**An Auto-levels-generator in Angry Birds using Real World Objects**

**Abstraction:**

we present an level generator which automatically convert the real world objects to the structure in the Angry Birds world. First, the we use one of TensorFlow model called “DeepLab” which re-purposes networks trained on image classification. This model is able to extract the real world objects’ segmentation and turn it to the matrix if the object include in database. It also can be trained to identify the uncatalogued objects. Second, we establish a physic model which is able to construct the structures similar to the segmentation of real world objects in Angry Bird. In physic model, it selects the key rows and columns in the matrices from ‘DeepLab’ to reduce complexity, and transforms it to honeycomb-like matrices in order to increasing stability of structures. Then, It labels each position of the matrix according to the status of each position’s neighbors. Therefore, it can extract the outline of the matrix and convert the outline to a series of stable rooms supported by squares and rectangle. At last, we places pigs and TNT in the rooms. In the experiment, we use COCO database which contains 80 different type of catalogs.

**1 Introduction:**

Imagining are the key component of the game creating especially in level construction. Mankind are very good at project the real world objects or concepts to the virtual world they desire. In 1983,

Gardner presented his theory of multiple intelligences [1]. One of the intelligences is the spatial intelligence which enable people to navigate in the space and visualize objects. This intelligence contains manipulation of information presented in a visual, diagrammatic, or symbolic form as opposed to verbal language-based modality [2].

In recent, the researchers are trying to investigate auto-level-generator in Angry Birds which is an Physics-Bases Simulating Game. They would like this generator can construct various levels that are interesting to play. The Angry Bird has an 2 dimensional physical simulating environment which players need to kill all the pigs in the environment by launching the birds from a slingshot. Those pigs are usually stand up or covered by structures so that player cannot one-shot them. However, this environment are highly control by physics laws. Player allow to make the structure fall down by shooting them and fallen structure can deal damage to pigs.

Our paper purpose an auto-level-generator demo which can visualize the real world objects in the game world. Since the objects in Angry Birds are manipulated by physical laws, our demo has two parts, object extraction model and physics model.

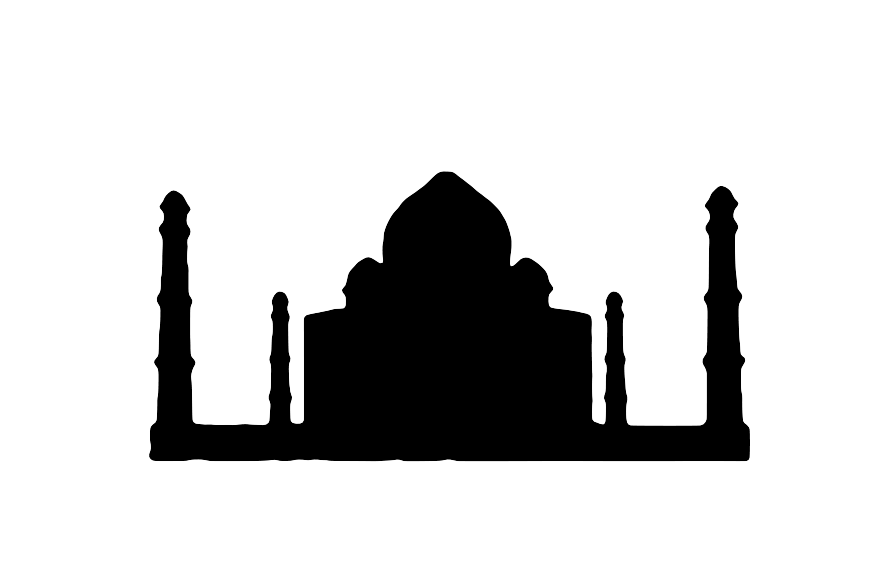
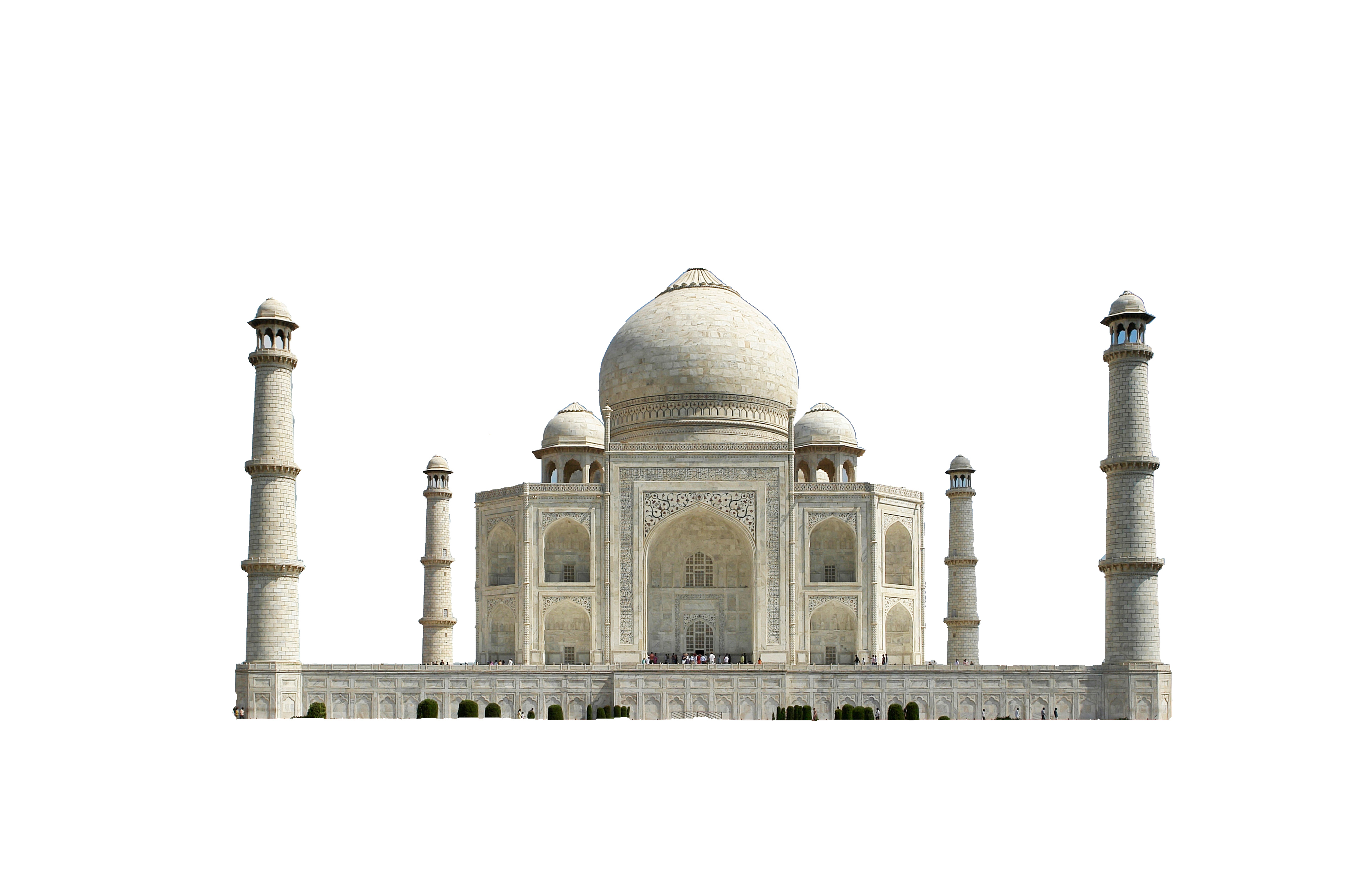
**2. Relative Work:**

