A KINGDOM KIDNAPPED
(Capstone Design)

Hanae El Mansouri
Supervised by Dr. Tajje-eddine Rachidi

April 2019
A KINGDOM KIDNAPPED

Capstone Report

Student Statement:
I, Hanae El Mansouri, do affirm that I have applied ethics to the design process and in the selection of the final proposed design. And that, I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

Hanae El Mansouri

Approved by the Supervisor

Dr. T. Rachidi
ACKNOWLEDGEMENTS

I would first like to thank my Capstone supervisor Dr. Tajje-eddine Rachidi for agreeing to supervise my project. Without the guidance that he provided me, I doubt I would have been able to succeed in making this game playable or even have thought about trying to integrate machine learning in it.

I would like to thank some of the professors who taught me, particularly Dr. Kevin Smith, Dr. Violetta Cavalli-Sforza, and Dr. Bouchaib Falah for allowing me to explore my passions within and around the field of computer science. Without them, I would not have found my passion for game development and machine learning.

I would like to thank my family and friends for encouraging me to follow my dreams. Without them, I would have neither the courage nor the drive to develop a game for my capstone project in such a difficult semester for me.

I would like to thank all the friends who tested my game and gave me such excellent feedback. Without them, my game would be a broken, buggy mess.

Finally, though they may never see this report, I would like to thank the kind users on OpenGameArt.org and itch.io for posting their assets online and allowing me to use them to complete my game. The assets will be cited appropriately at the end of this report.
CONTENTS

INTRODUCTION ................................................................................................................. 1

STEEPLE ANALYSIS ............................................................................................................. 1
  Societal................................................................................................................................. 1
  Technological...................................................................................................................... 1
  Economic............................................................................................................................ 2
  Environmental.................................................................................................................... 2
  Political ............................................................................................................................... 2
  Legal.................................................................................................................................. 2
  Ethical................................................................................................................................. 2

REQUIREMENT SPECIFICATIONS ....................................................................................... 2
  Functional Requirements .................................................................................................. 2
  Non-Functional Requirements .......................................................................................... 3
  Product Requirements ...................................................................................................... 3
    Usability Requirements .................................................................................................... 3
    Efficiency Requirements ................................................................................................ 3
    Security Requirements .................................................................................................... 3
  Organizational Requirements ............................................................................................ 4
    Environmental Requirements ........................................................................................ 4
    Development Requirements ............................................................................................ 4
  External Requirements ...................................................................................................... 4
    Regulatory Requirements ............................................................................................... 4
    Ethical Requirements ..................................................................................................... 4
    Legislative Requirements ............................................................................................... 4

FEASIBILITY STUDY AND ANALYSIS ............................................................................... 4
  Feasibility Study ............................................................................................................... 4
  Schedule ............................................................................................................................ 4
  Analysis .............................................................................................................................. 5
  Software Process .............................................................................................................. 6

DESIGN & ASSET-GATHERING ......................................................................................... 7
  Diagrams ............................................................................................................................ 7
    “Dummy” NPCs .............................................................................................................. 7
    “Slimers” ......................................................................................................................... 7
“Bonemen” 

