Birds. The key feature of our generator is it creates different structures in small amount of time, based on a predefined silhouette. We implemented breadth first search and K-means clustering, adopted constructive approach to achieve this goal.

Background

2.1 Angry Birds

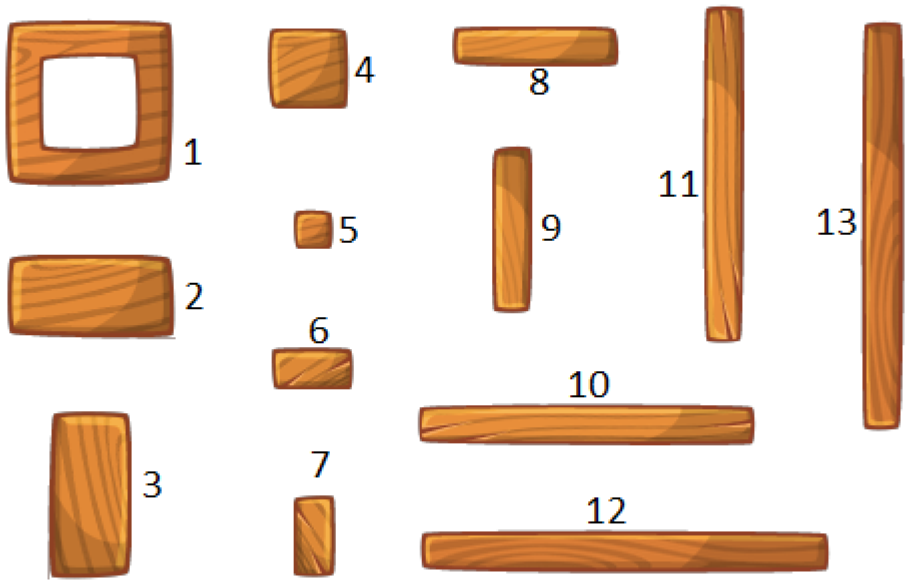
Angry Birds is a famous action-puzzle game developed by a Finnish company called Rovio Entertainment. The first Angry Birds game in the series was initially released in December 2009. It is a physics-based puzzle game where the player uses a slingshot to shoot birds at structures composed of blocks, with pigs placed within or around them. The player's objective is to kill all the pigs using the birds provided. A typical Angry Birds level, as shown in Figure 2.1, contains a slingshot, birds, pigs, and a collection of blocks arranged in one or more structures. The ground is usually flat but can vary in height for certain difficult levels. Each block in the game can have multiple different shapes as well as being made of several possible materials.

FIGURE (2.1) Screenshot of a level from the Angry Birds game



Because Angry Birds is a commercial game developed by Rovio Entertainment who do not provide an open-source version of their code, we used a Unity-based clone of the Angry Birds game called weird aliens. This clone provides many of the necessary elements to simulate our procedurally generated structures in a realistic physics environment. There are currently eight different rectangular blocks available, of which five can be rotated ninety degrees to create a new block type. This gives a total of thirteen different block variants with which to build our structure, see Figure 2.2. Each block is also assigned one of three materials (wood, ice or stone), bringing the number of possible options to thirty nine.

FIGURE (2.2) The thirteen different block type available

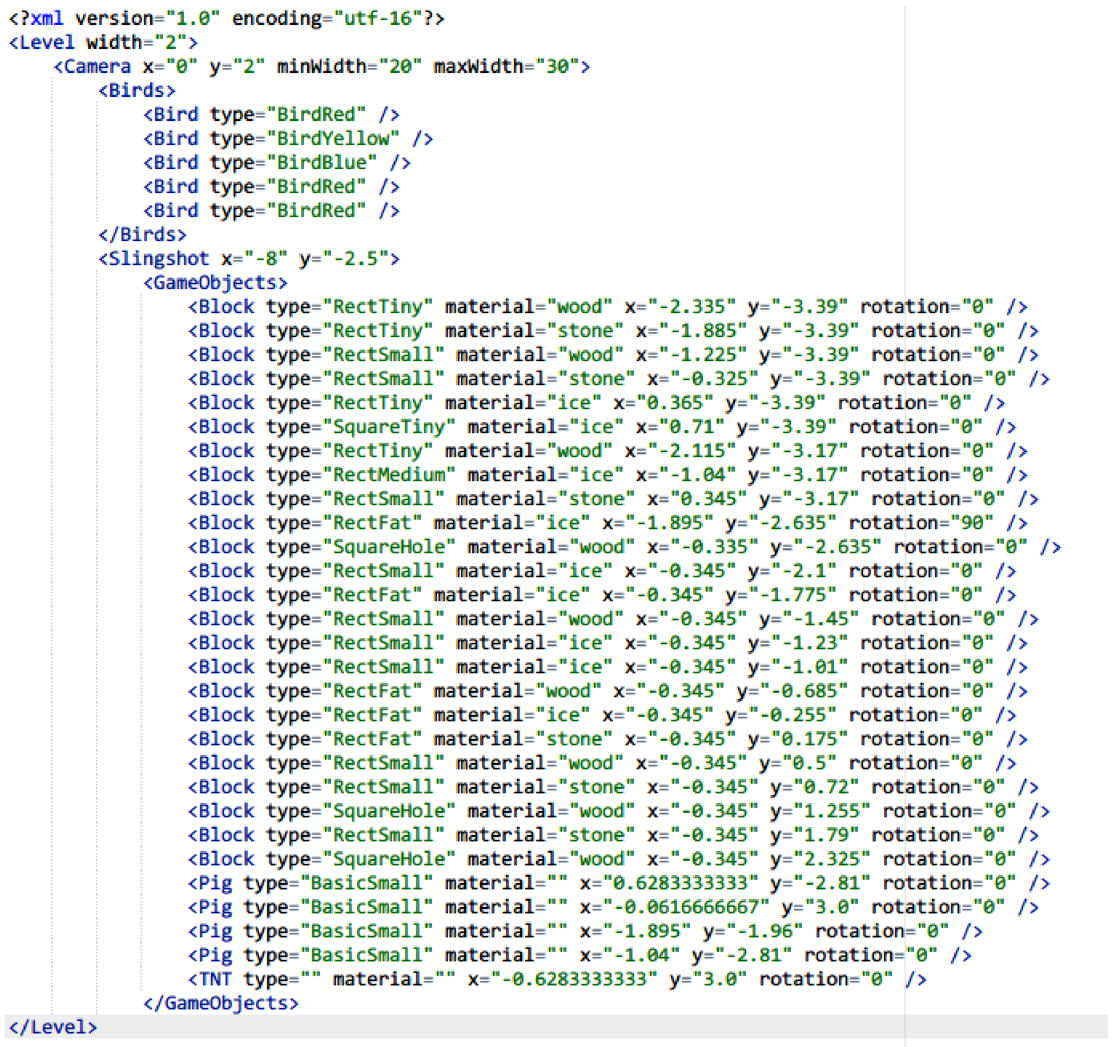


We will use xml file as the input of the game. Each xml file represent a level, and it includes the attributes of the level simulation and the positions of all the game objects (blocks, pigs, TNTs). An example of level xml file is showed in Figure 2.3. In this example, the level has 5 birds, 4 pigs and 1 TNT object. The other elements in the xml describe the type, position, and rotation of each object in the level.