**2.1 Object detection**

The object extraction model essential is an artificial intelligent trained by one of TensorFlow model ‘DeepLab’[3]. This artificial intelligent can help us to detect and extract the object from real world images by applying ‘Atrous convolution’ and conditional random fields. It converts the image from JPG format to the matrix contain different labels that indicate the categories of the objects if any.



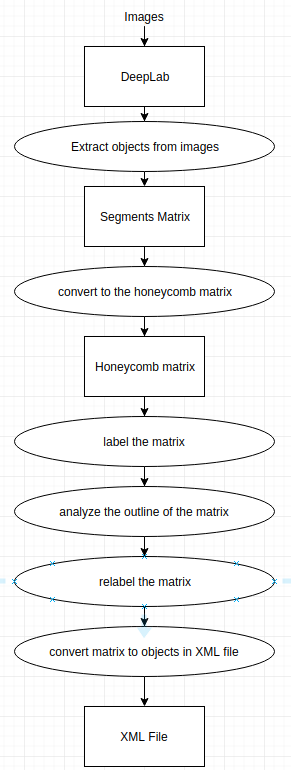
For any unclassified object, we use “GIMP” which is an image manipulated tools to manually cut off the background. Then, we transform the image to SVG polygon format which is an matrix contains the shape of object.

 **Origin SVG-polygon**

**2.2 Physic Model**

The second part is physics model which is going to construct the level automatically according to given matrix.

There are many different shape of structure objects and each object can be made by three different materials, ice, wood, and stone. The center weight of the objects affected by the total area and material density[2]. In our experiment, the width and height of the object in the Angry Birds are not exactly same with the documents. This factor gradually becomes unavoidable when the height of structure becomes higher and higher. We found that even for same structure like simple column with ice blocks. The higher structure still possible fall down when the lower structure structure are stable. Therefore, we make a series process to ensure the maximum stability of the structure.

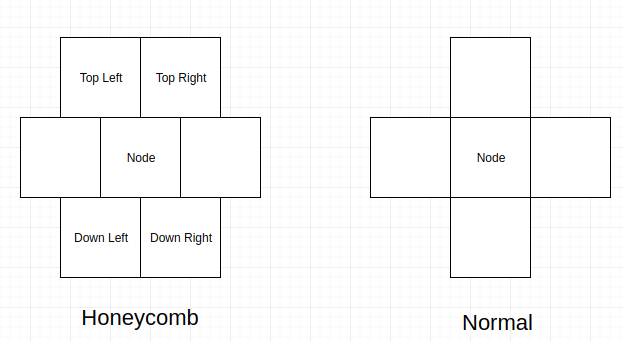
**3. Methodology:**

In methodology section, we detail describe how the program process the matrix generated by the Deeplab. The matrix is transformed into a honeycomb-like matrix first and each block in matrix will be labeled into one of four categories. Then, the program analyzes the shape of the matrix and calculate how many rectangle and square required in each row. After that, the program replace the content in each block to the labels of rectangles, squares, and pigs. At the last, the program contruct the XML file according to the label in the matrix.

**3.1 Honeycomb matrix**

First, the program shifts the matrix to a honeycomb-like matrix.

The advantage of honeycomb-like matrix is that each position has six neighbors instead of four in normal matrix. Therefore, each square can have one more supported blocks.



