AI Birds Competition

Level Generation Track

Jochen Renz, Julian Togelius, Lucas Ferreira, Matthew Stephenson, XiaoYu (Gary)

This document presents a tutorial for installing all the necessary software for generating and testing levels using the AIBirds Competition software.

The software is composed of two parts: a clone of the original Angry Birds (called Weird Aliens) implemented in Unity and an example algorithm, implemented in python, to generate levels. The clone is totally independent from the generator, which means that one can create another generator in any programming language. The interface between the generator and the game is done via XML. Thus, at the end of the generation process, the competitors have to encode their levels into the XML format expected by the game. This document is organized as follows: Section 1 presents a simple tutorial for installing all the software, Section 2 presents in detail the XML format for loading levels. Section 3 has a tutorial of how to run the example generator and test them in the Weird Aliens game. Section 4 contains a more detailed explanation of the baseline generator that is provided.

1. Installation

1. Download Weird Aliens build associated with your operating system:
   - Windows: https://aibirds.org/Level-Generation/WeirdAliens-Windows-2.0.zip
   - OSX: https://aibirds.org/Level-Generation/WeirdAliens-OSX-2.0.zip
   - Linux: https://aibirds.org/Level-Generation/WeirdAliens-Linux-2.0.zip
2. Download the example of level generation algorithm in python: http://aibirds.org/Level-Generation/LevelGeneratorBaseline-2.0.zip
3. Run the Baseline level generator to create 20 example levels based on the parameter file.
4. Copy these levels to *StreamingAssets /Levels* directory.
5. Run the game and the newly generated levels can now be played from the selection menu.

2. Level Structure

The original Angry Birds have several game objects: birds, pigs, blocks, TNTs and other miscellaneous objects used mainly for decoration. There are several types of birds and they vary in colour and size. Some of them have special abilities that are triggered by touching the screen after a shot. The game also have several types of pigs, each of which having a different size and "health" points. Blocks have different materials, what impact in their physical properties and "health points". TNTs are used to cause explosions that deal damage in area, affecting several blocks and pigs. All these objects are placed on a terrain that can be either completely flat or complex. Weird Aliens currently supports only part of these objects:

- **Birds**: represented in the clone by different coloured smiling faces.
    - **Red**: Normal bird, no strengths or weaknesses.
    - **Blue**: Splits into three birds when clicked, strong against ice blocks.
    - **Yellow**: Shoots forward at high speed when clicked, strong against wood blocks.
    - **Black**: Explodes when clicked or after impact, strong against stone blocks.
    - **White**: Drops explosive egg when clicked.
- **Pigs**: represented in the clone by the green snake circles.
- **Blocks**: ice, stone and wood.
- **TNT**: represented in the clone as yellow squares.
- **Terrain**: Flat by default, but can be more complex using static square platforms.

All these objects can be seen in the level shown in Figure 2. This is the level 2 that is already part of the game. It has three blocks of each material, three pigs, a TNT block and five birds (one of each type). Moreover, it has two rows of static square platforms floating in the air.