2.2 The Angry Birds AI Competition

The goal of Angry Birds AI Competition is to build computer programs that can automatically create fun and challenging game levels. The difficulty of this competition compared to similar competitions is that the generated levels must be stable under gravity, robust in the sense that a single action should not destroy large parts of the generated

Figure (2.3) Example of a level xml file



```
<?xml version="1.0" encoding="utf-16"?>
<Level width="2">
    <Camera x="0" y="2" minWidth="20" maxWidth="30">
        <Birds>
            <Bird type="BirdRed" />
            <Bird type="BirdYellow" />
            <Bird type="BirdBlue" />
            <Bird type="BirdRed" />
            <Bird type="BirdRed" />
        </Birds>
        <Slingshot x="-8" y="-2.5">
            <GameObjects>
                <Block type="RectTiny" material="wood" x="-2.335" y="-3.39" rotation="0" />
                <Block type="RectTiny" material="stone" x="-1.885" y="-3.39" rotation="0" />
                <Block type="RectSmall" material="wood" x="-1.225" y="-3.39" rotation="0" />
                <Block type="RectSmall" material="stone" x="-0.325" y="-3.39" rotation="0" />
                <Block type="RectTiny" material="ice" x="0.365" y="-3.39" rotation="0" />
                <Block type="SquareTiny" material="ice" x="0.71" y="-3.39" rotation="0" />
                <Block type="RectTiny" material="wood" x="-2.115" y="-3.17" rotation="0" />
                <Block type="RectMedium" material="ice" x="-1.04" y="-3.17" rotation="0" />
                <Block type="RectSmall" material="stone" x="0.345" y="-3.17" rotation="0" />
                <Block type="RectFat" material="ice" x="-1.895" y="-2.635" rotation="90" />
                <Block type="SquareHole" material="wood" x="-0.335" y="-2.635" rotation="0" />
                <Block type="RectSmall" material="ice" x="-0.345" y="-2.1" rotation="0" />
                <Block type="RectFat" material="ice" x="-0.345" y="-1.775" rotation="0" />
                <Block type="RectSmall" material="wood" x="-0.345" y="-1.45" rotation="0" />
                <Block type="RectSmall" material="ice" x="-0.345" y="-1.23" rotation="0" />
                <Block type="RectSmall" material="ice" x="-0.345" y="-1.01" rotation="0" />
                <Block type="RectFat" material="wood" x="-0.345" y="-0.685" rotation="0" />
                <Block type="RectFat" material="ice" x="-0.345" y="-0.255" rotation="0" />
                <Block type="RectFat" material="stone" x="-0.345" y="0.175" rotation="0" />
                <Block type="RectSmall" material="wood" x="-0.345" y="0.5" rotation="0" />
                <Block type="RectSmall" material="stone" x="-0.345" y="0.72" rotation="0" />
                <Block type="SquareHole" material="wood" x="-0.345" y="1.255" rotation="0" />
                <Block type="RectSmall" material="stone" x="-0.345" y="1.79" rotation="0" />
                <Block type="SquareHole" material="wood" x="-0.345" y="2.325" rotation="0" />
                <Pig type="BasicSmall" material="" x="0.6283333333" y="-2.81" rotation="0" />
                <Pig type="BasicSmall" material="" x="-0.0616666667" y="3.0" rotation="0" />
                <Pig type="BasicSmall" material="" x="-1.895" y="-1.96" rotation="0" />
                <Pig type="BasicSmall" material="" x="-1.04" y="-2.81" rotation="0" />
                <TNT type="" material="" x="-0.6283333333" y="3.0" rotation="0" />
            </GameObjects>
</Level>
```

structure, and most importantly, the levels should be fun to play and challenging, (that is, difficult but solvable).

2.3 Previous studies

Previous studies have also explored the use of PCG for Angry Birds. Ferreira et al. [3]. used a genetic algorithm (GA) [4] to optimize the stability of levels. They also proposed a method that can generate levels containing columns of some predefined structures or single objects by using specific defined stacks. This method uses probability tables updated by an estimation of distribution algorithm to locate objects in a given level. In their most recent work , a machine learning technique was used to estimate the stability and feasibility of generated levels which could reduce the high computational time by using simulation.

Kaidan et al. [5], proposed a level generation method that improves Ferreira's work by adapting the difficulty of levels to the player's skills measured according to his or her gameplay results. They later proposed another GA-based algorithm that uses the extended rectangle algebra to quantitatively evaluate the levels' difficulty, according to the structural relation of blocks and pigs in generated levels. They introduced this evaluation into the fitness function of GA.

Jiang et al. [6], proposed a pattern-struct approach to generate different appearance for one of the alphabetical letters each time. They also proposed a preset-model approach to increase the legibility and diversity, where models are built in advance for some capital letters and numbers.

Schiffer et al. [7], used computer vision to infer the scene of a game's instance. Based on this, they implemented a hierarchical search schema by using simulation as a guiding heuristic to determine the parameters of the actions it executes, where simulation is to rate subspaces that will be further explored in more detail on the next levels.

Calimeri et al. [8], presented Angry-HEX, an artificial player for Angry-Birds. The Angry-HEX drives both decisions about which target to hit for each shot (tactic gameplay), and which level should be played and how (strategic gameplay). This procedure is based on declarative knowledge bases, and it is executed by using answer set programming [9] and HEX program [10] which are a proper extension of ASP programs toward integration of external computation and knowledge sources.

Dasgupta et al. [11], proposed s-birds Avengers, an Angry Birds agent. They chose target block to hit in different strategies by estimating coefficients of parameters. In addition. They implemented heuristics to check reachability and prevent retarget previous block.

Wałęga et al. [12], presented a program to autonomously play Angry Birds. Based on qualitative spatial reasoning[13], they introduced qualitative space representation that could use normal words to express the relation between objects. In addition, and a reasoning method, which is used to infer the behavior of these objects after shot.

Stephenson and Renz [14] proposed an algorithm that creates complex stable structures using a variety of 2D objects without predefined substructures or composite elements. Resulting structures are evaluated based on a fitness function that considers several important structural aspects such as the number of pigs and blocks. In their other work[15], they presented a procedural generation algorithm that generates levels consisting of various self-contained structures placed throughout the 2D area, and they also considered the number of birds and the placement of pigs in addition to the number of pigs and blocks

2.4 Goal

Although some generators are using predefined structures to construct structures. All of them have strict constraint on structures which means it has few variation. The main goal of our study is to generate various fun and stable structures by predefining a silhouette.