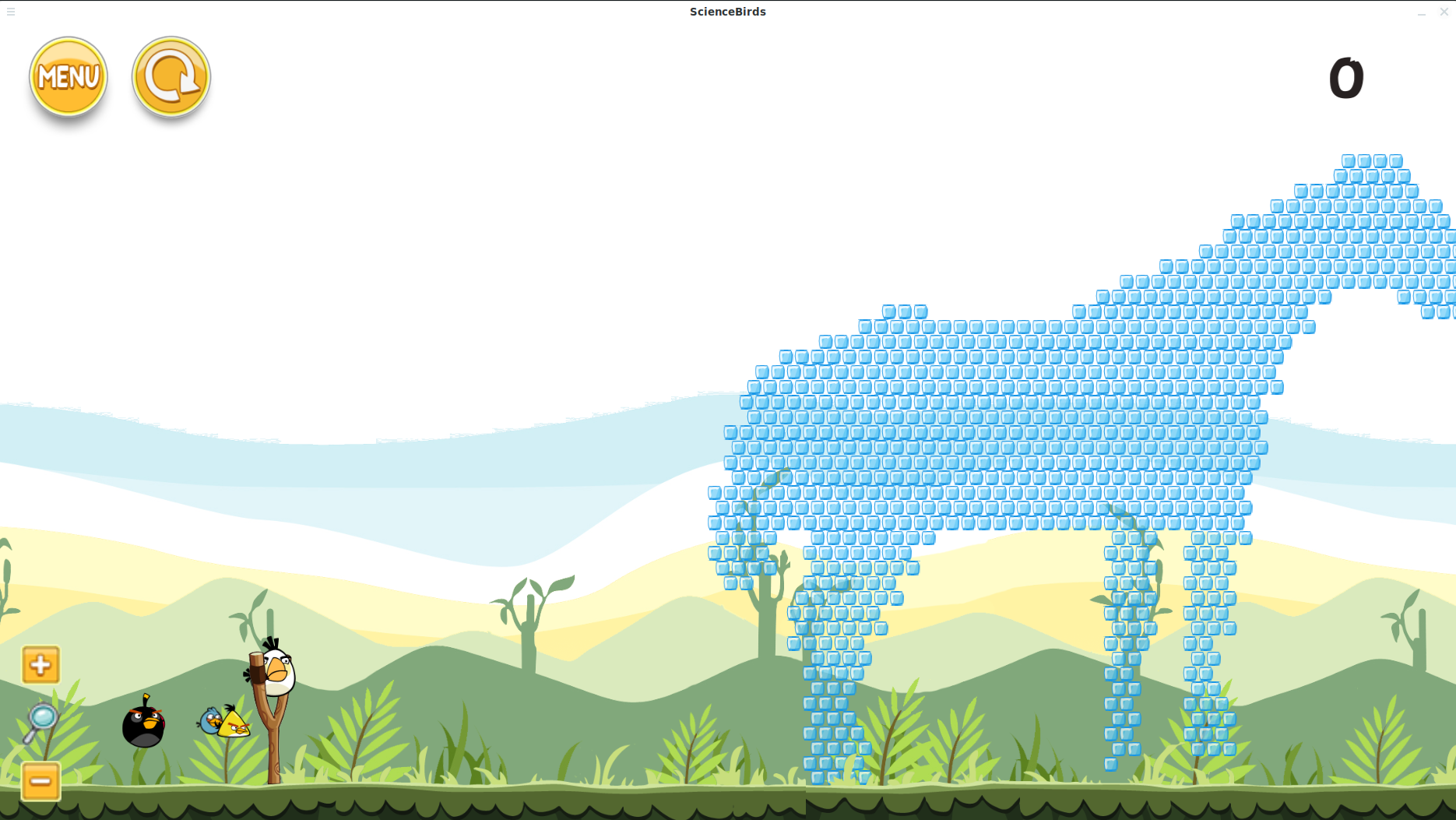
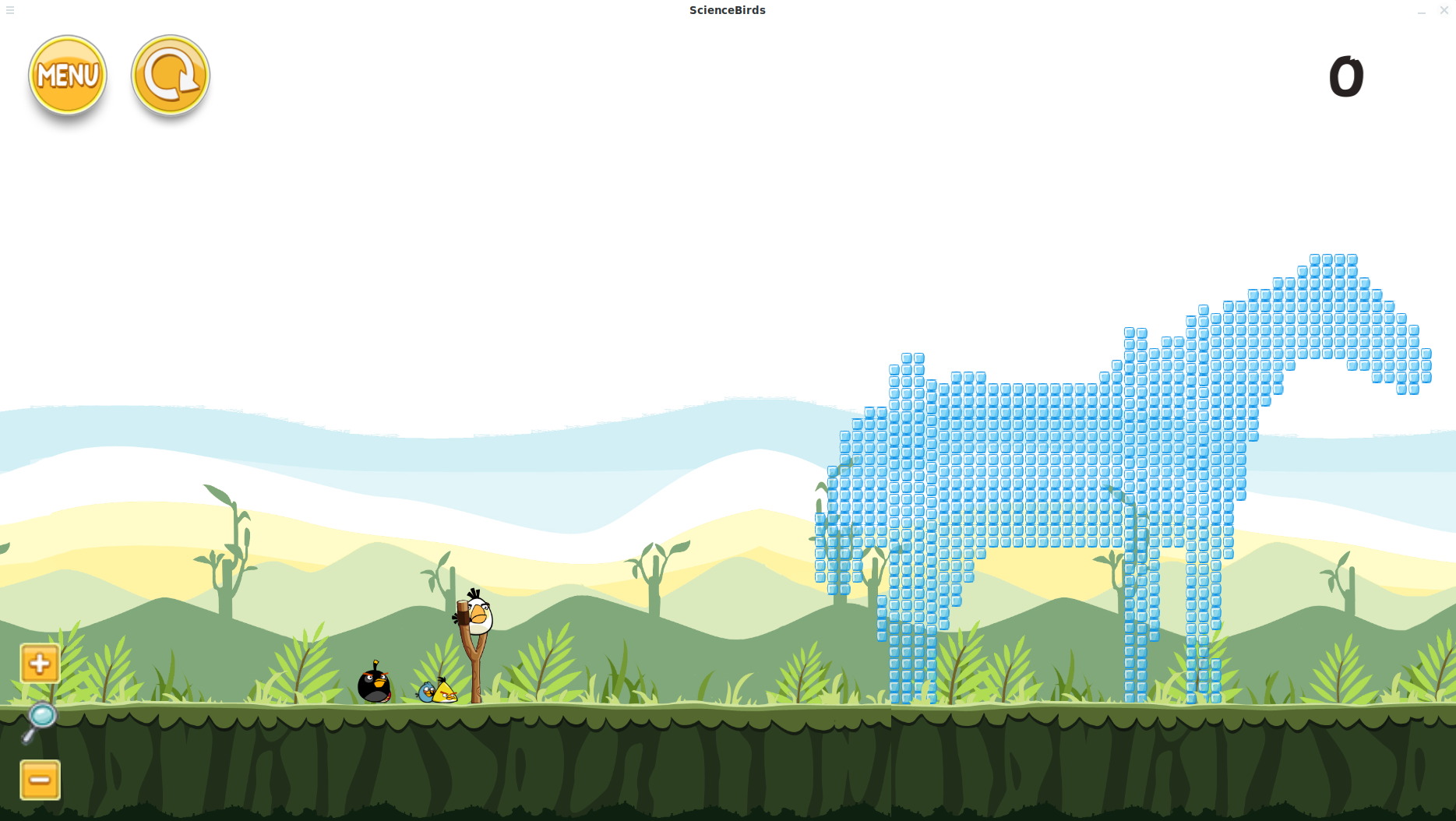
In practice, the program selects some of positions in the matrix in order to reduce the complexity of matrix. We have four different difficulty settings which basically determine how large the structure gonna be. In order to preserve the ratio of height and width from original image. We simply set an interval variable that pick only one row or column between the intervals from the input matrix from DeepLab. However, we found that some image with large size may excess the game boundary. Therefore, we make some restrictions for the interval in Table 1.

