The Report Committee for Matthew Graves
Certifies that this is the approved version of the following report:

Procedural Content Generation of Angry Birds Levels Using Monte Carlo Tree Search

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Constantine Caramanis

Sarvesh Nagarajan

Procedural Content Generation of Angry Birds Levels Using Monte Carlo Tree Search

by

Matthew Graves, B.S.Computer Sci.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2016

Dedication

This work is dedicated to my wife, Christy, without whose patience, love, and support none of this would be possible. I also dedicate this work to God the Father and my Lord and Savior, Jesus Christ, who guide me daily in knowledge and truth by the Holy Spirit.

Proverbs 2:6-8 (ESV):
For the LORD gives wisdom;
from his mouth come knowledge and understanding;
he stores up sound wisdom for the upright;
he is a shield to those who walk in integrity,
guarding the paths of justice
and watching over the way of his saints.

Acknowledgements

I would like to express sincere gratitude to Dr. Constantine Caramanis, for giving technical guidance, advice, and support for this report and project. Dr. Caramanis's class on Data Mining and Optimization inspired me to look into concepts related to data mining, such as artificial intelligence, the multi-armed bandit problem, and decision processes, which drove the ideas behind this report.

I would also like to extend my appreciation to Sarvesh Nagarajan, who has been a constant mentor of mine at National Instruments and throughout my degree program, as well as a strong technical lead whose insights have helped me form this report. He has served as the reader for this report, ensuring the technical quality and content within meet high standards.

Abstract

Procedural Content Generation of Angry Birds Levels Using Monte Carlo Tree Search

Matthew Graves, M.S.E.

The University of Texas at Austin, 2016

Supervisor: Constantine Caramanis

Monte Carlo Tree Search is a method for searching a decision-making process, usually employed in domains such as general game playing, where an artificial intelligence agent must decide the next move to make in a game simulation. There have been other domains that have been explored for MCTS, one of them being procedural level generation, which involves the automatic generation of game levels which are interesting to play, of an acceptable difficulty level, and not discernible from levels created by humans. In this report, I present a method for using MCTS to procedurally generate new Angry Birds levels, trained on a set of Angry Birds levels from existing games. This approach will scale for a requested level of difficulty by using multiple heuristics. I will examine the viability of the approach using playouts of AI agents on the generated levels, each with their own approach to winning the levels, used to simulate the experience level of human players (from naïve to advanced).

Table of Contents

List of Tables

List of Figures

INTRODUCTION

Angry Birds is a popular puzzle-based mobile game series created by the company Rovio Entertainment. All of the games in the series have been downloaded more than 3 billion times as of July 2015 [1]. In the original game, players are asked to take the role of the Angry Birds, controlling a slingshot to fling birds into structures which house pigs (see Figure 1 for an example level). Points are awarded for destroying structures, defeating pigs, and having leftover unused birds from the ones provided at the start of a level. As the player progresses through the levels, they unlock different types of birds with special abilities, and the levels get more difficult, with more blocks arranged in structures that are more difficult to destroy.

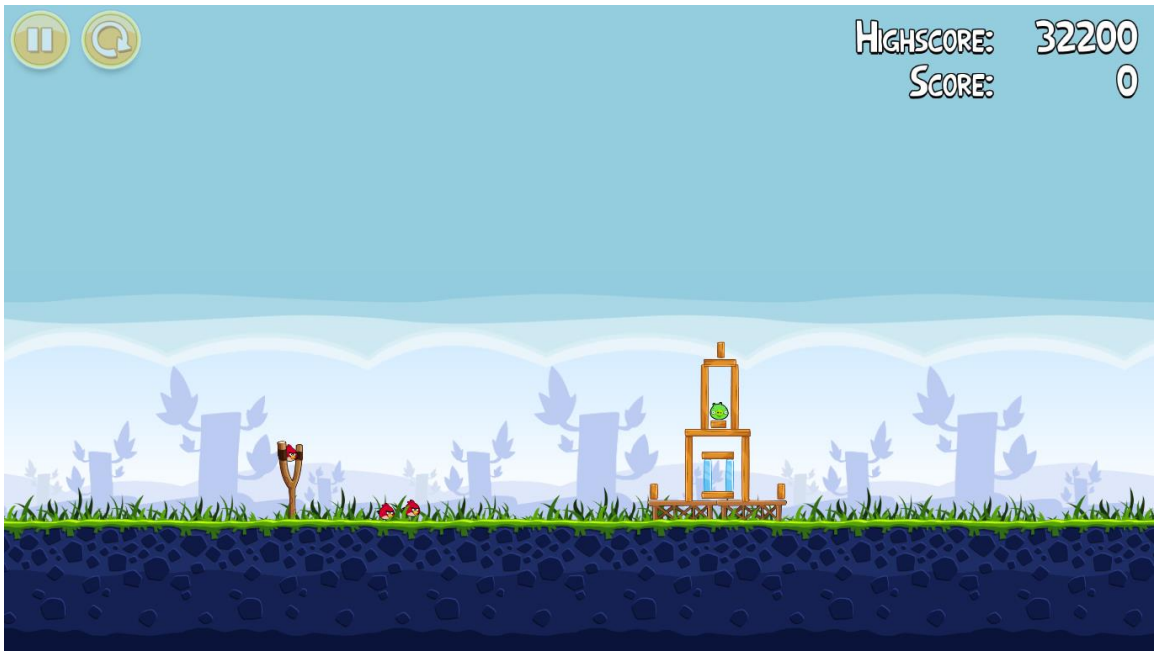

Figure 1: Level 1-1 of Angry Birds.

Due to the popularity of the game, many different areas of research in game AI have surfaced in the past few years around Angry Birds. General game playing has been a major research area, involving the creation of artificial intelligence agents that are used to play the game. In this paper, I will be covering another research area, procedural content generation, as it applies to Angry Birds level creation.

Chapter 1: Background

Procedural Content Generation

Procedural content generation is a mechanism to create data using algorithms, typically employed in the domain of video games to generate graphics, models, music, or some other aspect of gameplay. Usually, this approach is taken in order to provide variety and challenge, and has the side benefit that a developer does not need to manually design the content ahead of time. Some games, such as Minecraft [2] and the recently released No Man's Sky [3], rely on procedural content generation for game depth and variety. Minecraft uses procedural content generation to create topological maps of a game level's surface, making use of varying biomes (forest, desert, snow, etc.) and different wildlife for the user to explore and develop to their will, creating a "sandbox" nature to the game. No Man's Sky is a sci-fi exploration game, reportedly including a procedurally generated 18 quintillion planets [4]. Currently, procedural content generation is not being applied to the game Angry Birds, but that provides an interesting opportunity to develop algorithms that can make interesting and exciting levels, where normally a game designer or programmer would have to form these levels by hand.

AIBirds

AIBirds is a research community centered around game AI. Since 2012, they have put together competitions, created game AI simulation frameworks, and had a presence at multiple conferences, including CIG (Computational Intelligence and Games Conference), IJCAI (International Joint Conference on Artificial Intelligence), and ECAI (European Conference on Artificial Intelligence). This year, the community held a competition for procedural content generation of Angry Birds levels, taking place during CIG 2016 in Santorini, Greece on September 20-23, 2016 [5]. This competition included

both a “fun track,” where the enjoyment of levels was judged by a panel of judges, and a “hard track”, where the goal was to create difficult but solvable levels. Due to the timing of the writing of this report, I was unable to enter the competition, but the some of the concepts and frameworks used in this project were gathered from the AIBirds community.