Methodology

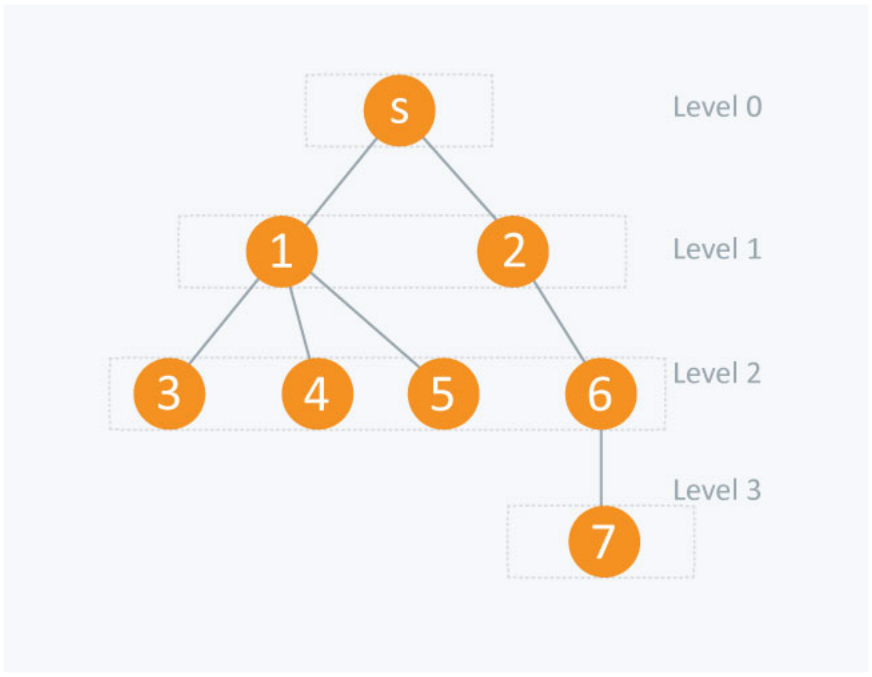
3.1 Method

3.1.1 Breadth first search

Breadth first search was invented by Edward F. Moore [16] in 1959. It is the most commonly used algorithm for traversing or searching tree or graph data structures.

First, it starts with a parent node, move horizontally and visit all the nodes of the current layer. Then it moves to next layer, does same thing in the first step, until all the nodes in that graph are visited.

FIGURE (3.1) Breadth first search



3.1.2 K-means clustering

K-means clustering is a method of vector quantization. It is the most important flat clustering algorithm. K-means clustering aims to partition the n observations into k ($\leq n$) sets S = $S_1, S_2, ..., S_k$ so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance). Formally, the objective is to find:

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 = \arg\min_{s} \sum_{i=1}^{k} |S_i| Var S_i \tag{3.1}$$

where μ_i is the mean of points in S_i

The first of K-means is to select initial cluster center called seed. Then repeat two steps: reassigning data to the closest centroid; and recomputing each centroid based on the current members of its cluster. It will stop when a stopping criterion is met. There are four termination conditions.

- A pre-specific number of iterations has been completed.
- Assignment of data to clusters does not change between iterations.
- Centroids $\vec{\mu}_k$ do not change between iterations.
- Terminate when WCSS Equation 3.1 falls below a threshold.

3.2 Structures generation

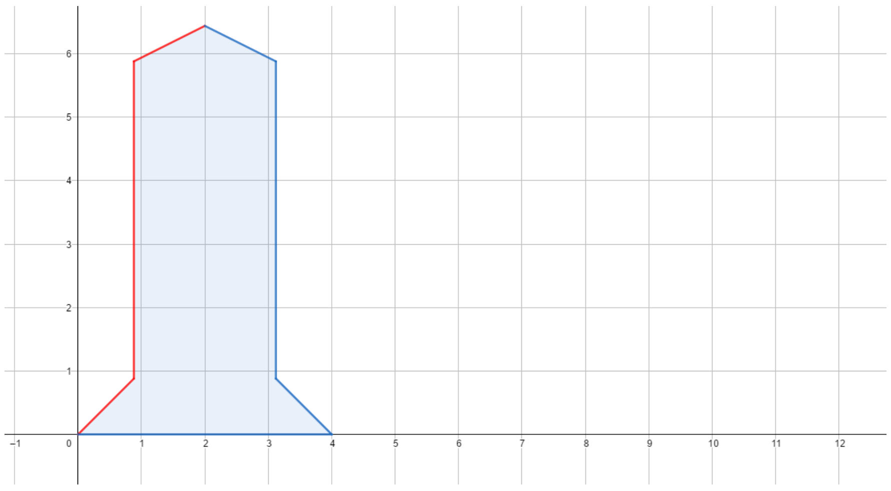
3.2.1 Define silhouette

In order to generate different silhouettes, we need to constrain the height and width limit so the structure could be generated within the silhouette. Here we simply create a file to define the piecewise function for height and width.

For convenience, we only construct symmetric silhouettes. Consider Figure 3.2, we only need to define the left boundary which is the red line. the limit should be a piecewise function, similar to Equation 3.2.

$$x = \begin{cases} y & y \leq 0.88, y \geq 0 \\ 0.88 & y \leq 5.88 \\ 2 \times y - 10.88 & y \leq 6.44 \end{cases} \tag{3.2}$$

Figure (3.2) Example of a silhouette



3.2.2 Generating Row Candidates

We will generate row candidates by using breadth first search. The seed are the medoids of K-means clustering's result, where the input data is previous row candidates, the initial seed is the root node. The row will be divided into several sections with a predefined width, we called it section size.

We will generate row by generating blocks from left to right. In each section, we will generate all possible blocks, then filter blocks based on constraint. In the next section, we will use the filtering result as parents to generate new blocks.

In each block's generation, we will generate all possible blocks that have same height as the max_height where max_height is the height that every block in that row should have and max_height is assigned when the first block is added to that row. It will generates all blocks when add the first block in that row since it have no height limit yet.

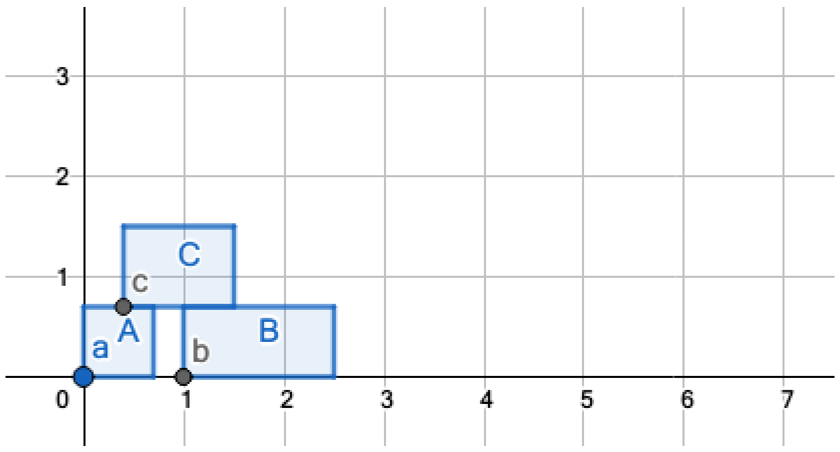
3.2.3 Compatible check

3.2.3.1 Stability check

The stability of the structure is the essential part of the generation. After generating all possibles blocks. We will check the stability of these blocks. Here we only consider the blocks that contact the possible block directly. So when we build the structure from bottom to top, we don't need to consider all the blocks that is under the block we want to add. It makes the generation more efficient.