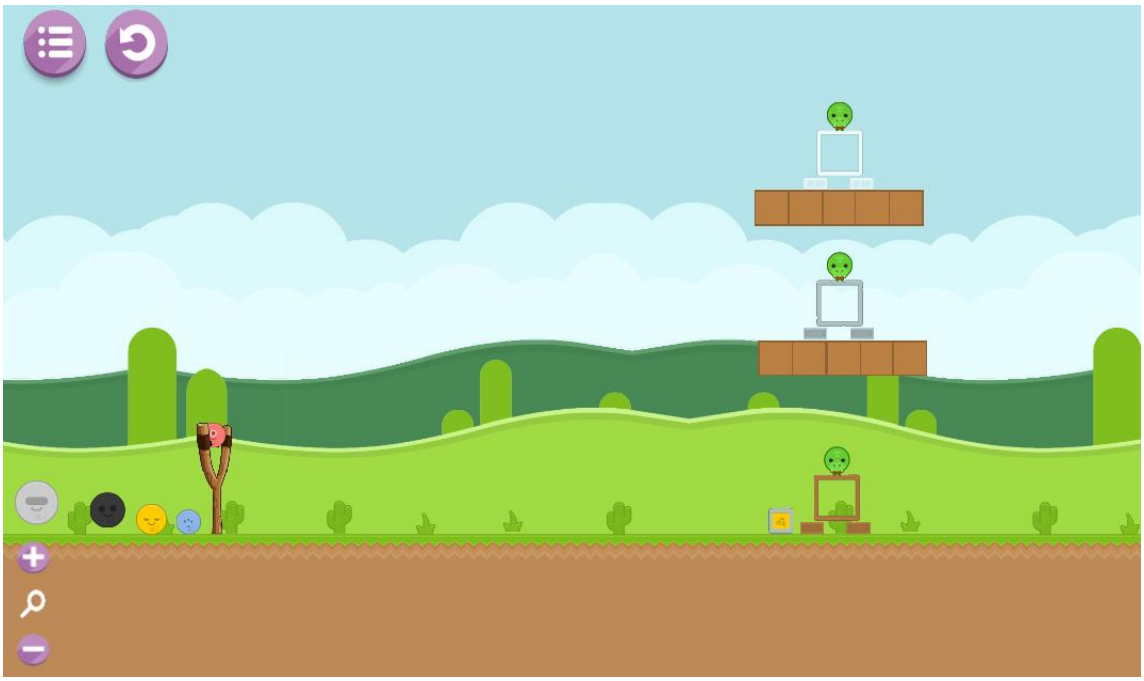

Figure 2: The second level of the Weird Aliens Game.

As mentioned before, levels are represented internally using a XML format. This format is basically composed by a number of birds and a list of game objects, as shown in Figure 3. Each game object has four attributes:

- **Type**: unique string representing the *id* of the object.
- **Material**: string defining the material of a block. Valid values are only "wood", "stone" and "ice".
- **X**, **Y***:* float numbers representing the position of the game object. The origin (0,0) of the coordinates system is the centre of the level.
- **Rotation:** float number that defines the rotation of the game object.

```xml
<?xml version="1.0" encoding="utf-16"?>
<Level width="2">
  <Camera x="0" y="0" minWidth="20" maxWidth="26">
    <Birds>
      <Bird type="BirdRed"/>
      <Bird type="BirdBlue"/>
      <Bird type="BirdYellow"/>
      <Bird type="BirdBlack"/>
      <Bird type="BirdWhite"/>
    </Birds>
    <Slingshot x="-9" y="-2.5">
      <GameObjects>
        <Block    type="RectTiny"    material="wood"  x="2.54" y="-3.23" />
        <Block    type="RectTiny"    material="wood"  x="1.70" y="-3.23" />
        <Block    type="SquareHole"  material="wood"  x="2.15" y="-2.68" />
        <Block    type="SquareHole"  material="stone" x="2.2"  y="0.72"   />
        <Block    type="RectTiny"    material="stone" x="1.76" y="0.17"   />
        <Block    type="RectTiny"    material="stone" x="2.6"  y="0.17"   />
        <Block    type="RectTiny"    material="ice"   x="2.6"  y="2.91"   />
        <Block    type="RectTiny"    material="ice"   x="1.76" y="2.91"   />
        <Block    type="SquareHole"  material="ice"   x="2.2"  y="3.46"   />
        <Pig      type="BasicSmall"  material=""      x="2.15" y="-2.12"  />
        <Pig      type="BasicSmall"  material=""      x="2.2"  y="1.38"   />
        <Pig      type="BasicSmall"  material=""      x="2.2"  y="4.12"   />
        <Platform type="Platform"    material=""      x="1.04" y="-0.31"  />
        <Platform type="Platform"    material=""      x="1.64" y="-0.31"  />
        <Platform type="Platform"    material=""      x="2.27" y="-0.31"  />
        <Platform type="Platform"    material=""      x="3.47" y="-0.31"  />
        <Platform type="Platform"    material=""      x="2.86" y="-0.31"   />
        <Platform type="Platform"    material=""      x="2.79" y="2.4"    />
        <Platform type="Platform"    material=""      x="3.4"  y="2.4"    />
        <Platform type="Platform"    material=""      x="2.2"  y="2.4"    />
        <Platform type="Platform"    material=""      x="1.58" y="2.4"    />
        <Platform type="Platform"    material=""      x="0.97" y="2.4"    />
        <TNT type="" x="1.11" y="-4" rotation="0" />
      </GameObjects>
    </Level>
```

Figure 3: XML representation of a Weird Aliens level

3. Generating and testing a level

In order to learn how to generate and test a level using the clone of the game, participants can use the baseline generator implemented in python (see Section 1 for download). The source code for the generator will also be accompanied by an input data file called *parameters.txt*. This file contains the necessary information about the levels that will be generated. This information is provided in blocks of four lines, with each block containing the following information in the given order:

Number of levels to generate	(positive integer)
Forbidden block and material combinations	(list of valid materials/block types, may be empty)
Range for number of pigs	(two positive integers, minimum and maximum)
Time limit to generate levels (minutes)	(positive integer)

For the time limit value this will typically be around one hour for every 10 levels and should not be an issue for most generators.