MONTE CARLO TREE SEARCH (MCTS)

Monte Carlo Tree Search is an algorithm used to explore decision spaces, making use of different heuristics to weight the potential value of a given state in a decision tree. MCTS has made a large splash in the realm of AI and games. Even without knowing the game rules, an MCTS-based AI agent can learn the most promising moves during gameplay and still win, given lower-level heuristic information to quantify the value of a state (is the game over, what is the current score, did the player win or lose, etc.). This search problem is called general game playing.

The challenge behind search-based algorithms is managing the principles of exploration and exploitation. A greedy algorithm will always choose the most promising moves that are directly in the states following the current game state, which would heavily favor exploitation rather than exploration. To properly balance the two principles, a random element of exploration is introduced in MCTS, which is guided by various heuristics to determine the value of game states.

MCTS, in general, works using four steps [6]:

- Selection: Begin at the root node, and choose the child with the most potential in succession until a leaf node is reached.

- Expansion: If the leaf node that was reached has any moves that have not been attempted, a random action from the remaining possible actions is chosen at random, and the action is taken.
- Rollout: Additional actions are applied at random from the current state until a condition occurs that terminates this step (for example, a timeout).
- Backpropagation: Calculation of a "score" which weights the value of a rollout is calculated, then gets propagated up to all parent nodes until the root is reached.

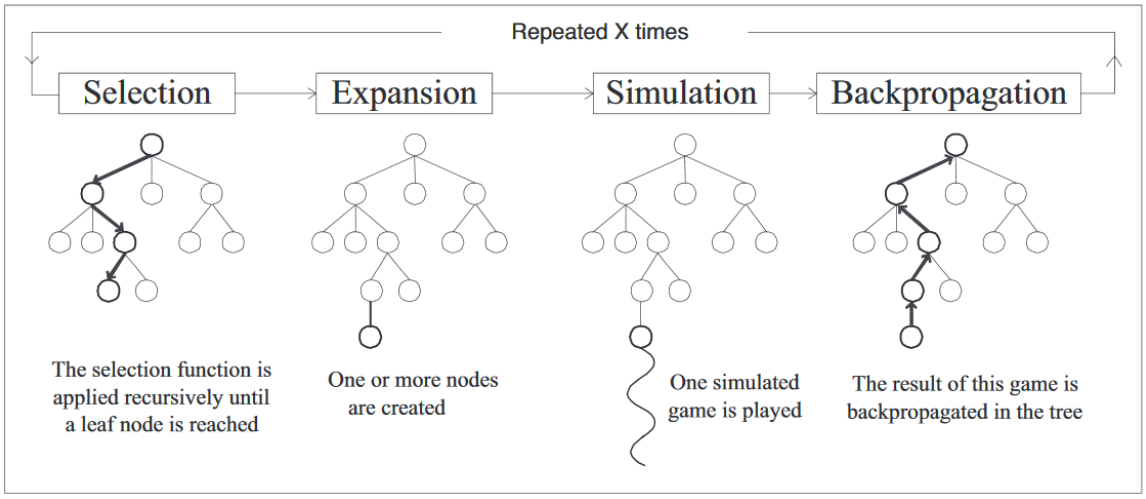

Figure 2: A graphical representation of Monte Carlo Tree Search [6].

There are certain tunable parameters in MCTS. The most important parameter is the Selection strategy, which is used to pick which node has the most potential. A common approach for this and the approach used in this project is Upper Confidence bound 1 applied to Trees (UCT) [7]. In UCT, the following formula is used to weight the potential of a node:

$$UCT_i = \frac{w_i}{n_i} + c\sqrt{\ln t / n_i}$$

where:

- w_i represents the number of "wins" after the ith move
- n_i represents the number of simulations after the ith move
- c represents the exploration parameter (theoretically, $\sqrt{2}$, but usually depends on the application)
- t represents the total number of simulations for the node (eg. the sum of all n_i values)

In this project, I used the UCT approach to assign potential value to Angry Birds level generation nodes. The method used to calculate scores, the condition used to terminate a rollout, and other parameters provided to MCTS will be discussed in the Approach and Implementation section.

Chapter 2: Relevant Work

As was stated earlier, there are research communities dedicated to the advancement of AI research related to procedural content generation and search-based algorithms in video games. I drew inspiration and ideas from two prior research papers [8] [9]. The first paper applies MCTS for procedural content generation of platformer game levels, and was helpful to understand the concepts of level difficulty and proper level layout, as well as the application of MCTS to procedural level generation. The second paper uses a genetic search-based algorithm to generate Angry Birds levels, and provides an Angry Birds clone which the AIBirds competition uses to simulate games.

MONTE CARLO TREE SEARCH TO GUIDE PLATFORMER LEVEL GENERATION

There has been prior work using MCTS to procedurally generate game level content. One of the main approaches has been MCMCTS [8], which uses Markov chains to create platformer levels. This paper's main focus was platformer levels, namely, Super Mario Brothers levels. It's important to realize that platformer games, such as Super Mario Brothers, have different requirements than the puzzle genre of Angry Birds. Platformer levels make use of a large amount of static geometry (eg. elements of the level which are not affected by gravity), whereas in Angry Birds the platforms are the exception, with the rule being blocks that are affected by gravity and physics. There are similarities, however. For example, the difficulty of both Angry Birds and Super Mario Brothers has a common factor which depends on the amount and size of gaps between blocks. This difficulty aspect allows more maneuverability for Mario and more strategic potential for the firing path of Angry Birds, benefiting the player's progress by giving the player more options for movement.

A Search-based Approach for Generating Angry Birds Levels

There have been some past attempts at procedural generation of Angry Birds levels, and these approaches vary quite a bit. This paper [9] was one of the first to apply procedural generation to Angry Birds levels, using a genetic algorithm as the level generation algorithm. The paper goes into detail about an evolutionary approach that initializes a population of randomly-chosen level elements, then performs crossovers of the elements to mutate the populations and try new ones that better fit their proposed fitness function. This paper is also the source of the AIBirds clone that is being used in this paper to simulate runs of Angry Birds via the Unity game engine [10]. It employs many similar visual and audio assets, as well as the mechanics of the original game, so researchers can "plug in" their algorithms and experiment with different aspects of gameplay and level formats.

APPROACH AND IMPLEMENTATION

To implement a procedural Angry Birds level generator using MCTS, I employed use of the aforementioned framework provided by the AIBirds community to represent the levels, using the content of existing levels as input to the MCTS algorithm. This framework allowed use of a given set of basic blocks to form levels, providing an XML-based level format to plug in to a playable Angry Birds clone written in the Unity environment [11]. Instead of using the basic blocks, I developed a program to parse existing levels for their block-based structures, and then developed derivative unique structures for my levels based on those existing structures.

The following figure denotes the architecture of the MCTS-based procedural level generation system. Each piece will be explained in detail in the following sections.