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traps (not yet implemented)</td>
<td>7</td>
</tr>
<tr>
<td>Finite State Machines</td>
<td>7</td>
</tr>
<tr>
<td>Main Character</td>
<td>8</td>
</tr>
<tr>
<td>Finite State Machine</td>
<td>8</td>
</tr>
<tr>
<td>Collectable Objects</td>
<td>9</td>
</tr>
<tr>
<td>Finite State Machines</td>
<td>9</td>
</tr>
<tr>
<td>Main Character, Collectable Objects, and Enemies</td>
<td>10</td>
</tr>
<tr>
<td>Class Inheritance Diagram</td>
<td>10</td>
</tr>
<tr>
<td>Asset-Gathering</td>
<td>10</td>
</tr>
<tr>
<td>Ground, Detail, and Background Tilesets</td>
<td>11</td>
</tr>
<tr>
<td>Ground and Detail</td>
<td>11</td>
</tr>
<tr>
<td>Backgrounds</td>
<td>12</td>
</tr>
<tr>
<td>Main Character Tilesheets</td>
<td>13</td>
</tr>
<tr>
<td>Enemy Tilesheets</td>
<td>13</td>
</tr>
<tr>
<td>Collectables Tilesheets</td>
<td>14</td>
</tr>
<tr>
<td>Portals Tilesheets</td>
<td>14</td>
</tr>
<tr>
<td>Font</td>
<td>14</td>
</tr>
<tr>
<td>Level Design</td>
<td>15</td>
</tr>
<tr>
<td>IMPLEMENTATION DETAILS</td>
<td>15</td>
</tr>
<tr>
<td>Technological Enablers</td>
<td>15</td>
</tr>
<tr>
<td>Implementation</td>
<td>15</td>
</tr>
<tr>
<td>Ready-Made Components and C#</td>
<td>15</td>
</tr>
<tr>
<td>UI Elements</td>
<td>15</td>
</tr>
<tr>
<td>Main Camera</td>
<td>17</td>
</tr>
<tr>
<td>Tilemap</td>
<td>18</td>
</tr>
<tr>
<td>Main Character</td>
<td>19</td>
</tr>
<tr>
<td>Enemies</td>
<td>21</td>
</tr>
<tr>
<td>Collectables</td>
<td>23</td>
</tr>
<tr>
<td>Other</td>
<td>23</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>24</td>
</tr>
<tr>
<td>Choosing a Form of Machine Learning</td>
<td>24</td>
</tr>
<tr>
<td>Kinds of Brains</td>
<td>25</td>
</tr>
<tr>
<td>Actual Implementation</td>
<td>26</td>
</tr>
<tr>
<td>REQUIREMENT VALIDATION</td>
<td>33</td>
</tr>
</tbody>
</table>
Requirement Checklist ......................................................................................................................................................... 33
TESTING ........................................................................................................................................................................... 35
CONCLUSIONS .................................................................................................................................................................. 35
Future Perspectives ............................................................................................................................................................ 36
REFERENCES ..................................................................................................................................................................... 37
APPENDIX A: LEVEL DESIGNS ......................................................................................................................................... 41
**Table of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1:</td>
<td>Proposed Schedule</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2:</td>
<td>Followed Schedule</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3:</td>
<td>Software Process Diagram</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4:</td>
<td>Finite State Machine for a Slimer Enemy</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5:</td>
<td>Finite State Machine for a Boneman Enemy</td>
<td>7</td>
</tr>
<tr>
<td>Figure 6:</td>
<td>Finite State Machine for Traps</td>
<td>8</td>
</tr>
<tr>
<td>Figure 7:</td>
<td>Old Finite State Machine for the Main Character</td>
<td>8</td>
</tr>
<tr>
<td>Figure 8:</td>
<td>New Finite State Machine for the Main Character</td>
<td>9</td>
</tr>
<tr>
<td>Figure 9:</td>
<td>Finite State Machine for Collectable Items</td>
<td>9</td>
</tr>
<tr>
<td>Figure 10:</td>
<td>Class Inheritance Diagram for the Main Character, Collectables, and Enemies</td>
<td>10</td>
</tr>
<tr>
<td>Figure 11:</td>
<td>Fantasy Tileset by OddPotatoGift [11]</td>
<td>11</td>
</tr>
<tr>
<td>Figure 12:</td>
<td>Extra Set of Fantasy Tiles by OddPotatoGift [1]</td>
<td>11</td>
</tr>
<tr>
<td>Figure 13:</td>
<td>Forest Background (used for Main Menu) by edermunizz [14]</td>
<td>12</td>
</tr>
<tr>
<td>Figure 14:</td>
<td>Starry Sky Background by OddPotatoGift [1]</td>
<td>12</td>
</tr>
<tr>
<td>Figure 15:</td>
<td>Tileset Used as Background for a Level by ansimuz [9]</td>
<td>12</td>
</tr>
<tr>
<td>Figure 16:</td>
<td>Main Character Tilesheet by rvros [3]</td>
<td>13</td>
</tr>
<tr>
<td>Figure 17:</td>
<td>Slimer Tilesheet by rvros [4]</td>
<td>13</td>
</tr>
<tr>
<td>Figure 18:</td>
<td>Boneman Tilesheet by Jesse Munguia [36]</td>
<td>13</td>
</tr>
<tr>
<td>Figure 19:</td>
<td>Skull Tilesheet by unTied Games [12]</td>
<td>14</td>
</tr>
<tr>
<td>Figure 20:</td>
<td>Collectables by La Red Games [15] and OddPotatoGift [1]</td>
<td>14</td>
</tr>
<tr>
<td>Figure 21:</td>
<td>Portal Tilesheet by Elthen’s Pixel Art Shop [2]</td>