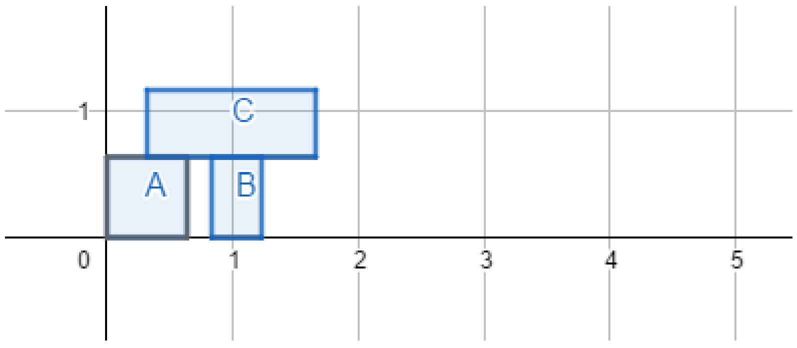
In checking process, we need consider two situations, the first one is several blocks supporting the top block, showed in Figure 3.3. Suppose block A's left boundary point is a(X_a, Y_a) and width is W_a, B's left boundary point is b(X_b, Y_b) and width is W_b, and C's left boundary point is c(X_c, Y_c) and width is W_c, so the structure will be stable only if $X_a + W_a > X_c$ and $X_b < X_c + W_c$.

FIGURE (3.3) Example of stable blocks 1



But this constraint still can not ensure the stability as the structure may be the shape showed in Figure 3.4. So, we need to ensure the right most block could provide firm hold on upper block. In this study, we didn't consider some extreme situations. Therefore, we consider the structure will be stable if it meet the constraint which is $X_b + \frac{W_b}{2} > X_c + \frac{3 \times W_c}{4}$.

FIGURE (3.4) Example of stable blocks 2



Another situation is only one block supports upper block. In this situation, two blocks' center should have same x-axis value showed in Figure 3.5 if the bottom block's length is smaller than the top block's length. Or the bottom block's left boundary is smaller than top block's and right boundary is larger than top block's boundary, showed in Figure 3.6

FIGURE (3.5) Example of stable blocks 3

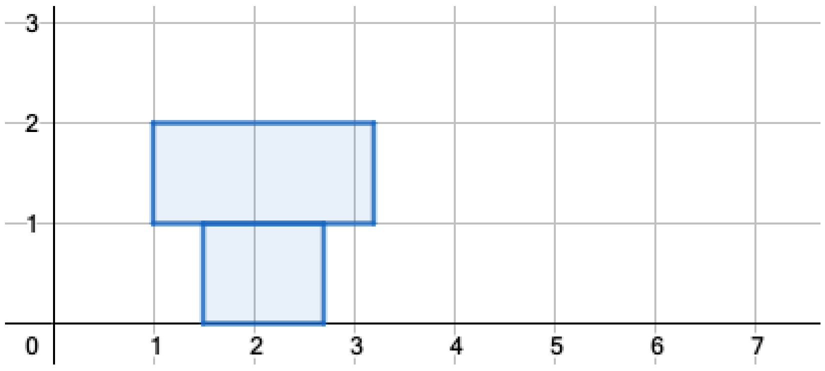
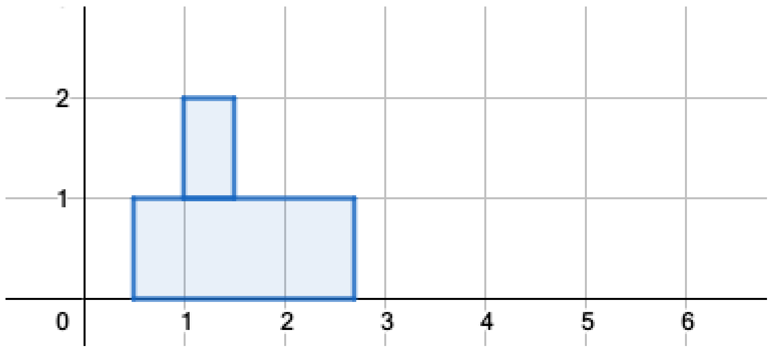


FIGURE (3.6) Example of stable blocks 4



3.2.3.2 Limitation and Overlap check

The limits ensure all blocks are added within the silhouette. We need to check if the structure exceeds boundary when adding blocks, or the height of the structure is larger than silhouette's height limit when adding that possible block. In addition, we need to ensure there is no overlap between the possible block and previous blocks.

3.2.4 Candidates pruning

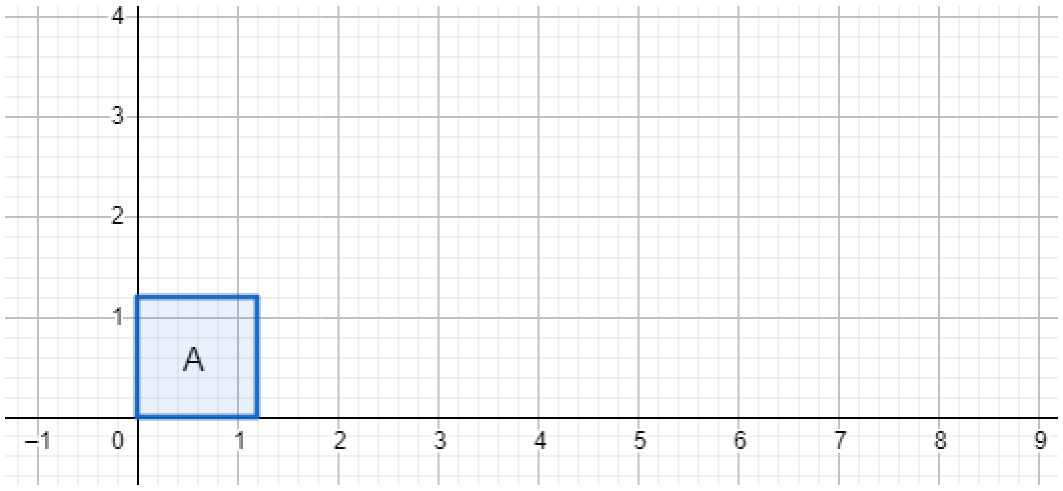
After generating the row, I find there exists lots of similarity between rows. And the number of parents for generating next section increase exponentially. So it is better to filter candidates to select rows that could represent as much features in candidates as possible.