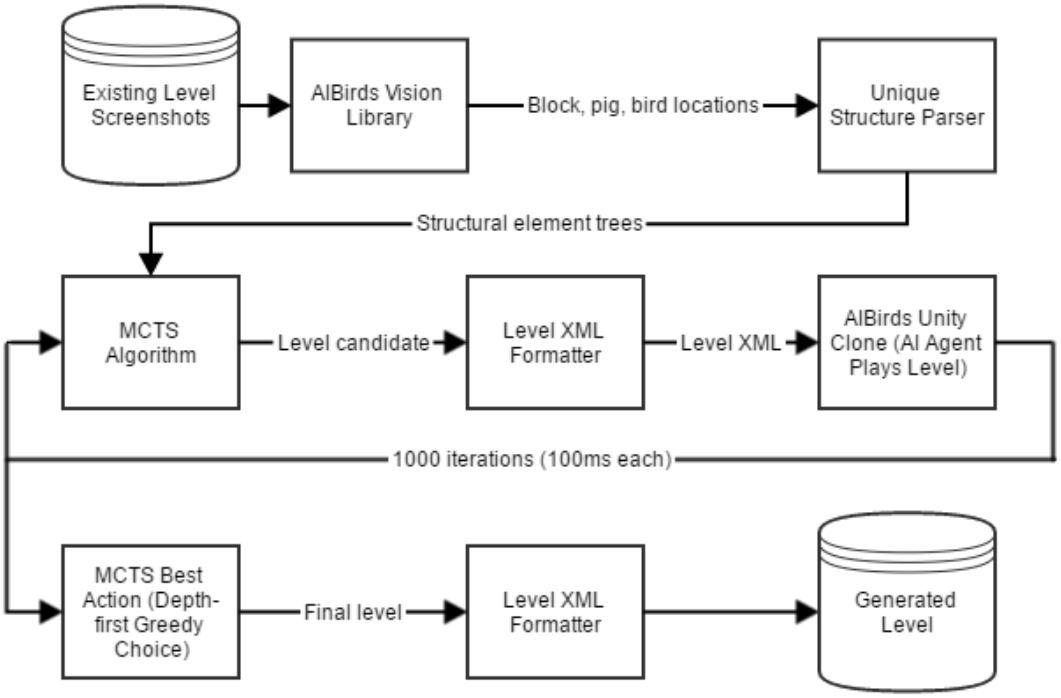

Figure 3: Architecture diagram of MCTS-based procedural level generation.

Chapter 3: Level Formatting

Before discussing the algorithm behind putting a level together using MCTS, we must first decide what blocks and structures we want to work with and support in the generated Angry Birds levels. My approach uses existing Angry Birds levels as inspiration to drive the structures used in level generation.

Basic Blocks

At the core of procedural content generation is the arrangement of basic level elements to produce fully-featured levels for the user to play. There are quite a few methods of doing this, and the methods vary based on the genre of game. In Angry Birds, the most basic elements are squares, rectangles, circles, and triangles, made of the materials wood, ice, or stone (these elements have physics applied to them). The figure below denotes the images that represent the various blocks supported by the AIBirds research clone. The blocks in green are the ones this procedural level generator supports

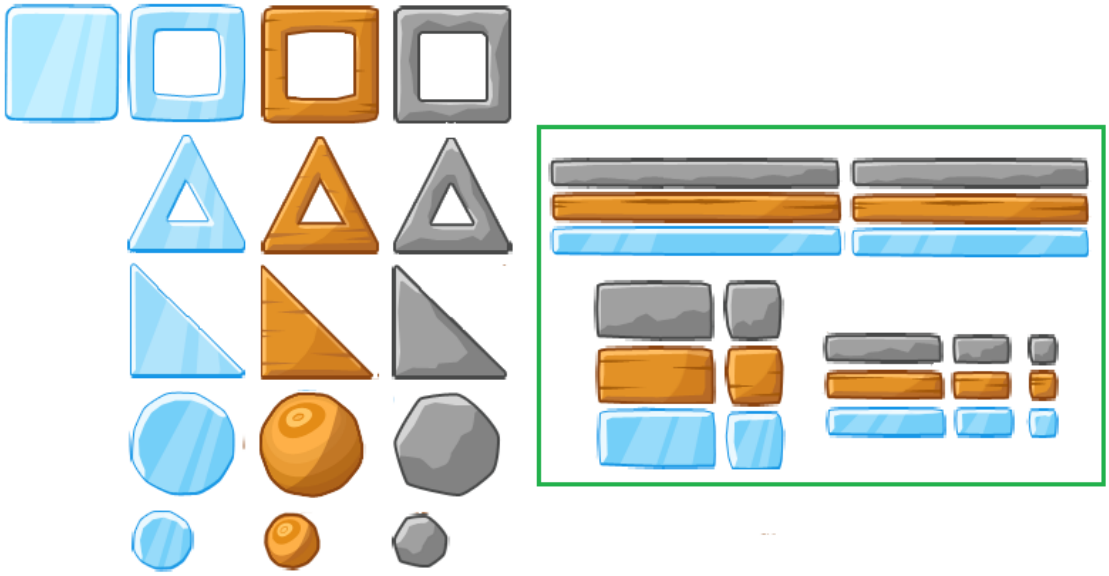

Figure 4: Supported blocks in the AIBirds Angry Birds research clone [11].

and uses. There are also platforms (which have no physics applied to them, so structures can stand above the ground), pigs, and birds.

LEVEL PARSING

The AIBirds organization provides a computer vision library [12] to analyze images, video, and gameplay of existing Angry Birds levels. Usually, this would be used to write an AI agent in order to play the game, but for level generation purposes, I simply use this library to determine the layout and organization of the blocks in existing levels. The library denotes a bounding rectangle around each basic block, which is used to establish the location of each block, its rotation, and what type of block it is. This information from existing levels is fed in to the unique substructures algorithm, described in the sections below.

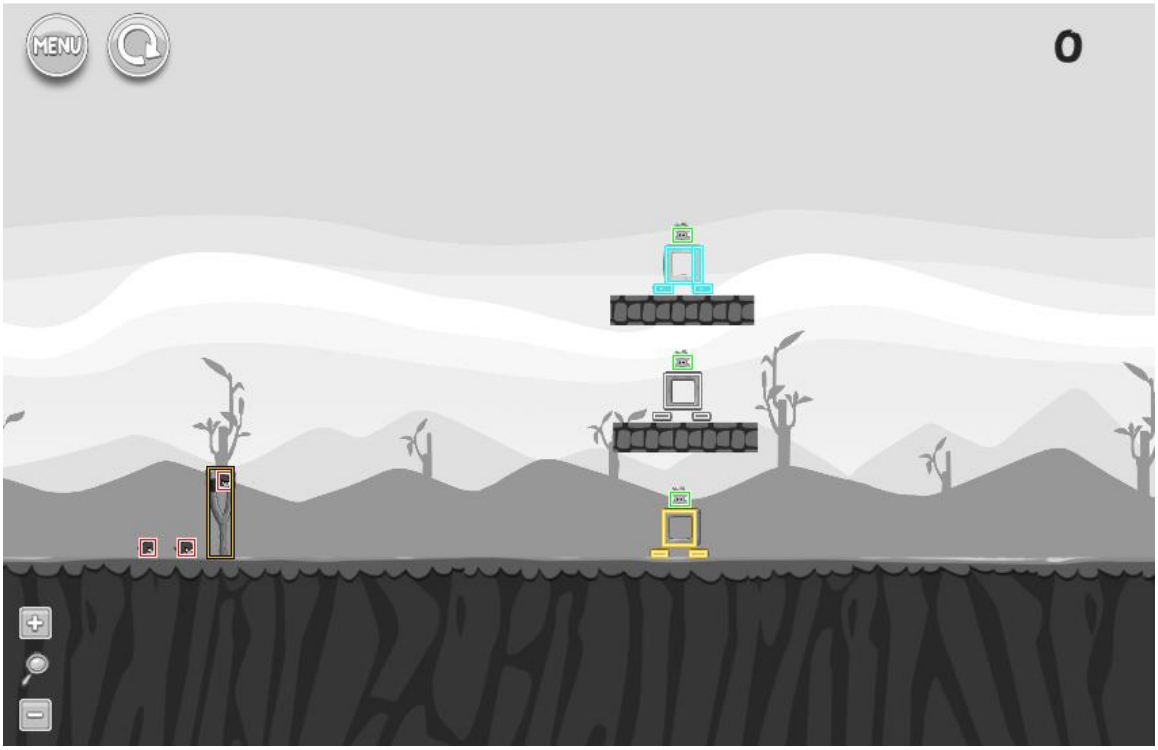

Figure 5: Example output of the AIBirds vision library.