|  |  |
| --- | --- |
| Interval\_width | min(game\_boundary\_width, image\_width/ 30 \* difficulty) |
| Interval\_length | min(game\_boundary\_length, image\_length/ 30 \* difficulty) |
| Interval | max(Interval\_width,Interval\_length) |

Then, the program let every second row shift one position to the right. Therefore, each position have two blocks supporting, down-left and down-right, and it supports two blocks, top-left and top-right. The relation between the position and its neighbors is in Table 2.

|  |  |  |
| --- | --- | --- |
|  | Row is even | Row is odd |
| Top-left | Column - 1 | Column |
| Top-right | Column | Column +1 |
| Down-left | Column -1 | Column |
| Down-Right | Column | Column |
| Top-left, Down-left = Column -1 + Row % 2 | | |
| Top-Right, Down-Right = Column -1 + Row % 2 | | |

**Table 2**



**Figure 1**

**Honeycomb Matrix(bottom) verse Normal Four Neighbors Matrix(top)**

As we can see in the above Figure 1, the honeycomb could hold the shape in a short periods when game loading. In the other hand, the normal four neighbor matrix literally fall down vertically when the beginning of game. Although at the late of game stating, the honeycomb matrix still will fall down, the honeycomb will be split into parts rather than directly fall down. Therefore, if we can just adding some support under the key area of the those part, the whole structure can be stabilized.

**3.2 Analyze**

Then, we analysis the outline of the structure from matrix in order to only keep the key shapes. By inspecting neighbors, we can label each position in one of four situations in Table 2.

Table 2

|  |  |  |
| --- | --- | --- |
|  | Down-right exists | Down-right empty |
| Down-left exists | 1 | 2 |
| Down-left empty | 4 | 3 |



Figure 2

Honeycomb Matrix of House’s Shape

In the experiment, we found that the squares are stable if it satisfied those conditions in Table-3.

Table-3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Label = 1 | Label = 2 | Label = 3 | Label = 4 |
| Only Top-left  exist | Stable | Stable | Unstable | Unstable |
| Only Top-right exist | Stable | Unstable | Unstable | Stable |
| Both exist | Stable | Stable | Unstable | Stable |
| Both empty | Stable | Unstable | Unstable | Unstable |

Therefore, as long as the block satisfies one of three conditions. The block is stabilized. These three conditions are:

1. label is 1.

2. label is 2 and top right are stabilized.

3. label is 4 and top left are stabilized.

Figure 3

Simple Stabilized structure – in which numbers in the image are corresponding to the label of the block

**3.3 Relabel**

After analyzing the matrix, we can relabel the matrix according the analysis. First, we only keep the information of each fourth row . The other row will be removed from the matrix. Then, we calculate the width of the object in each row.

**Algorithm 1:**

# input : an labeled integer matrix, “matrix”

# output: an 2-d list contains start positions and end positions of object in each row

widths ← create an empty array

for i ← 0 to length of matrix, i += 1 for each loop:

width ← create an empty array

copy ← make a copy of matrix[i]

for j ← 1 to length of matrix -1, j += 1 for each loop:

bool\_row ← copy[j] != 0

bool\_plus ← copy[j+1] != 0

matrix[i][j] = 0 # remove the label from matrix

if (NOT bool\_row) AND bool\_plus:

left ← j + 1

if (NOT bool\_plus) AND bool\_row:

width ← add [left,j] into list

widths ← add width into list

return widths

Because the object could exist multiple independent parts in same row, we have to use list to store every widths in the row.

Hence, we use a sliding windows to isolate each section in the row, which is the two Boolean values in the algorithm. This algorithm is going to check the status of current and next block to see whether those blocks contains labels. If the window is process to the starting point of one part of the object, the current value should be false and next value is true, vice verse.