3.2.4.1 Vectorization

The best way to filter candidates is remove similar candidates. So we need to find difference between candidates so we can filter it based on difference.

The best way to find difference is to use mathematic functions to calculate similarity. Similar to vector space model[17], We will vectorize rows. We convert each section into a vector, each vector represents the space that blocks occupied in that section.

FIGURE (3.7) Example of section vectorization



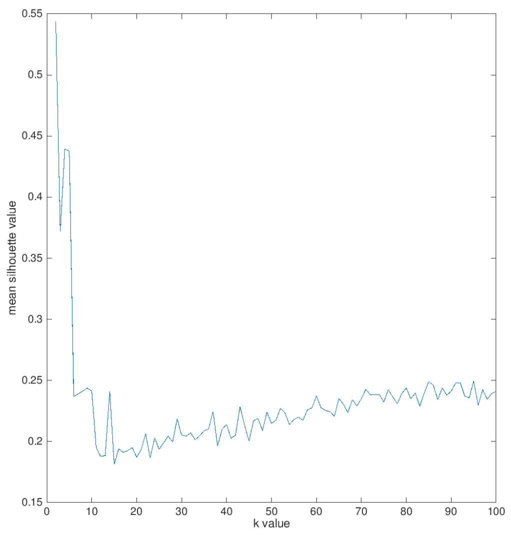
If the block's size in Figure 3.7 is 1.2×1.2, and suppose each section's length is 1 and using the Coordinate origin as start point, we can vectorize first section into a vector which should be $\begin{bmatrix}1.0\\1.2\end{bmatrix}$ and the second section should be $\begin{bmatrix}0.2\\1.2\end{bmatrix}$, the first element in vector is the width that block occupied in this section, and the second element is the height that block occupied. By using this method, we vectorize all sections in one row. The result should be a $2 \times N$ vector where N is the number of sections. So the result is a $x \times N$ vector where the element in that vector is a vector for a section. For convenience, we flatten the $2 \times N$ vector, convert it into $1 \times 2N$ vector and store it into a csv file.

3.2.4.2 Clustering

Using the data stored in csv file, we implement k-means clustering to cluster all candidates. In K-means clustering, it is important to find a K value that could put data vectors into appropriate clusters. So the centroid could represent the most features in that cluster. The first step is to choose an appropriate K. In order to find the best K, we implement MATLAB functions to test result in different K value.

The silhouette value here indicates the distance between each point in one cluster and points in the neighboring clusters. The value ranges from $+1$, indicating points that are very distant from neighboring clusters, through 0, indicating points that are not distinctly in one cluster or another, to -1, indicating points that are probably assigned to the wrong cluster. As we can see, the clustering's result decreases as K value increases. the best mean is 0.59 when $K = 2$. So, we select 2 as K value in this candidates set. But

FIGURE (3.8) Mean silhouette value in different K value



note that the K value might varies in different candidates set since it will have different number of sections in different height.

Instead of choosing centroid as the nodes that we selected to be the parents of next row. We will choose medoid to be the parents of the next row as centroid may not be the vector in the date set, which means it may not be able to represent certain structure that the available blocks could construct.

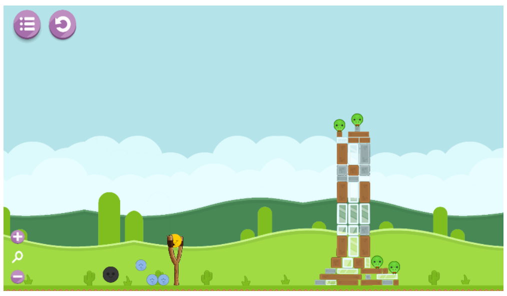
3.2.5 Generate overall structure

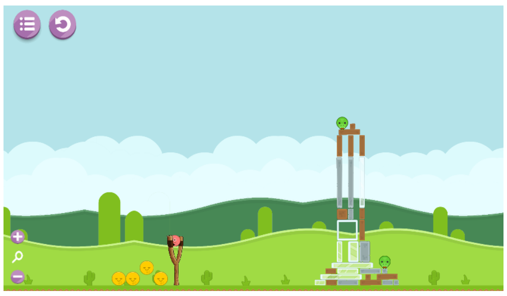
We will use a Node object to store a block's information, e.g. block type, block bottom-left corner's x-axis point and y-axis point. And using a pointer to connect every node. By using the approach stated above, the final structure is a single linked list, where every element in the list is a Node object.

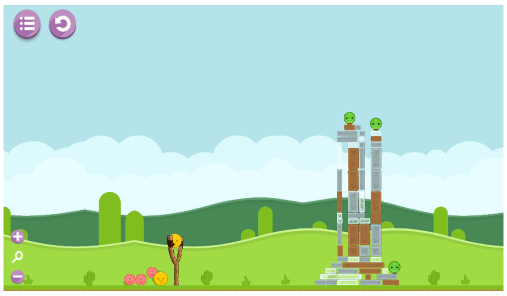
Experiment results

4.1 Variation in one silhouette

In order to test the variation of structures, we perform an empirical evaluation. We use the silhouette of Figure 3.2 mentioned before to construct structures with a section size 0.45. And here is the result of generation.


(A) Structure 1


(B) Structure 2


(C) Structure 3

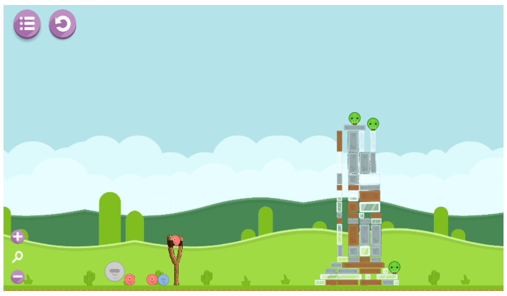
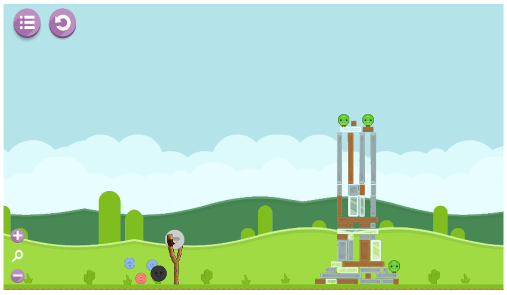
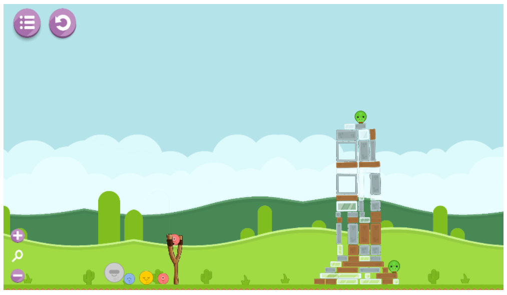

(D) Structure 4

FIGURE (4.1) Result of generation 1.