EXISTING STRUCTURES

As mentioned in the prior section, all blocks in Angry Birds (except platforms and the ground of the level) have physics applied to them. This makes levels that are created from the basic level blocks (eg. from the ground-up) more difficult to create. The elements have to be stacked correctly in order for the structure to not fall over before the user has even had the chance to fire a bird. To solve this problem, the MCTS generator uses portions of the structures from existing levels in the Angry Birds – Poached Eggs collection. The structures contained in these levels are already proven to stand on their own in terms of the physics engine. However, it wouldn't be very interesting or fun for the player if the same full structures from the existing levels were used, just in different combinations. Therefore, a way to uniquify these structures is needed.

UNIQUE SUBSTRUCTURES

In an effort to remove the repetition involved with reusing the existing structures from Angry Birds, I parse the existing levels, and then apply an algorithm to create data structures that are subsets of the original structures. Starting at a block which has no block above it within a given structure (the "root node"), and then traversing down the structure until I hit either the ground or a platform (which acts as ground in midair), I form a tree which represents an Angry Birds level structure. This way, an individual structure is made up of many substructures, each of which could stand on its own. I can then apply a breadth-first traversal of this tree to derive many unique substructures of the original structure, which can then be arranged using Monte Carlo Tree Search. These structures have a high probability of also standing when physics is applied, since every block underneath a given block is included in the substructure.

The following figure shows an example of an existing Angry Birds structure with the computer vision library enabled, Each frame enumerates all of the possible

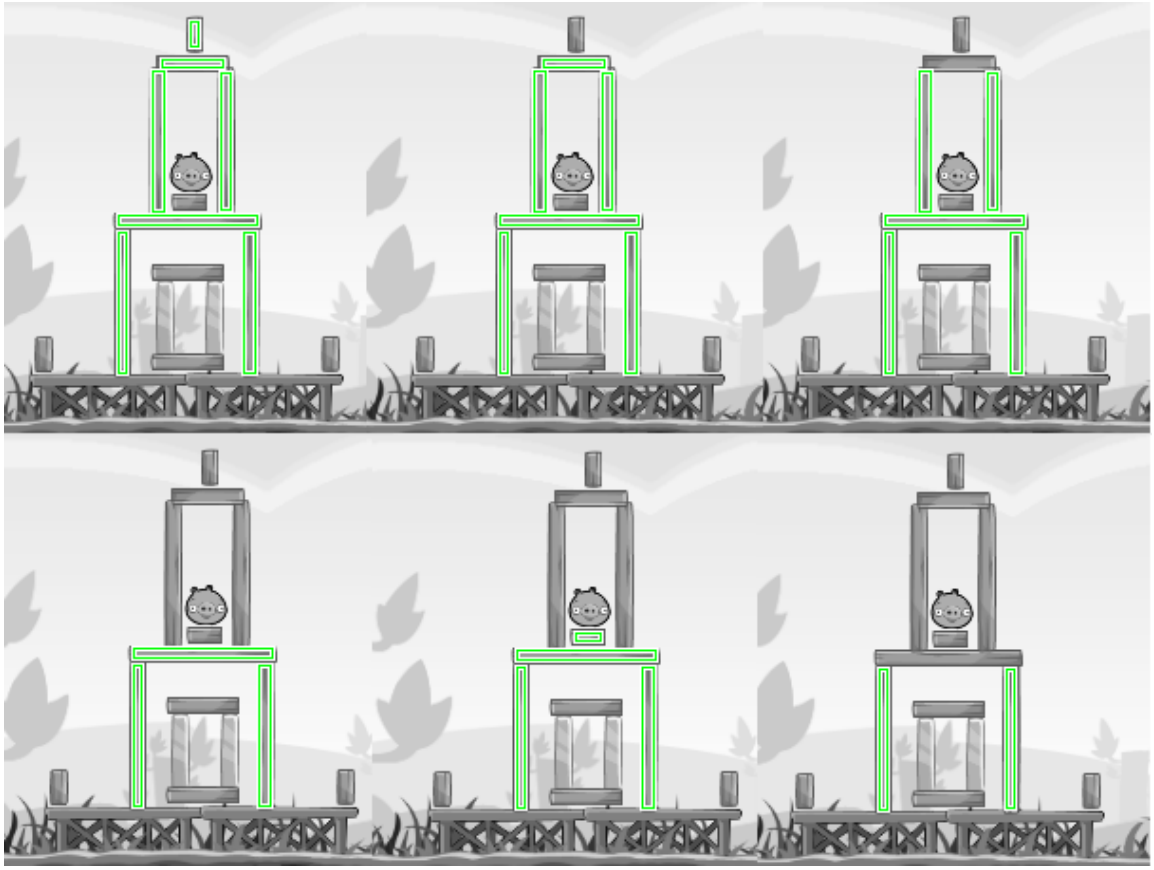

Figure 6: Possible substructures of outer structure by breadth-first search.

substructures that can be created from that structure. The structure inside this structure (with the four blocks arranged in a rectangle) would be considered its own standalone structure, and would be part of a different iteration of the parsing process.

XML Level Format

The AIBirds competition level format uses XML to describe level elements. Top-level elements under the <Level> tag include a <BirdsAmount> tag that indicates the total amount of birds in the level, as well as a <GameObjects> tag, which contains three

different types of tags: <Block>, <Pig>, and <Platform>. A block is a piece of physics-based geometry, with various shapes and sizes. A pig is a special type of block that denotes a foe to be destroyed in the level. A platform denotes a piece of non-physics geometry that holds structures up in midair. The following figure shows an example file for an AIBirds level.

```
<?xml version="1.0" encoding="utf-16"?>
<Level>
    <BirdsAmount>4</BirdsAmount>
    <GameObjects>
        <Block type="RectMedium" material="wood" x="2.78" y="-2.68" rotation="0" />
        <Block type="RectFat" material="wood" x="2.75" y="-2.16" rotation="90.00001" />
        <Block type="RectFat" material="wood" x="3.42" y="-2.35" rotation="0" />
        <Block type="RectFat" material="wood" x="2.03" y="-2.34" rotation="0" />
        <Block type="SquareHole" material="stone" x="2.04" y="-1.7" rotation="0" />
        <Block type="SquareHole" material="stone" x="3.43" y="-1.7" rotation="0" />
        <Block type="SquareHole" material="wood" x="-0.07" y="-2.3" rotation="0" />
        <Block type="SquareHole" material="wood" x="-0.05" y="-1.45" rotation="0" />
        <Block type="SquareHole" material="wood" x="-0.05" y="-0.59" rotation="0" />
        <Block type="SquareHole" material="wood" x="5.3" y="-2.34" rotation="0" />
        <Block type="SquareHole" material="wood" x="5.3" y="-1.48" rotation="0" />
        <Block type="SquareHole" material="wood" x="5.32" y="-0.64" rotation="0" />
        <Block type="RectSmall" material="wood" x="1.51" y="-2.67" rotation="0" />
        <Block type="RectSmall" material="wood" x="4.05" y="-2.68" rotation="0" />
        <Pig type="BasicSmall" material="" x="2.74" y="-1.48" rotation="0" />
        <Pig type="BasicSmall" material="" x="-0.04" y="0.06" rotation="0" />
        <Pig type="BasicSmall" material="" x="5.32" y="0.03" rotation="0" />
    </GameObjects>
</Level>
```