<td>14</td>
</tr>
<tr>
<td>Figure 22:</td>
<td>Font Used by Astramat [34]</td>
<td>14</td>
</tr>
<tr>
<td>Figure 23:</td>
<td>Unity Logo</td>
<td>15</td>
</tr>
<tr>
<td>Figure 24:</td>
<td>Logos for Python and Anaconda</td>
<td>15</td>
</tr>
<tr>
<td>Figure 25:</td>
<td>TensorFlow Logo</td>
<td>15</td>
</tr>
<tr>
<td>Figure 26:</td>
<td>Main Menu Canvas When Editing and In-Game</td>
<td>16</td>
</tr>
<tr>
<td>Figure 27:</td>
<td>HUDCanvas When Editing and In-Game</td>
<td>16</td>
</tr>
<tr>
<td>Figure 28:</td>
<td>PauseCanvas When Editing and In-Game</td>
<td>17</td>
</tr>
<tr>
<td>Figure 29:</td>
<td>DeadCanvas When Editing and In-Game</td>
<td>17</td>
</tr>
<tr>
<td>Figure 30:</td>
<td>Camerafollow Script In-Engine</td>
<td>17</td>
</tr>
<tr>
<td>Figure 31:</td>
<td>Camerafollow Script in Visual Studio 2017</td>
<td>18</td>
</tr>
<tr>
<td>Figure 32:</td>
<td>Example of a List of Tilemaps In-Engine</td>
<td>18</td>
</tr>
<tr>
<td>Figure 33:</td>
<td>Screenshot with Grid Layout In-Engine</td>
<td>18</td>
</tr>
<tr>
<td>Figure 34:</td>
<td>List of Components Used for Ground Tilemap</td>
<td>19</td>
</tr>
<tr>
<td>Figure 35:</td>
<td>List of Components in the King Object</td>
<td>20</td>
</tr>
<tr>
<td>Figure 36:</td>
<td>PlayerBehavior Script as a Component In-Engine</td>
<td>20</td>
</tr>
<tr>
<td>Figure 37:</td>
<td>Animator Controller for the Main Character (with the Parameter in the Red Box)</td>
<td>20</td>
</tr>
<tr>
<td>Figure 38:</td>
<td>Collection Script In-Engine</td>
<td>21</td>
</tr>
<tr>
<td>Figure 39:</td>
<td>EnemyController Script In-Engine</td>
<td>21</td>
</tr>
<tr>
<td>Figure 40:</td>
<td>LevelPortal Script In-Engine</td>
<td>21</td>
</tr>
</tbody>
</table>
Figure 41: Editing Slimer In-Engine ................................................................. 22
Figure 42: Crucial Slimer Components .......................................................... 22
Figure 43: Editing the Boneman Object In-Engine (Green Lines Are All 19 Attack
Polygon Collider 2Ds) ................................................................................. 22
Figure 44: BoneMovement Script With All the Colliders In-Engine ............... 22
Figure 45: SkullOtherBehavior Script In-Engine ............................................ 23
Figure 46: Editing Skull Object In-Engine ....................................................... 23
Figure 47: Table of Collectables and Their Values ......................................... 23
Figure 48: Transport Script In-Engine ............................................................ 24
Figure 49: Editing Portal and House Doors In-Engine ..................................... 24
Figure 50: BedHeal Script In-Engine ............................................................... 24
Figure 51: Editing Bed Object In-Engine ........................................................ 24
Figure 52: Icons of Heuristic, Learning, and Player Brains .......................... 25
Figure 53: Skull Agent RigidBody 2D's Gravity Scale Set to 0 ....................... 26
Figure 55: SkullAcademy Script In-Engine ..................................................... 26
Figure 54: SkullAcademy Script in Visual Studio 2017 ............................... 26
Figure 56: SkullAgent Script in Visual Studio 2017 .................................... 27
Figure 57: SkullAgent Script In-Engine .......................................................... 28
Figure 58: Setting Parameters of the Skull's Learning Brain ......................... 28
Figure 59: Navigate to Cloned Repository on PC and Activating Virtual Environment ....................................................... 29
Figure 60: Parameters in the trainer_config.yaml File ................................. 29
Figure 61: Training Session Run in Command Prompt (part 1) ...................... 30
Figure 62: Training Session Run in Command Prompt (part 2) ..................... 31
Figure 63: Training Session Run in Command Prompt (part 3) ..................... 31
Figure 64: Training Session Run in Command Prompt (part 4) ..................... 31
Figure 65: TensorBoard Graphs Accessed After Several Training Sessions with
Emphasis on the First Graph ......................................................................... 32
Figure 66: SkullLearningBrain with the Neural Network Training Model ....... 32
Figure 67: Functional Requirements Fulfillment Status ................................ 33
Figure 68: Non-Functional Requirements Fulfillment Status ....................... 34
Figure 69: Level 1 Design ............................................................................. 41
Figure 70: Level 2 Design ............................................................................. 42
Figure 71: Level 3 (page 1) Design ................................................................ 42
Figure 72: Level 3 (page 2) Design ............................................................... 42
ABSTRACT