An example *parameters.txt* is provided along with the baseline generator.

During the competition, information about what levels to create will be passed to each generator using this input file, please ensure that your level generator uses this input file correctly.
Save the source code and execute it (you will need to have python installed). It will generate a collection of files, starting with *level-04.xml* and increasing upwards from there. To test and play these levels, copy each file to the *StreamingAssets /Levels* directory. Finally, open the scene *LevelSelectMenu* in the Weird Aliens game and press play. The menu will show the generated levels along with the base three provided.

4. Algorithm Overview

The baseline level generator provides a simple and effective method for generating levels within Weird Aliens. Several self-contained structures are placed within each level, either on the ground or on floating platforms. The number of ground and platform structures is determined randomly within a defined range. The available ground space within a level is divided amongst the number of chosen ground structures and a suitable structure created for each of these sections. After these ground structures have been generated we attempt to place the desired number of platforms within the level. A platform's location and width are determined randomly and several checks are made to ensure that it will not overlap any ground structure or be too close to the top of the level. If a suitable location for a platform cannot be found after a certain number of attempts then the platform is disregarded. A structure is then generated for each of the successfully placed platforms. Each structure is generated with respect to the area it will occupy. For ground structures this area is defined as the width of its allocated section along with two thirds the height of the level. For platform structures this area is defined as the width of its platform along with the distance between the platform and either the top of the level, or the closest platform above it (whichever is less).

Each structure is generated using the available blocks, with the probability of a block being selected based on the defined probability table. Weird Aliens currently contains eight standard rectangular blocks, five of which can be rotated 90 degrees to give a different shape. This creates a total of thirteen regular blocks which are used to build each structure.

The algorithm selects a block type at random using a probability table and uses this block to form the peak(s) of the structure. Currently there can be between 1 and 5 peaks but this number can be altered fairly easily. The distance between the peaks is also randomly selected. New rows are then recursively added at the bottom of the current structure using the method described below.

The blocks at the base of the current structure are split into subsets based on the distances between them, with all possible combinations being recorded:

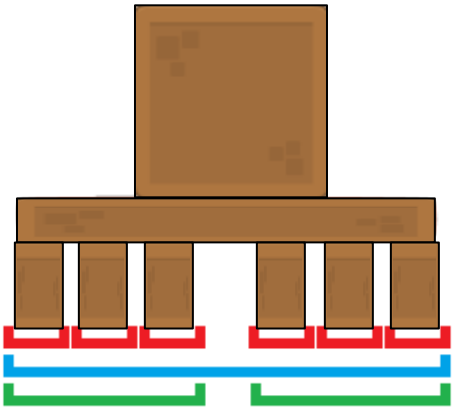

Figure 4: The bottom row of this structure has three possible subset combinations: each block is in a separate set (red), all blocks are in a single set (blue), and the three left/right blocks are partitioned into two sets (green).

A new block type is then selected at random. For each possible subset combination there are now three possible supporting block placement options:

- Blocks are placed underneath the middle of each subset.
- Blocks are placed underneath the edges of each subset.
- Blocks are placed underneath both the middle and edges of each subset.

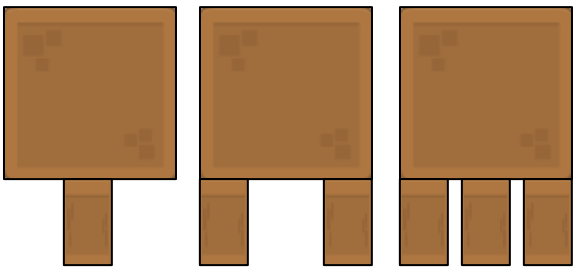

Figure 5: Three options for each subset possibility

The number of possible options for the new row is the number of subset combinations multiplied by three. For each of these options we place the new selected block type and test for two things.

1. No blocks are overlapping each other.
2. Each block on the bottom row of the current structure must be supported at both its edges from the blocks in the new proposed row (ensures stability).