Figure 7: Example level using AIBirds XML level format.

Chapter 4: Monte Carlo Tree Search

To lay out the Angry Birds levels, I used the Monte Carlo Tree Search algorithm. As explained before, the MCTS algorithm uses a combination of exploration and exploitation to decide what choices to make at each game state. The way it decides this is through use of heuristics, which weight the perceived value of each state. In this section, I will cover the heuristics that were developed in this project and what their impact is on the generation of levels, as well as the overall process the algorithm goes through (eg. the level generation loop).

HEURISTICS

There are a few heuristics that must be provided to the MCTS algorithm to tell the algorithm what a "good" level state includes. One of the main heuristics for Angry Birds is a difficulty level (easy, medium, or hard), which is taken in as input. Difficulty settings are denoted by the number of ideal blocks that the levels would have: easy levels have an ideal number of blocks that is less than medium levels, which have an ideal number of blocks that is less than hard levels. A similar approach follows for the ideal number of pigs and birds in the level.

Another heuristic is whether the level is solvable or not. This is determined by playouts of AI agents, described in a later section. If the level is solvable, it is awarded a large amount of points, and if it is not solvable, the same points are deducted from the overall state score.

Other optional heuristics that were added included rewarding points for levels with pigs which are inside structures (not just on top of them, as in Figure 8), and some

Figure 8: Example of a pig inside a structure.

heuristics that can be enabled based on the desired shape of the level. The possible shapes that I implemented were a bell curve, following the normal distribution as in Figure 9, and a square root curve, as in Figure 10. All blocks that follow this heuristic get extra points if they lie under the curve.

Figure 9: Bell curve shape heuristic (maximum score).

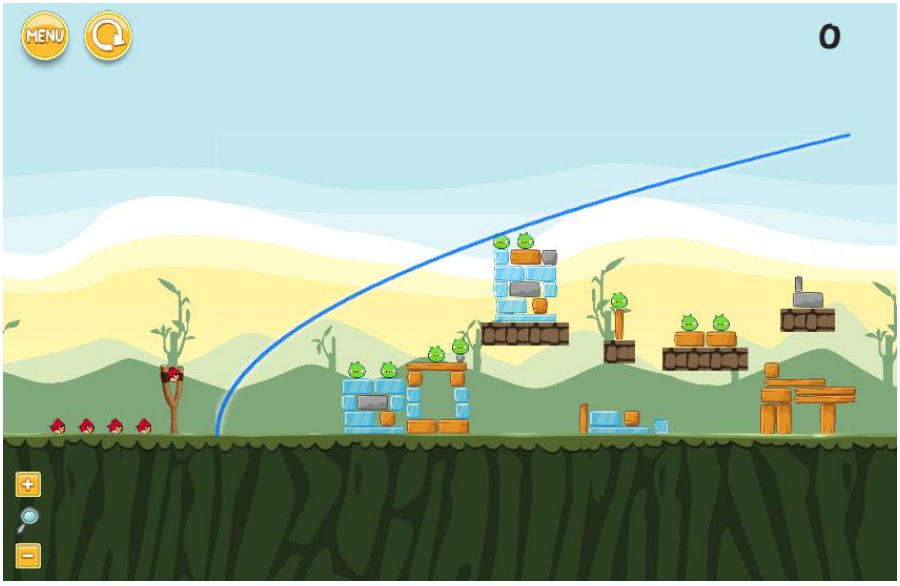

Figure 10: Square root shape heuristic (maximum score).

Finally, I implemented a heuristic that takes points away if structures have no pigs on them. This will help suppress uninteresting parts of levels that serve no purpose except to get in the way of the player's progress, especially with unnecessary platforms that can't be toppled by fired birds.

The following table describes the heuristics that were implemented, their ideal values, and how many points are possible to be awarded for each heuristic. The relative weighting of each heuristic is represented by the ratios of their corresponding awarded points.

Heuristic	Ideal Value	Points Awarded
Level play result by AI agents	Level won by Naïve agent (Easy), Level won by both agents (Medium and Hard)	Win: 10000000 Loss: -10000000
Number of blocks	30 (Easy), 50 (Medium), 70 (Hard)	7000
Number of pigs	4 (Easy), 5 (Medium), 6 (Hard)	5000
Number of structures without pigs	0	3000
Pigs within structures (not on top of structures or on ground)	Number of structures	2000
Level shape (Gaussian or square root)	All structures under shape	2000

Table 1: Heuristics that are supported by the MCTS procedural generator.

LEVEL GENERATION LOOP

The creation of the level takes place using a loop. MCTS has the following options for an action to take at each iteration of the loop: place a structure, place a pig, or nil (the "do nothing" action). Level layout begins at the lowest-left part of the level, and proceeds to lay out structures programmatically in a column until the top of the level is reached. Once the top of the level is reached, the y location of the next structure is reset to the ground level, and the x location is incremented by the width of the current column (see Figure 11). A pig can be placed on or in an existing structure, or on the ground. A nil

Figure 11: "Zig-zag" approach the MCTS algorithm uses for level layout.

action increments the y-pointer by a constant height, but does nothing else.

The program begins with 1000 training iterations, in which the MCTS algorithm explores as much of the state space as possible, while exploiting good designs that match

the heuristics stated earlier. At the end of these training iterations, the program descends the tree, starting at the root (bottom-left of level), and takes every action that MCTS has found to be of the highest value (the “best” action) until reaching the top-right.

RESULTS

As a test of the generated Angry Birds levels, I generated levels with differing levels of difficulty, using two different AI agents (the AIBirds default provided naïve agent and one former AIBirds gameplay contest entry) to play the levels and determine if they actually met the difficulty level specified and were beatable. 100 different levels were generated for each difficulty level, each using 1000 MCTS testing iterations with 100ms allocated to each iteration.

Chapter 5: AI Agents

The agents that were chosen for this test have been provided by the AIBirds community as example agents. The agents chosen are the naïve agent and the DataLab agent [13]. The naïve agent is included with the AIBirds competition framework as a very basic agent that is meant to get the user familiar with how the framework works. The DataLab agent, on the other hand, was developed by a team that competed in the AIBirds AI agent competitions for the past few years (even winning in 2014 [14] and 2015 [15]), and is more sophisticated than the naïve agent, in an effort to scale with what could be considered increasing levels of human capability to solve the levels.