From this result, we could find that these structures are very distinct from each other. Some structures occupy most of the silhouette' space, like the structure 1. But some structures only occupy small amount of space, like structure 7, it looks like a twin tower.


(E) Structure 5


(F) Structure 6

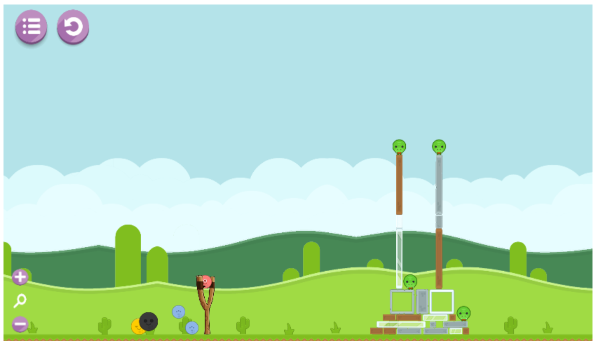
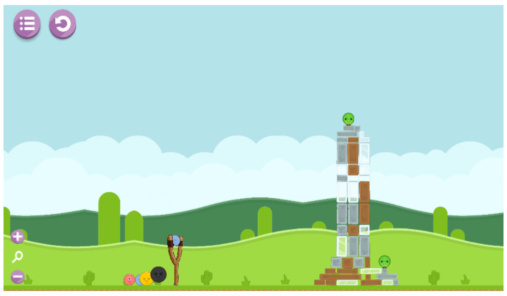
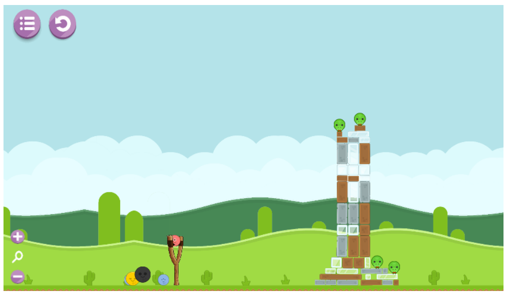

(G) Structure 7

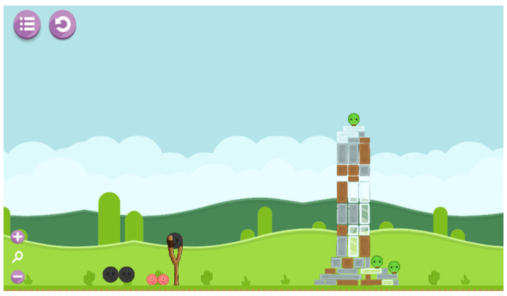
FIGURE (4.1) Result of generation 1.

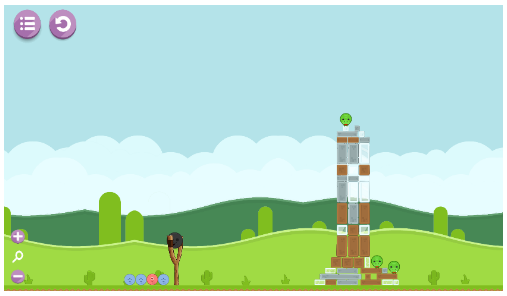
We notice that there are some identical rows among structures. For instance, all the structures' first rows have same height. The reason is, when moving pointer towards end, it will always choose the shortest blocks that satisfy the height constraint since the upper boundary is inclined instead of vertical. And once the first block of that row is chosen, the row's height is established which is the first block's height. Every block in that row have same height. It causes the bottom rows are always have same height with the smallest block's height. Furthermore, in Figure 2.2, we can find that it has few choices which is block 8, 10, 12. So the variation in the first row will be small, and it will be fewer after clustering as it select the vector that represents most features in that cluster.

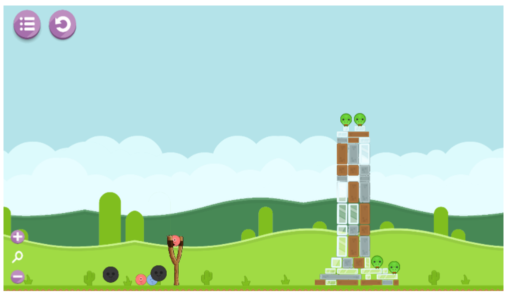
In addition. We find that some structures showed below are almost same, except some trivial difference. The reason why it has some almost identical structures in result is some of the clusters' medoids have same parents. So it results to a bunch of structures that have same bottom but only have difference on top of structures. We also found some rows are consists of same blocks. For instance, the fifth level consists of same "RectFat" blocks. The first block's height is 0.84, as we can see from Figure 2.2, it has few choice at this height. So it ends up with a row with same blocks, and due to the stability constraint, it can not extend the length that the bottom row has.


(A) Structure 1


(B) Structure 2


(C) Structure 3


(D) Structure 4


(E) Structure 5

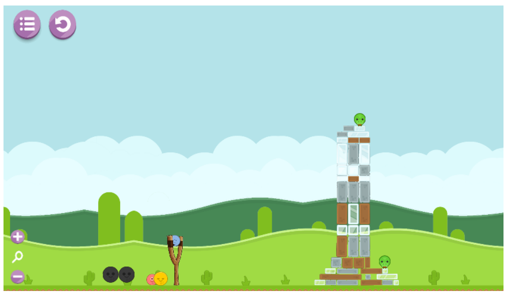
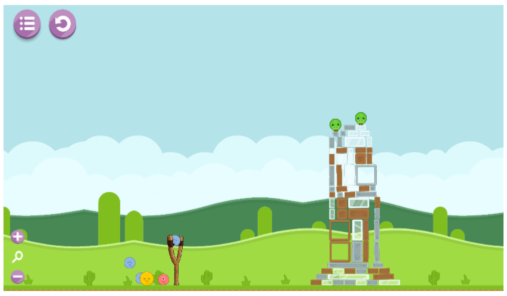

(F) Structure 6

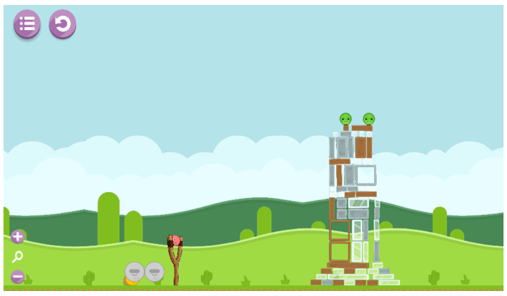
FIGURE (4.2) Result of generation 1.