**Example:**

**0 0 0 0 2 1 3 3 3 1 1 1 4 0 4 2 0 1 0 (matrix)**

**0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 (current)**

**0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 (next)**

**|-----------------------| |---| | (intervals)**

Then, the program can relabel the matrix. First, the program calculate how many different rectangles object need to fill in to the row according to the its width. Then, it adds support to the both side of the rectangles to increase stability. Therefore, two kinds of support, squares and platform. The square is an 1 \* 1 rectangle which are the basic unit of the matrix. The platform is a unbreakable territory which ignore all physics law in the Angry Birds. In some cases, the objects are fly in the sky or swimming in the water which hardly be mimic by the game. Hence, we use territory to support the object.

**Algorithm 2:**

#input: an integer matrix contains rectangle labels ,“matrix”

a 2-d array contains all point need to be supported

,“rectangle\_points”

#output: an integer matrix which all rectangle are supported

for row in rectangle\_points:

for left, right in the rectangle\_points[row]:

supported = False

for offset ← -2 to 2:

if matrix[row+4][left+offset] != 0:

add\_square \_support() ← from matrix[row][left] to matrix[row+4][right+offset] supported\_left = True

if matrix[row+4][right+offset] != 0:

add\_square \_support() ← from matrix[row][right] to matrix[row+4][right+offset]

supported\_right = True

if Not supported:

matrix[row+2][left] ← label represent platform

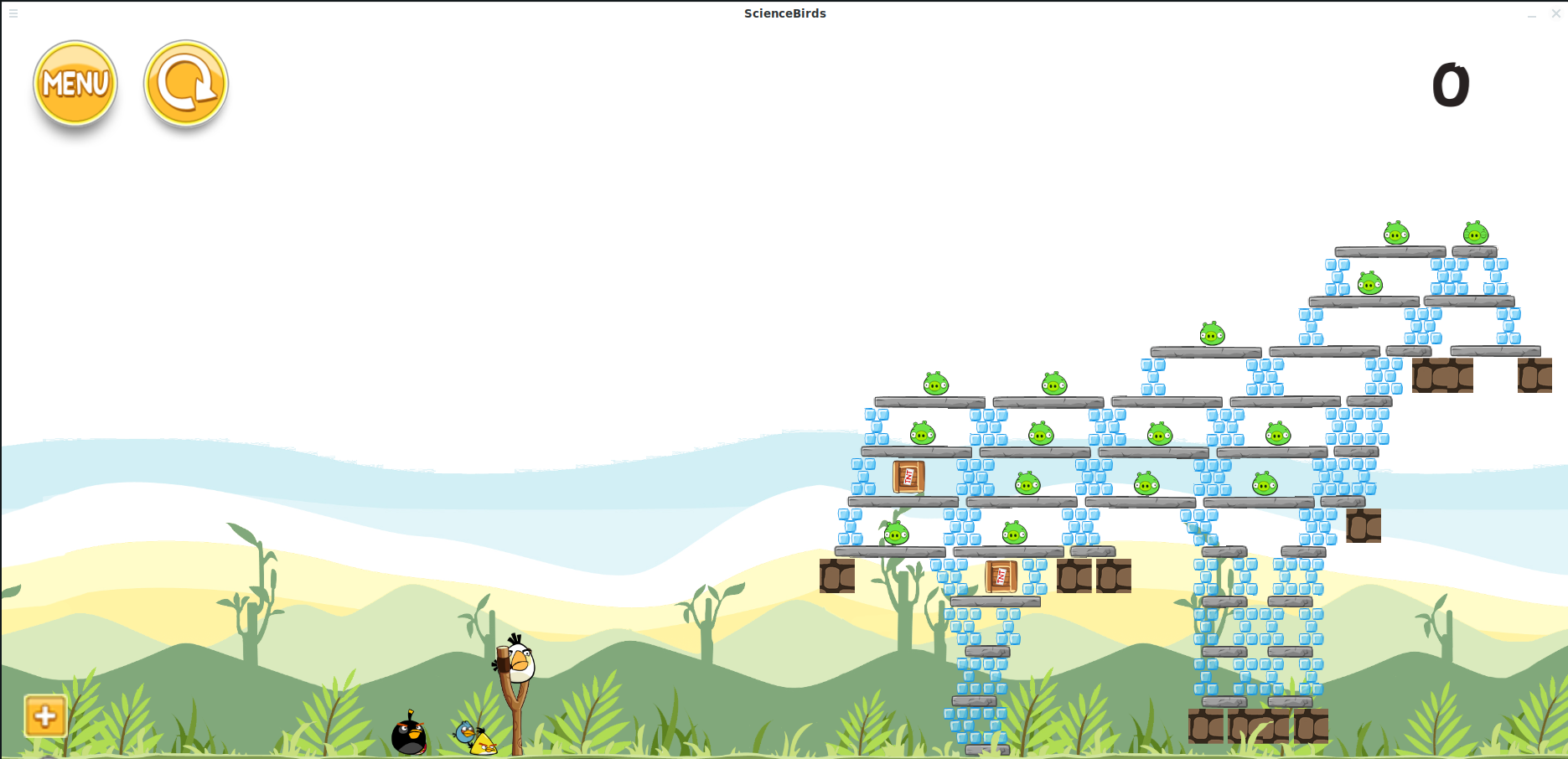
return matrix

After placing support, the structure is divided into rooms. The program detects these room to see whether its empty. Then, program randomly places the pigs and TNT into empty room.