NAÏVE AGENT

The naïve agent's algorithm is simple: aim directly at a random pig. Obviously, this approach is limited, and should only succeed on easier levels. This agent occasionally chooses a high trajectory to hit the pig, in order to avoid hitting platforms that may be in the way of the direct firing trajectory. Two possible trajectory options for a pig are shown in the following figures.

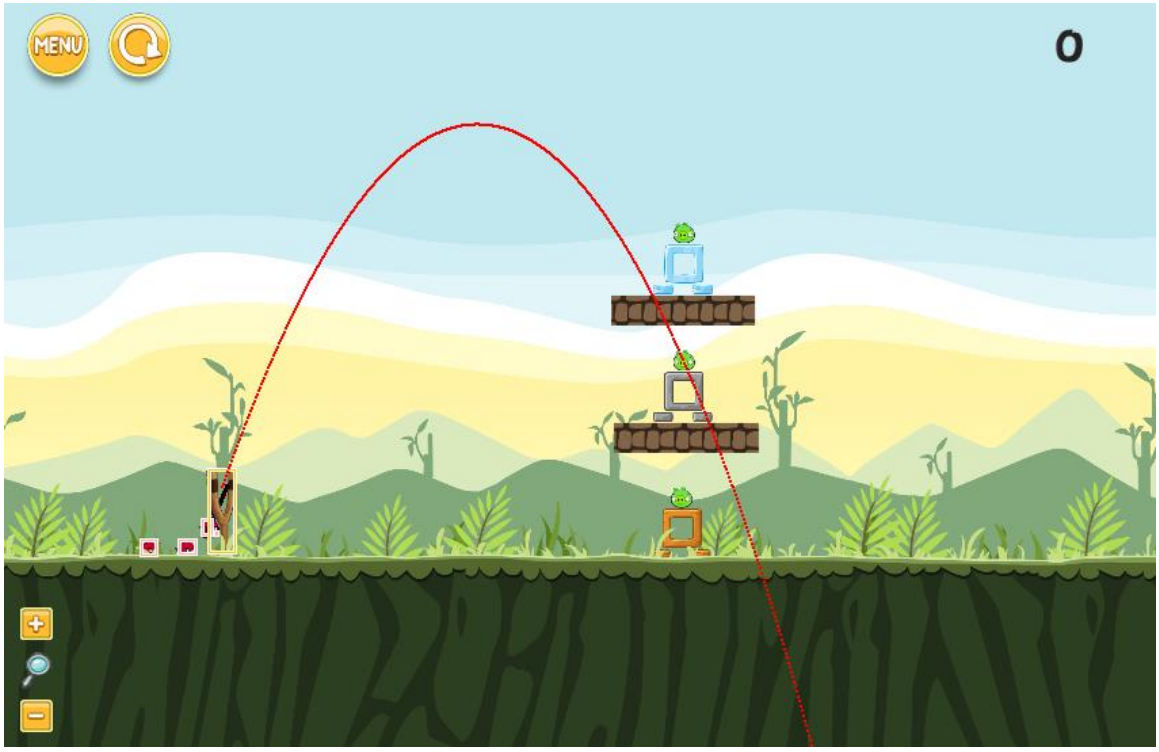

Figure 12: Alternative high approach option for naïve agent.

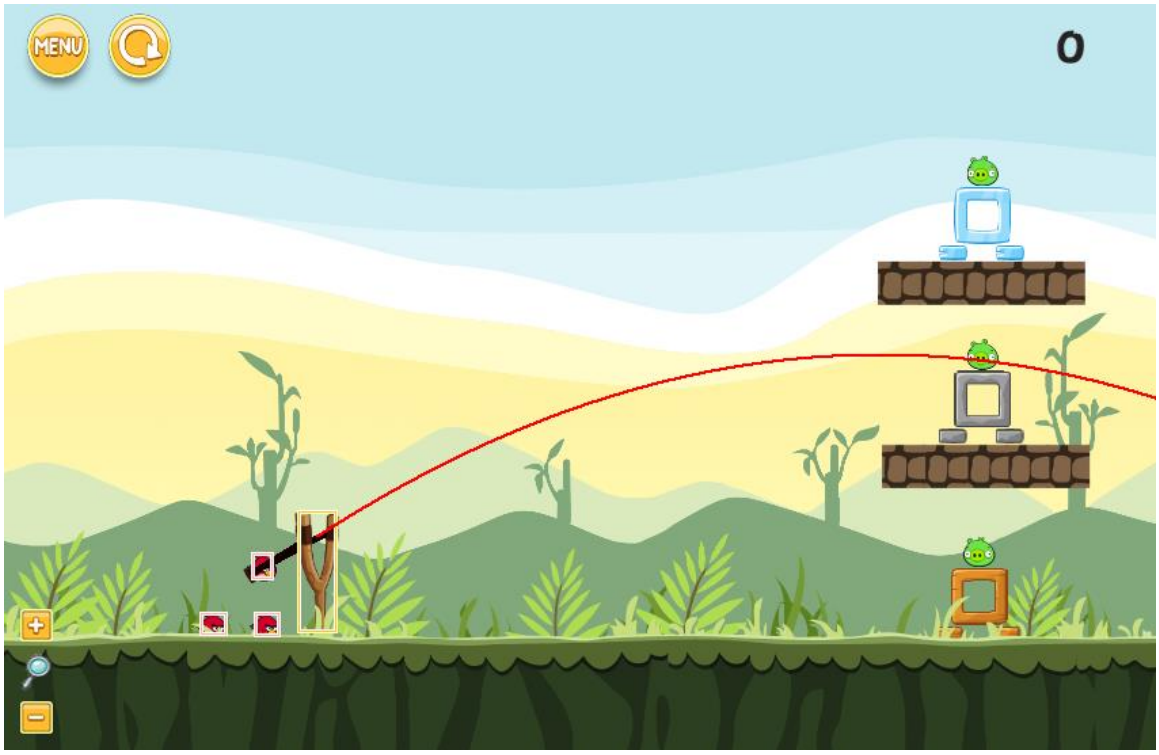

Figure 13: Direct approach option for naïve agent.

DATALAB AGENT

The DataLab agent [13] was created by team DataLab Birds from the Czech Technical University in Prague. It was the AIBirds 2014 champion, scoring 406340 points and solving 6 out of the 8 levels in the final round. The idea behind the agent is that each move is decided based on following one of four strategies:

- Destroy as many pigs as possible strategy: discovers the trajectory that has as many pigs in it as possible. This is the default strategy in the DataLab algorithm.