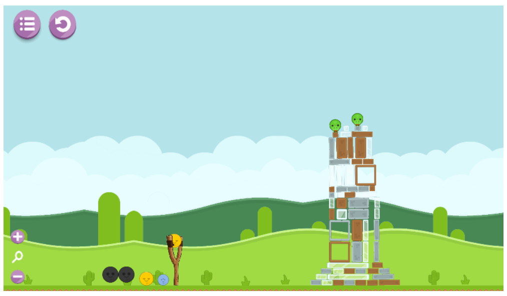
4.2 Variation in same silhouette with different section size

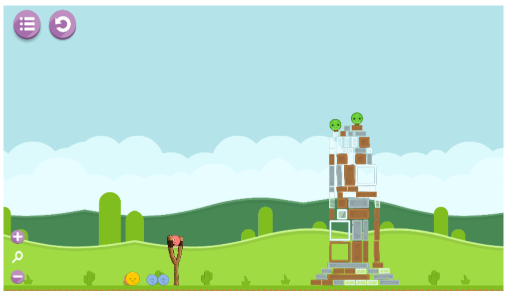
The section size could also cause variance even using same silhouette. We apply the same silhouette used before and choose a section size 0.15 to test. The results are showed below.

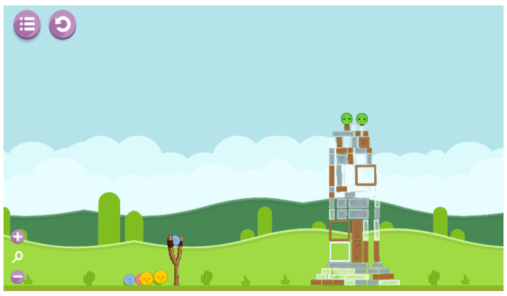
In this result, we find these structures are totally different to the structure generated before. And each structure is different from other structures in result. The reason that it generates more diverse structures is it could have more position to place blocks when the section size decreases. As a result, it leads to a more diverse structures in one generation and more different structures from previous generation when section size is 0.45.

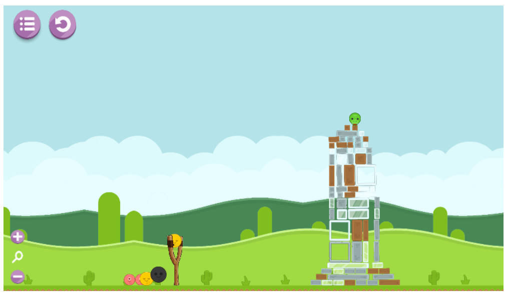

(A) Structure 1

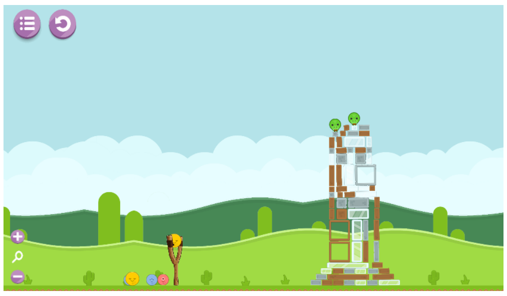

(B) Structure 2

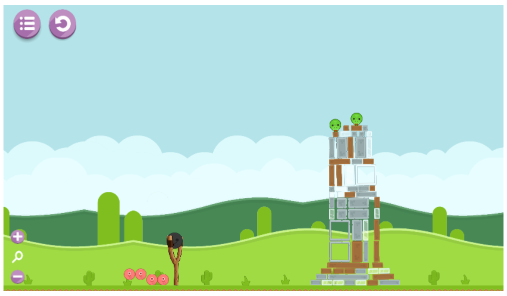

(C) Structure 3


(D) Structure 4


(E) Structure 5


(F) Structure 6


(G) Structure 7


(H) Structure 8

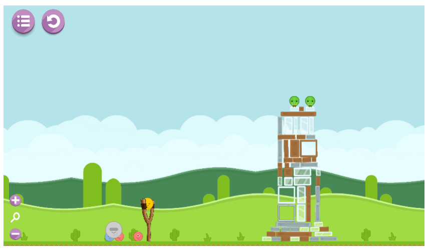

(I) Structure 9

FIGURE (4.3) Result of generation 2.

4.3 Variation in different silhouette

To test variation in different silhouette, we choose different silhouette Figure 4.4 to test. The results are showed below.

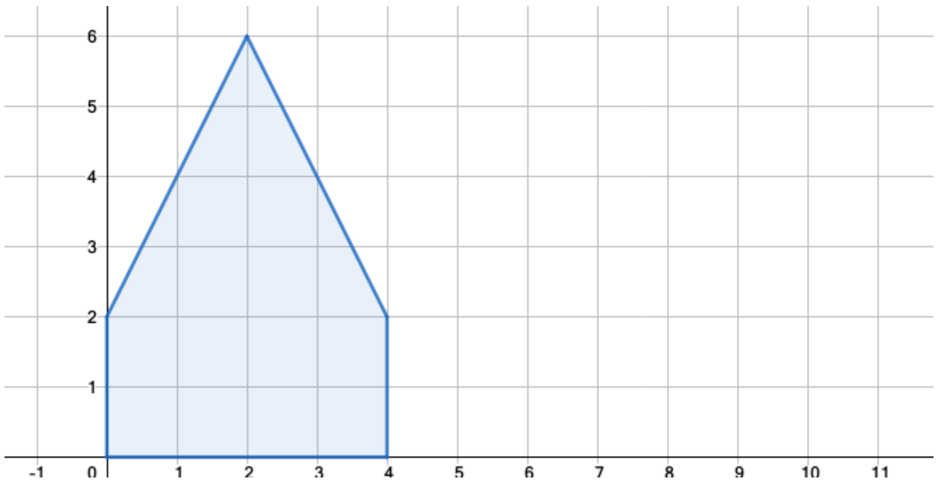
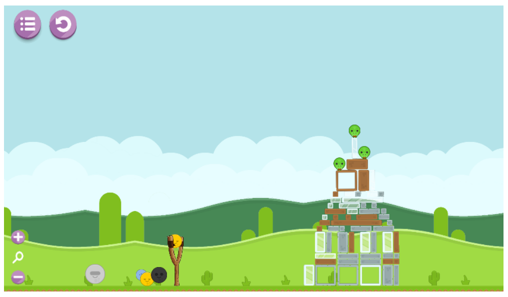
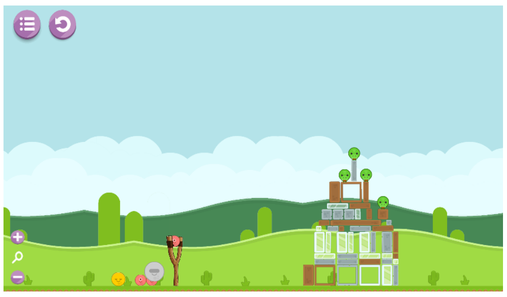