We then randomly select an option from all combinations which follow these requirements and use that along with our selected block type to create a new row at the base of the structure. If no options are deemed valid then a new block type is selected. This process continues until the width or height of the structure is greater than the respective width or height values for the structure's area. Once this happens we remove the last row added and the structure is complete.

Once the structure has been created it is populated with pigs. The space directly above the middle and edges of each block is analysed to see if there is space for a pig to fit such that it doesn't overlap any other blocks. Any positions that are found to be big enough to place a pig are recorded. Next, we test all the possible ground positions that are within the structure (to a set precision). Again we check for any overlap with nearby blocks and valid positions are recorded. We then randomly choose a position from all the valid possibilities and place a pig at the given location. Any remaining pig locations that would overlap the newly placed pig are removed and another location is chosen at random. This continues until there are no more valid locations. This process ensures that the structure will always contain at least one pig, as a pig can always be placed on top of the structure's peak block(s).

After all structures have been generated and populated we attempt to remove or add pigs as required. If the number of pigs within the level is greater than the desired number of pigs then we randomly remove pigs until this restriction is satisfied. Any removed pigs have a random chance of being replaced by TNT. If the number of pigs within the level if less than the desired number of pigs then we randomly place pigs on the ground until this restriction is satisfied. The number of birds that are available to the user is based on the number of pigs and structures that are within the level. The type of each bird is selected randomly.
Once the pig numbers and locations are confirmed we attempt to place additional blocks within the level (TriangleHole, Triangle, Circle, CircleSmall). These blocks are placed in a similar fashion to the pigs. Valid locations are determined for each block type, after which a specific block is selected at random and any remaining blocks that would overlap this selected block are removed as valid possibilities. This continues until no more valid locations remain.

Lastly we convert this structure into the desired xml format and assign each block with a random material.

It is our hope that participants will be able to improve and enhance our baseline algorithm to create advanced level generator software. It is also possible for participants to create their own level generators from scratch, providing fresh ideas and insight into generating fun and exciting levels for Angry Birds.

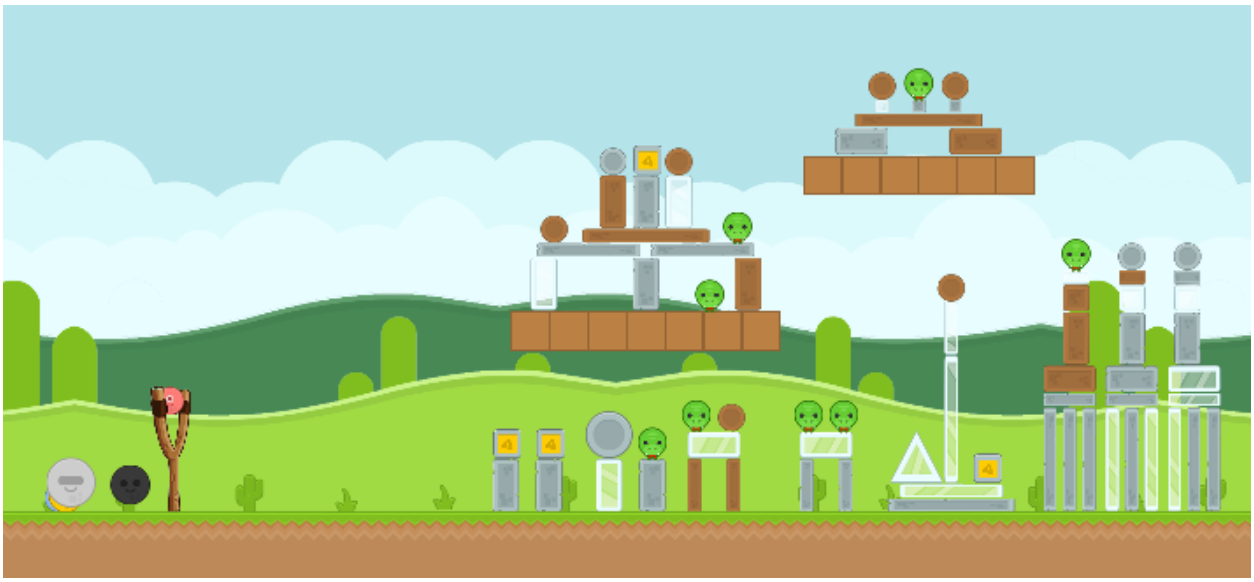

Figure 7: An example of a generated level