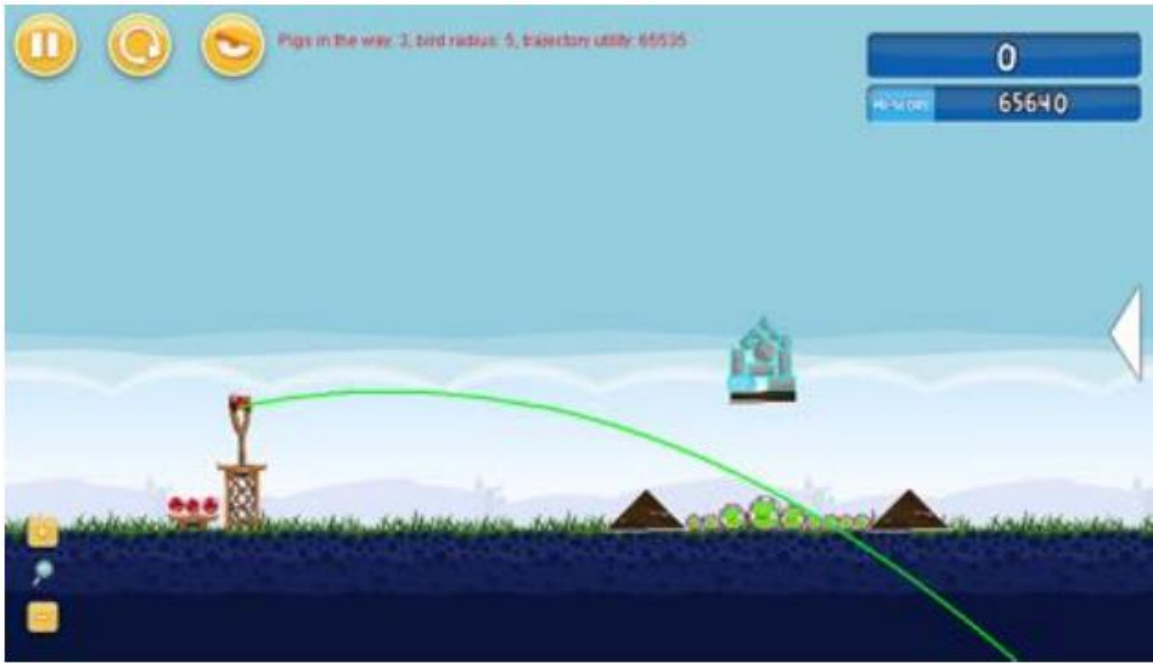

Figure 14: Example path that intersects 4 pigs in a row [13].

- Round blocks strategy: attempts to either coerce a round-shaped object to roll down a hill or other structure and hit pigs, or release a slew of round objects from a shelter that is holding them back. This strategy exploits the kinetic energy of round objects as they roll down a slope or incline. This strategy will

not be used with these procedurally-generated levels (round objects are not supported in my level generator).

- Dynamite strategy: tries to aim at a TNT, but only if there is a pig nearby. The utility of this strategy also goes up if there are many stone objects and other TNT's within the TNT's shockwave range. This strategy will not be used with these procedurally generated levels (TNT's are not supported by the AIBirds research clone).
- Building strategy: finds a connected block structure that houses pigs. This strategy only targets the blocks on the structure that will make it more favorable to target the pigs, since sometimes a misplaced bird can change a structure into a format that is harder to access the pigs inside it. A distinction is also made in this strategy between different types of buildings (eg. a pyramid, rectangle, skyscraper, etc.).

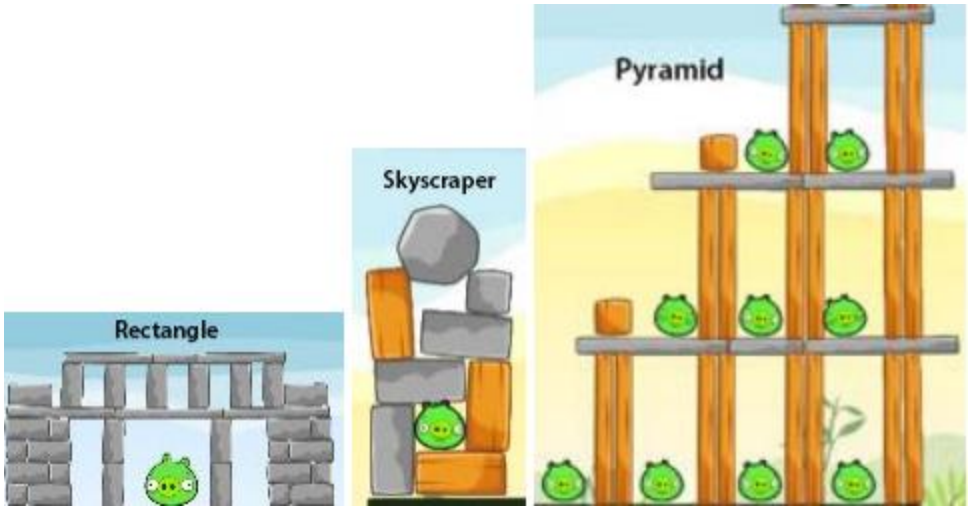

Figure 15: Buildings supported by the DataLab agent [13].

Chapter 6: Level Generation Results

The following table contains the results of the above experiment. The maximum heuristic score in all cases is 10019000, as described in Table 1.

Difficulty Level	Average Heuristic Score	Naïve Agent Wins / Losses	Datalab Agent Wins / Losses
Easy	10016021	100 / 0	100 / 0
Medium	10016251	63 / 37	100 / 0
Hard	10015056	32 / 68	100 / 0

Table 2: Results for level generation by varying difficulty level.

EXAMPLE GENERATED LEVELS

Below are some levels generated by the algorithm for each difficulty level.

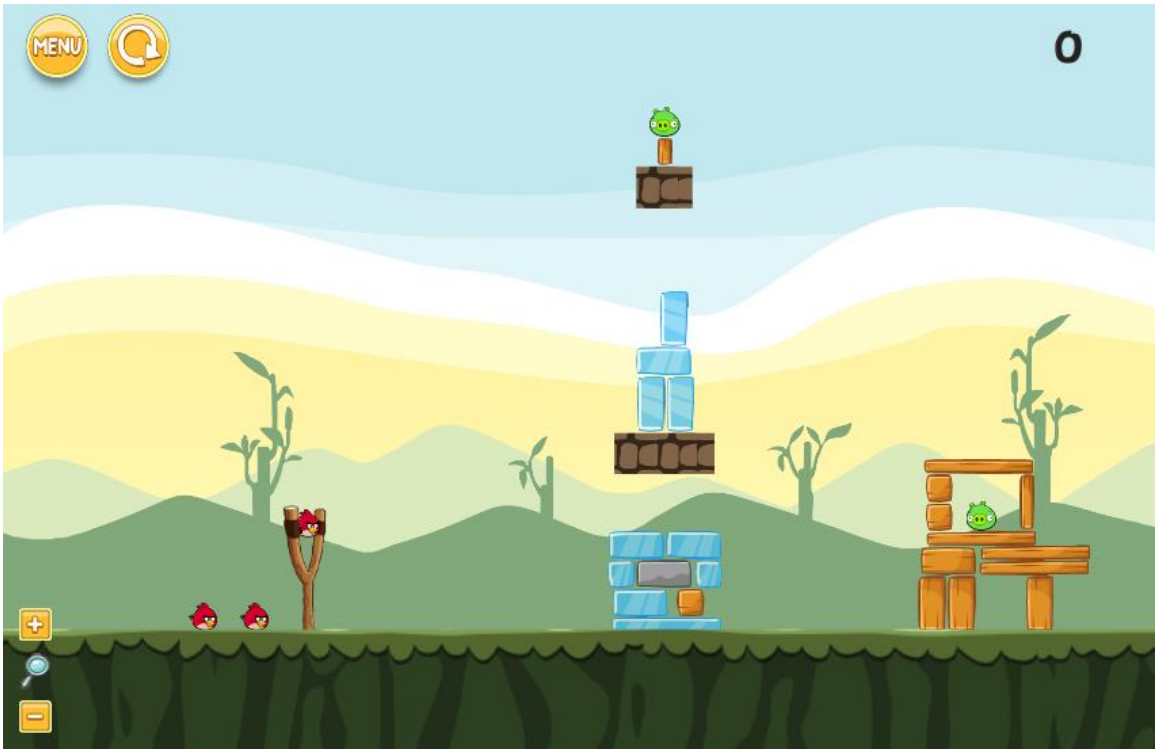

Figure 16: Example generated level with easy difficulty (29 blocks, 2 pigs, 3 birds).