FIGURE (4.4) silhouette 2


(A) Structure 1


(B) Structure 2

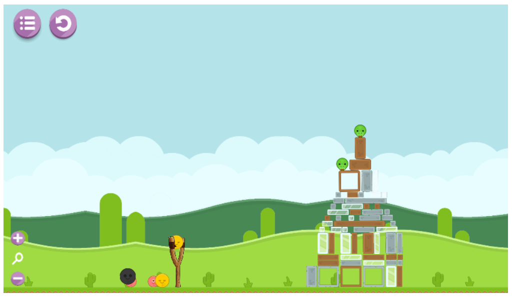
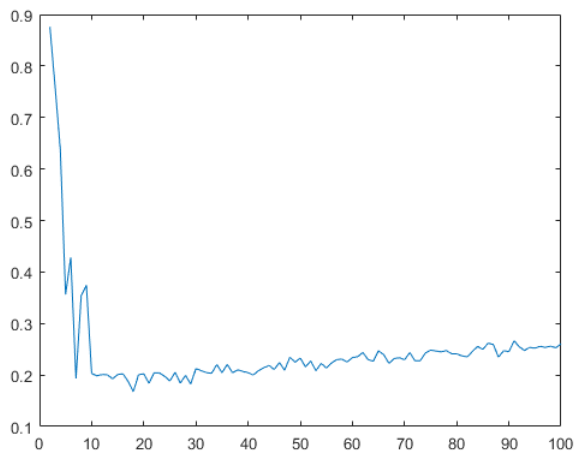

(C) Structure 3

FIGURE (4.5) Result of generation 3.

We notice that it only generates three structures. Since we select K value based on the silhouette mean, and in the Figure 4.6 below, it has the highest value when k=2. Therefore, it leads to smaller amount of variation. By checking the original data, we find a lot of variation which means the criteria for selecting K is still not ideal. As mentioned before, those blocks with less height will be the first one to fit in the silhouette when the upper boundary is inclined.

FIGURE (4.6) Mean silhouette value in different K value

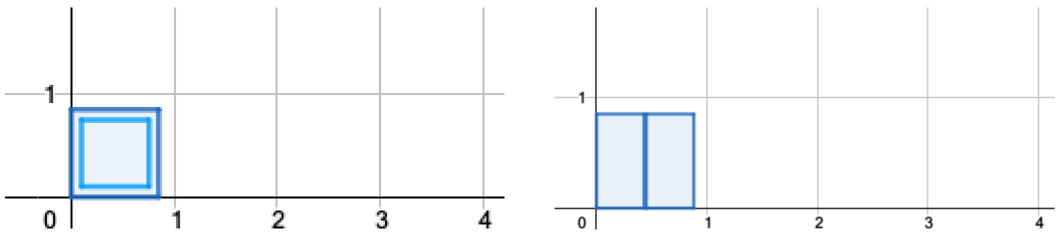


4.4 Result analyze

In these results, we found many differences. But we also found some similarities. Part of the reason lies in the row quantification. The vectorization that we used still can not quantify rows ideally. Suppose we use 0.45 as section size. And here are two candidates, showed in Figure 4.7.

FIGURE (4.7) Example of row quantification

(A) Row 1 (B) Row 2



The first candidate has one block, its size is 0.85×0.85. The second candidate has two blocks, both of the blocks have same size, which is 0.43×0.85. So the first candidate's vector should be $\begin{bmatrix} \begin{bmatrix} 0.45 \\ 0.85 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.85 \end{bmatrix} \end{bmatrix}$, and the second candidate's vector should be $\begin{bmatrix}\begin{bmatrix}0.43\\0.85\end{bmatrix} & \begin{bmatrix}0.43\\0.85\end{bmatrix}\end{bmatrix}$. By using cosine similarity in Equation 4.1.

$$similarity = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2 \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}} \tag{4.1}$$

where A_i and B_i are components of vector A and B respectively.

The similarity is 0.99 which indicates these two vectors are pretty similar. But in reality, the row 1 is a "SquareHole" block, but the second row is two "RectFat" blocks. These two rows are pretty different.

Conclusion and future work

This thesis presents the author's approach to generate various structures in Angry Birds, by using different silhouettes. The experiments confirmed that the structures it generated are stable and have variance.

However, we've noticed some silhouette only generates small amount of structures due to stability limit. The future work will include will finding a better criteria to allow generator to generates more diverse structures. In addition, we've noticed that it took about 4 hours to generate all structures with a normal size silhouette and a large section size, and took almost 11 hours to generate all structures with a smaller section size, which is not efficient. In the future work, we will implement GPU acceleration in MATLAB to accelerate clustering and in Python generator to increase computation efficiency. Besides, as we can see in the section 4, the criteria for selecting K is still not ideal, so we still need to analyze the result and find a way to get K value with the best variance. Furthermore, the row quantification is not ideal. Future work will also include finding a better way to quantify the row structure.

Bibliography

[1] N. Shaker, J. Togelius, and M.J. Nelson. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer International Publishing, 2016. ISBN 9783319427164. URL https://books.google.ca/books?id=-IdJDQAAQBAJ.

[2] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[3] Lucas Ferreira and Claudio Toledo. A search-based approach for generating angry birds levels. In *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games*, CIG'14, 2014. doi: 10.1109/CIG.2014.6932912. URL http://www.lucasnferreira.com/papers/2014/cig-evoab.pdf.

[4] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[5] Misaki Kaidan, Chun Yin Chu, Tomohiro Harada, and Ruck Thawonmas. Procedural generation of angry birds levels that adapt to the player's skills using genetic algorithm. In *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*, pages 535–536. IEEE, 2015.

[6] Yuxuan Jiang, Tomohiro Harada, and Ruck Thawonmas. Procedural generation of angry birds fun levels using pattern-struct and preset-model. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, pages 154–161. IEEE, 2017.

[7] Stefan Schiffer, Maxim Jourenko, and Gerhard Lakemeyer. Akbaba—an agent for the angry birds ai challenge based on search and simulation. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):116–127, 2016.

[8] Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton

Wimmer. Angry-hex: an artificial player for angry birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games*, 8 (2):128–139, 2016.

[9] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[10] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *IJCAI*, volume 5, pages 90–96. Citeseer, 2005.

[11] Sourish Dasgupta, Savan Vaghela, Vishwa Modi, and Hitarth Kanakia. s-birds avengers: A dynamic heuristic engine-based agent for the angry birds problem. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):140–151, 2016.

[12] Przemysław A Wałęga, Michał Zawidzki, and Tomasz Lechowski. Qualitative physics in angry birds. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):152–165, 2016.

[13] Anthony G Cohn and Jochen Renz. Qualitative spatial representation and reasoning. *Foundations of Artificial Intelligence*, 3:551–596, 2008.

[14] Matthew Stephenson and Jochen Renz. Procedural generation of complex stable structures for angry birds levels. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.

[15] Matthew Stephenson and Jochen Renz. Procedural generation of levels for angry birds style physics games. In *The AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2016.

[16] E.F. Moore. *The Shortest Path Through a Maze.* Bell Telephone System. Technical publications. monograph. Bell Telephone System., 1959. URL https://books.google.ca/books?id=IVZBHAAACAAJ.

[17] Ray R Larson. Introduction to information retrieval, 2010.