This report serves as the documentation of the research done and process used to go about designing and developing my capstone project, *A Kingdom Kidnapped*. This project is a two-dimensional game that runs on both Windows and Linux systems, developed using the Unity Engine.

For the first part of the implementation, pre-made components and C# were used to implement everything. The pre-made components were used to apply physical attributes and behaviors to the objects (such as gravity, mass, friction, etc.). The C# scripts were created, using functions from libraries provided by Unity, to control everything from the main character’s movements, to the autosaving between levels, and even for flipping the animations for both main character and enemies when they move in the opposite direction. These components and scripts were combined to define objects, platforms, camera movement, and an autosave system, that all functioned as expected.

For the second part, *ML-Agents* [30], a package developed for machine learning in the Unity Engine that is still in beta, was used to train the machine learning-taught enemies. Python and Anaconda were used to run prewritten Python code that were provided in the *ML-Agents* package as well as the other packages that were installed along with it. At this stage, no Python knowledge was needed to implement the machine learning. Though it is possible to used other forms of machine learning with *ML-Agents*, such as imitation/curriculum learning, reinforcement learning was used to program how the enemies were to develop their behaviors (also called *policies* [26]) during the training session. After many trials and errors, the trainings were successful, and the enemies learned to both navigate around their parts of the level while also approaching the main character (their target). However, more trainings will be required in the future to improve their accuracy.
INTRODUCTION

Video games are a significant form of interactive storytelling that can influence anyone who plays them. It is a technical art form that requires not only proficiency in software engineering, but also in art, music, and sometimes language. Designing a game means crafting a story or playable experience that players will enjoy.