Figure 18: Example generated level with medium difficulty (50 blocks, 5 pigs, 4 birds).

Figure 17: Example generated level with hard difficulty (70 blocks, 6 pigs, 4 birds).

Chapter 7: Results Analysis

PROJECT CHALLENGES / LIMITATIONS

There are some challenges that I ran into while working on the project. Lots of computational power and time is required for this system, and better level generation could be accomplished with more computation. I constrained each round of MCTS to 100ms and total rounds to 1000, but conceivably, the MCTS algorithm could go on for quite some time, as it tries out every action at each node of the tree (3 actions possible). Within the level's space constraints of 96 Unity units in area, you could conceivably fit about 2000 of the smallest blocks inside, and depending on how many input structures you have and how large they are, the number of structures to explore before reaching the top-right of the level is very large. Pigs can be placed in any location on top of ground or in/on a structure where they fit, as well, so they can be assumed to contribute to the search space size at every level of the state space tree. My use case was much more forgiving than having many small structures for each option in the tree. For example, my application derived 122 unique substructures off of existing levels. As an approximation, and based on the size of those 122 structures, the placement algorithm could fit 15 of those structures in the level on average. Thus, the total exploration space could be approximated by the size of a perfect k-ary tree:

$$\#\ total\ states = \frac{k^{h+1} - 1}{k - 1}$$

where:

- k is the number of unique structures
- h is the average height of the tree (changes based on the average size of the structures used)

In this case, the average number of states to explore would be $= \frac{124^{15}-1}{123} \approx 2.04 * 10^{29}$. It is clear that a brute force search would take a long time to search all of those states for a level that best fits the heuristics (and this is just an average case).

That being said, it is possible to create levels using a more naïve approach (eg. with the possibility of winning being the only heuristic, and a greedy search or a brute-force search approach to lay out the levels), but the MCTS algorithm has a very high probability of resulting in a level that will actually be possible to win, found in a reasonable amount of time with a reasonable heuristic score. Players expect to be able to beat levels, and arrive at a greater challenge with each passing level. A naïve approach could try to add more blocks and pigs, but it would most likely be brittle and not be able to scale in creation time with the difficulty level like MCTS can. Theoretically, a server running the MCTS placement algorithm could be chunking away at these levels and handing them out to users online and on-the-fly, which is what players of procedurally-generated levels also expect.

CONSEQUENCES OF MORE COMPUTATION

With more computing time/power, I could increase the limit on the MCTS round to a higher number (say one second), or keep it the same, and have more of the tree be searched in the same amount of time. This would increase the probability of a winnable level, as well as one that fits the secondary heuristics even more, perhaps even one that has the maximum possible score. Throughout my exploration of MCTS, I was not able to achieve the maximum possible score on my computer while under the defined time constraints.

CONSEQUENCES OF MORE INPUT DATA

With more input data, MCTS could form better levels. The distribution of level elements is dependent on samples retrieved from existing levels, so the more varied the input structures, the more choices the MCTS algorithm has when it does its layout. Especially useful are structures that are different shapes (small, tall and skinny, long and skinny, long and fat, etc.). This is the main reason why the MCTS application derives other structures off of existing structures. For example, when emphasizing the shape of a level, it's important to have lots of small (yet interesting and varied) structures that fit under the curve. However, if more input data was available, characterizing the data may be necessary so the MCTS algorithm has an informed decision within the input space (for example, "binning" the structures into their various shapes). Currently, the decision of which structure to explore next is random.

CONCLUSION

In conclusion, procedural content generation using MCTS is a good candidate to produce challenging and fun Angry Birds levels. The MCTS approach uses a combination of exploration and exploitation to meet the design constraints of a level, based on defined heuristics that determine the difficulty of the level. Various AI agents are able to successfully win these generated levels, due to the win/lose heuristic included in the algorithm. You can download my code on GitHub [16], as well as try it out with the AIBirds level generation framework which is also on GitHub [11].

REFERENCES

[1] "'Angry Birds 2' Arrives 6 Years And 3 Billion Downloads After First Game," 16 July 2015. [Online]. Available: http://www.forbes.com/sites/andyrobertson/2015/07/16/angry-birds-2. [Accessed 18 August 2016].

[2] "minecraft.net - Home," Mojang, [Online]. Available: https://minecraft.net. [Accessed 21 September 2016].

[3] "No Man's Sky," Hello Games, [Online]. Available: http://www.no-mans-sky.com. [Accessed 21 September 2016].

[4] O. Good, "It's impossible to visit every planet in No Man's Sky," 19 August 2014. [Online]. Available: http://www.polygon.com/2014/8/19/6045933/its-impossible-to-visit-every-planet-in-no-mans-sky. [Accessed 21 August 2016].

[5] J. Renz, J. Togelius, M. Stephenson, X. Ge and L. Ferreira, "AI Birds.org - Level Generation Competition," [Online]. Available: http://aibirds.org/other-events/level-generation-competition.html. [Accessed 24 August 2016].

[6] G. Chaslot, S. Bakkes, I. Szita and P. Spronck, "Monte-Carlo Tree Search: A New Framework for Game AI," in *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE'08, 216–217*, Stanford University, California, 2008.

[7] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proceedings of the 17th European Conference on Machine Learning, ECML'06, 282-293*, Berlin, Heidelberg: Springer-Verlag, 2006.

[8] A. Summerville, S. Philip and M. Mateas, "MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation," University of California, Santa Cruz, 2015.

[9] L. Ferreira and C. Toledo, "A Search-based Approach for Generating Angry Birds Levels," in *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games*, 2014.

[10] "Unity - Game Engine," [Online]. Available: https://unity3d.com. [Accessed 9 September 2016].

[11] L. Ferreira, "GitHub - lucasnfe/ScienceBirds at NewArt," [Online]. Available: https://github.com/lucasnfe/ScienceBirds/tree/NewArt. [Accessed 24 August 2016].

[12] X. Ge, S. Gould, J. Renz, S. Abeyasinghe, J. Keys, A. Wang and P. Zhang, "Angry Birds Game Playing Software Version 1.32," aibirds.org, 2014.

[13] T. Borovička, R. Špetlík and K. Rymeš, "DataLab Birds Angry Birds AI," 2014.

[14] "AI Birds.org - Results," [Online]. Available: https://aibirds.org/past-competitions/2014-competition/results.html. [Accessed 23 September 2016].

[15] "AI Birds.org - Results," [Online]. Available: https://aibirds.org/past-competitions/2015-competition/results.html. [Accessed 23 September 2016].

[16] M. Graves, "GitHub - LinkOfHyrule/AngryBirdsMCTSLevelGenerator," [Online]. Available: https://github.com/LinkOfHyrule/AngryBirdsMCTSLevelGenerator. [Accessed 3 October 2016].