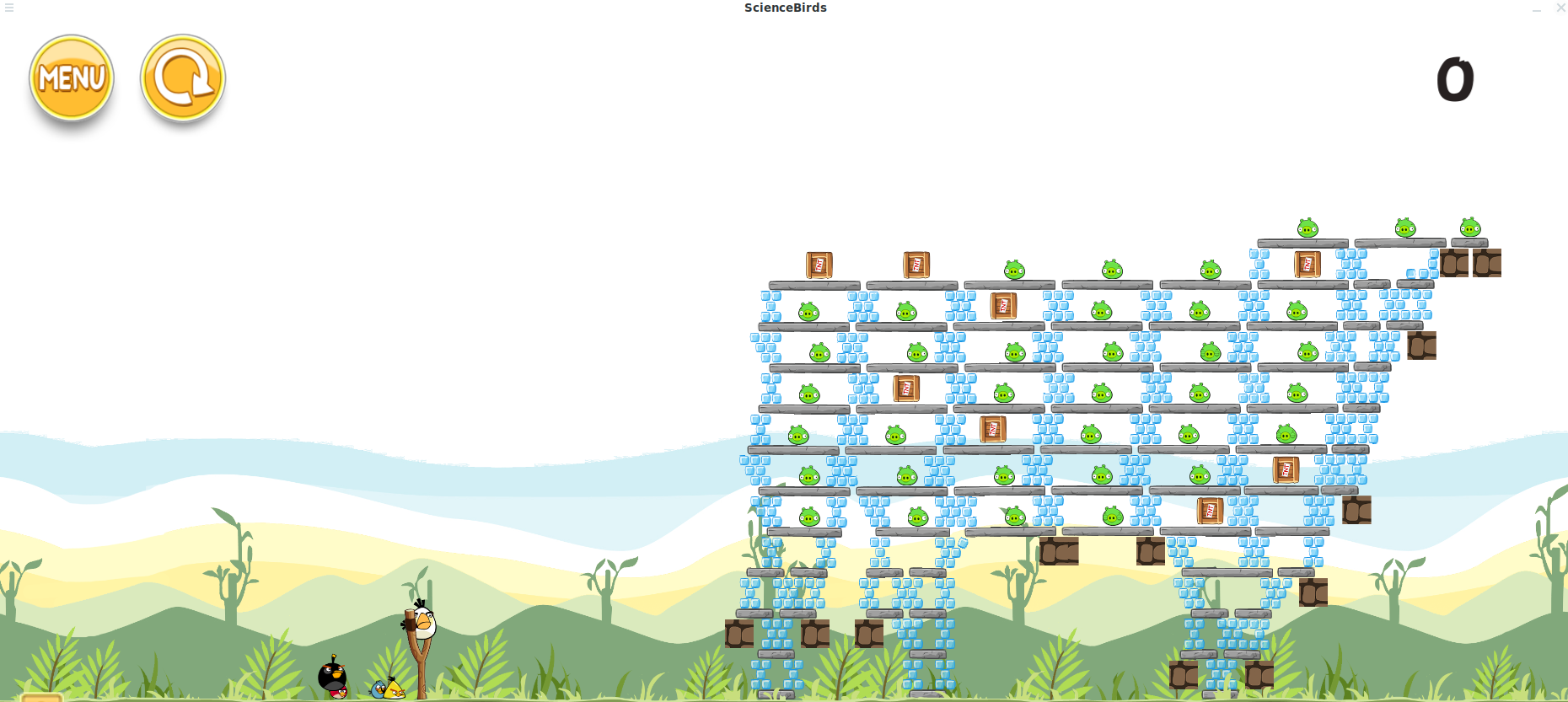
**3.3 Construct**

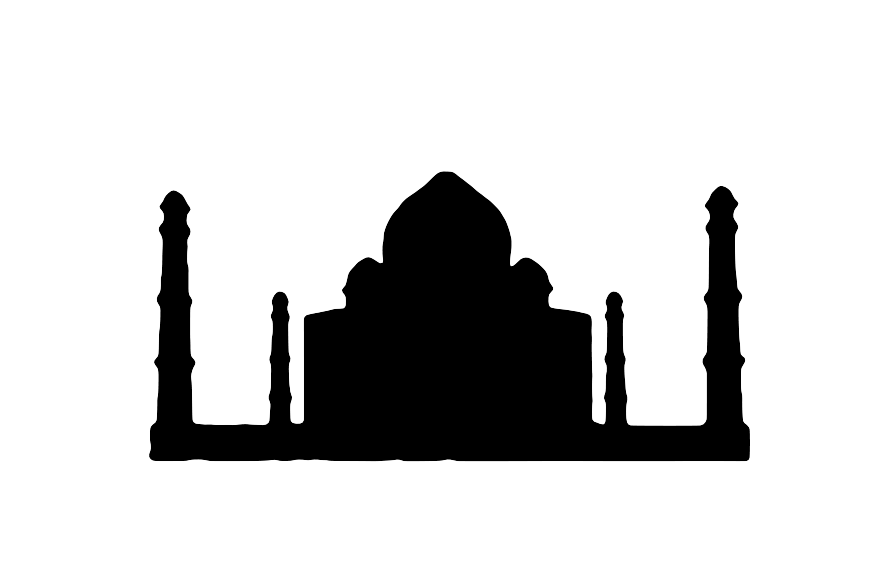
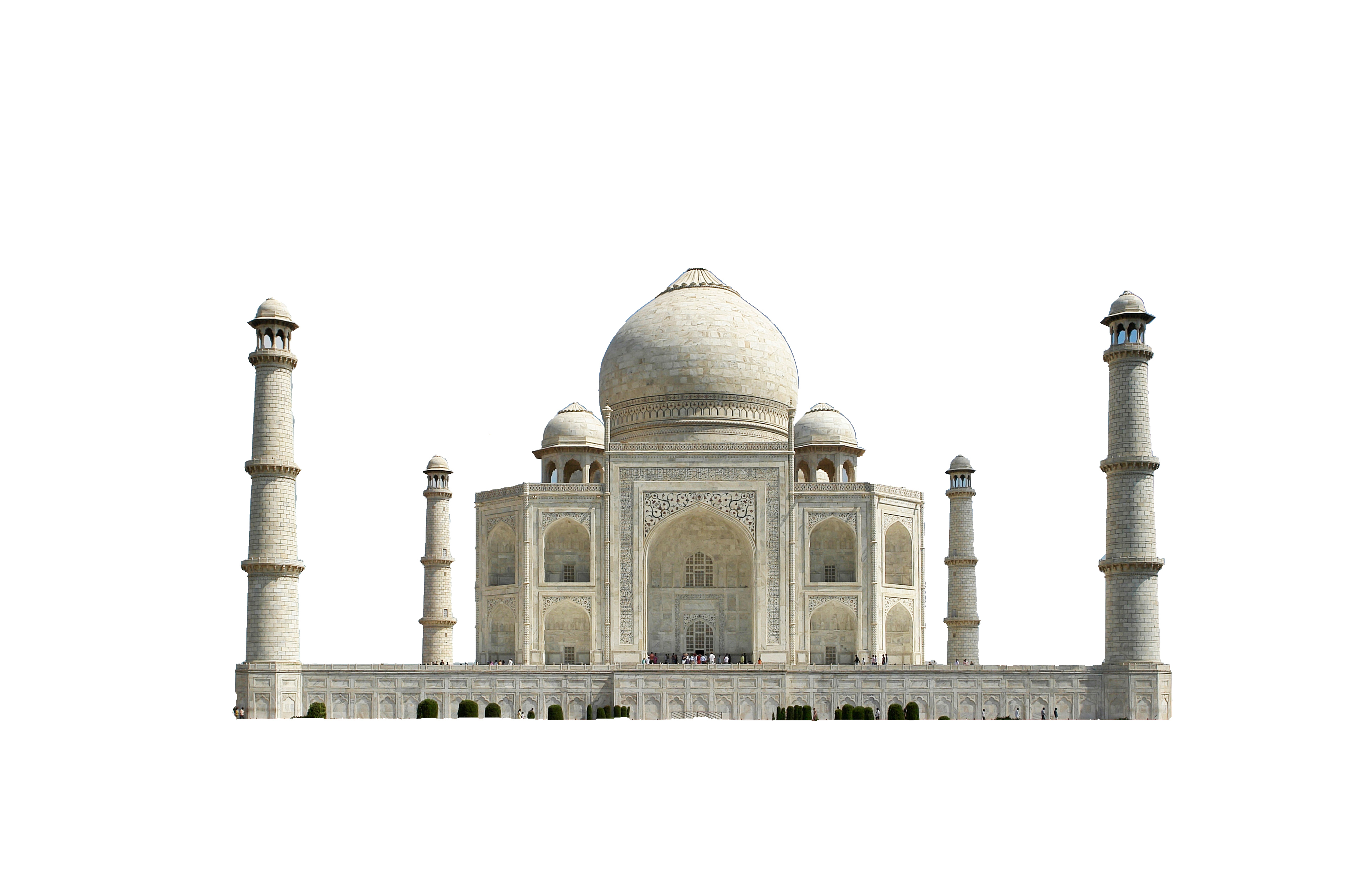
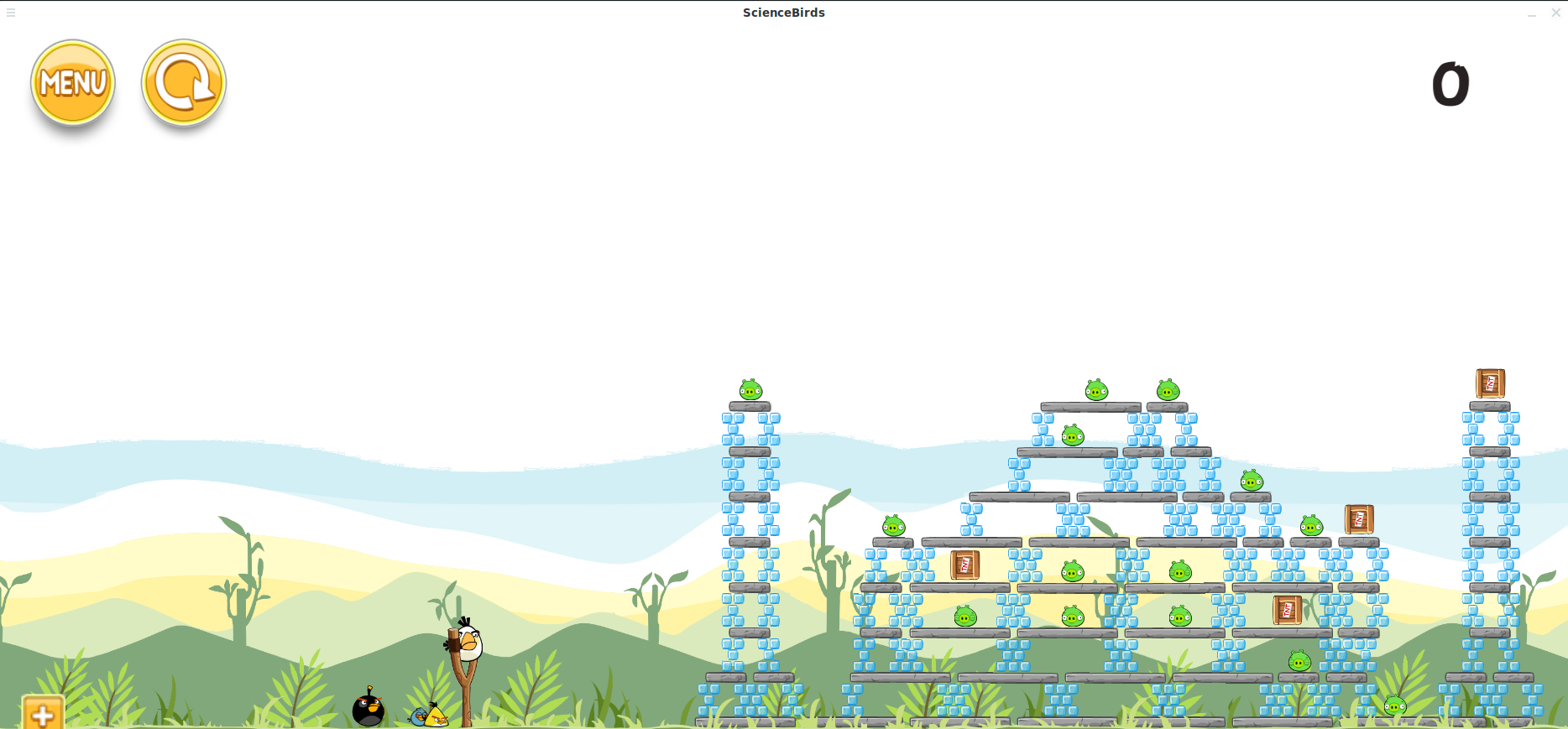
At last, we construct XML files by the value in the matrix. The program parses the value in the matrix and convert it to the objects in the XML file. The program shifts the objects in even rows by half of length of squares. Therefore, the structure looks like a honeycomb which each object gets more support from others.







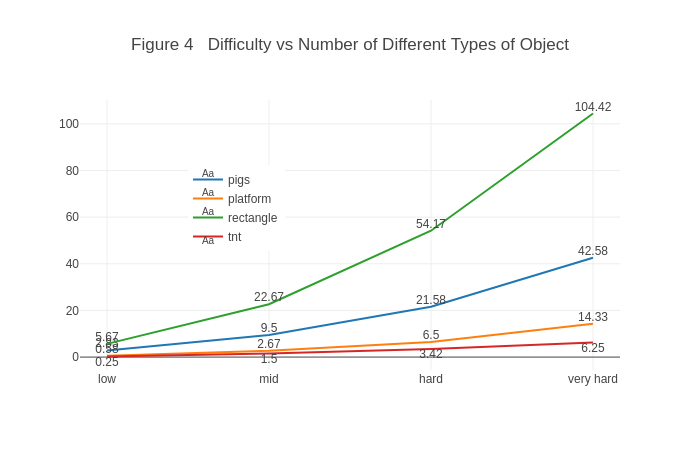
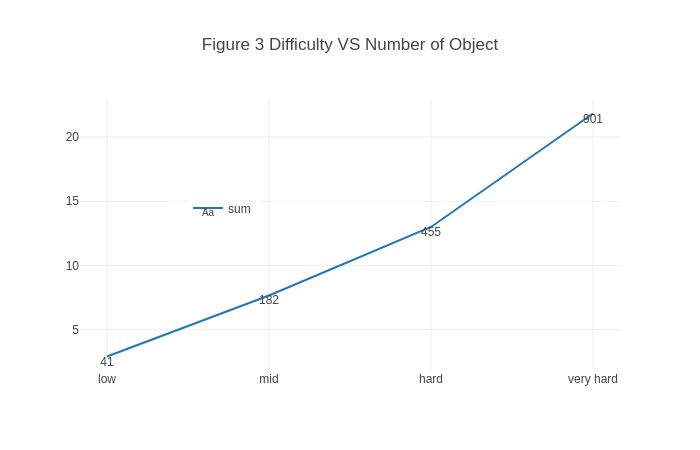


****

**3. Experiment**

In experiment, we made a number of tests to find out the object, clear rate, and stability of our structure in four different difficulty settings. First, we want to understand what factor impact the number of object in the structure.

In Figure 3, we found that the number of object have a positive relationship with the difficulty. Deeply, in Figure 4 we found that the player needs to kill average about 3 pigs in the lowest difficulty, 9.5 in mid, 21 in hard, and 42.58 in very hard. Therefore, we set the player can have three red birds in low level, which we expect players can kill one pig per shot. We place an additional black bird(boom bird) in mid level for player to denoting the TNT(average 1.5), another one additional black bird in hard level, and one more white bird in very hard.



Then, we test

**4. Conclusion:**

The program is designed to auto-generate playable levels in Angry Birds according to different real world images. One of the TensorFlow model, “Deeplab”, provides the matrix of object segmentation from input image. With these matrices, our physics model can construct the levels by using different type of objects in Angry Birds. The combination of deep convolution neural network and physics model is able to identify and extract objects segments and build structures similar to the objects in the images.

However, there still have multiple aspects in our model can be refine. The program currently can’t identify various building if the pictures only show the front side of the structures. Most of paper works for building detection is using satellite image instead of frontal image. The model lacks of ability to detect building in the front. Hence, there require more researches on detection and extraction of man-made structure in frontal image.

Also, when the input size of image is growing, our physical model possibly begins to ignore the key feature of objects since there only have a limit space in the game. Therefore, we plan to investigate how to use generation selection to select the best column and row as the input to the physics model in order to preserve the key feature from images.

**Reference:**

**[1] H. Gardner, Frames of Mind: The Theory of Multiple Intelligences. New York, NY, USA: Basic Books, 1983.**

**[2] P. A. Wałęga, M. Zawidzki and T. Lechowski, "Qualitative Physics in," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 8, no. 2, pp. 152-165, June 2016. doi: 10.1109/TCIAIG.2016.2561080**

**[3] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 834-848, 1 April 2018.  
doi: 10.1109/TPAMI.2017.2699184**