This game, like many that have been and will be created, does not serve a particular purpose. Nor is it designed to solve a particular problem. It is a game created for the purpose of providing entertainment, much like an adventure movie or fictional book might. The reason that I decided to create a game for my capstone is simply that it is a passion of mine that I discovered during my time as an undergraduate student at Al Akhawayn University.

The game has a simple premise: a king of a distant, fictional land wakes up one morning to find that everyone has disappeared. His family, his guards, the citizens, even the citizens’ pets. He exits his castle to find a note that reveals that a soldier he had imprisoned many years ago held a grudge against the king for all those years and learned dark magic to make the king pay for what he had done. He finally escaped and cast a spell on the entire kingdom so that the king would have to come rescue them himself. The “user” plays as the king and must collect treasure and either bribe the dark magician’s minions or fight them to get rescue his people because what is a king without his subjects?

STEPPLE ANALYSIS

Societal
This game could be considered a fun way to subconsciously instill the idea that people should always try to do the right thing when faced with hard times. That the needs of the many outweigh the needs of the self.

Technological
This game will be available for Windows and Linux, at the very least. It may even make it so that Linux is a more used operating system. This project will also be using a machine learning tool that is still in Beta. It will open the doors to a whole new way of developing games.
Economic
This game will be free to download off itch.io when fully completed and therefore, should not affect anything or anyone economically.

Environmental
The game should not use up too much of the computers’ resources due to the simple 2D art so should, therefore, not use too much power.

Political
Though the story is one based on the concept of a monarchy, there should not be any political issues. This game does not reflect on any real government or entity in any place.

Legal
Everything obtained to develop this game was obtained legally and with the appropriate permissions from the creators of said assets.

Ethical
All due credit will be given, and I will not take credit for work that I did not complete myself.

REQUIREMENT SPECIFICATIONS

Functional Requirements
1. Functional Requirement

   User Requirement

   The avatar will be controllable with the keyboard and a controller.

   System Requirement

   The user will be able to use the keyboard/controller to control where the avatar walks, runs, and jumps to and how the character fights.

2. Functional Requirement

   User Requirement
The user should be able to use the keyboard/controller for things other than controlling the avatar.

System Requirement

They should be able to use the keyboard to make the avatar gather collectables, bring up a menu, look through the gathered collectables, and to select options-buttons.

3. Functional Requirement

User Requirement

The user shall be able to save their progress between levels (provided that the user did not restart the game from the beginning).

System Requirement

The game should automatically save the user’s progress at certain checkpoints and after levels.

Non-Functional Requirements

Product Requirements

Usability Requirements

- The game shall have a start menu with Start, Credits, and Exit.
- The game may have a help section explaining how to play.
- The game should be intuitive and interactive.

Efficiency Requirements

Performance Requirements

- The game should boot in less than 10 seconds.
- The game should perform at a framerate of at least 30 fps.

Space Requirements

- The game shall not take more than 1 GB in space.
- The game should limit updates space to 2 GB.

Security Requirements

- The game shall restrict player form using cheating software (Cheat Engine).
The game should be protected from all attacks and especially patches and harmful programs.

Organizational Requirements

Environmental Requirements
- The game should not be used to damage the environment by any way.

Development Requirements
- The game shall be implemented using a version of Unity 2018.
- The game should respect C# and Python restrictions.
- This game be built for Windows and Linux.

External Requirements

Regulatory Requirements
- The game shall allow the interaction of the user within the boundaries of the game rules and game physics.

Ethical Requirements
- The game should be available for free and to the open public.
- All due credit will be given in the Credits of the game and in this report.

Legislative Requirements

Privacy Requirements
- Any personal information of the player that may be used shall be encrypted and fully secured.

Safety Requirements
- The game will not in any way harm anyone who plays it or installs it.

FEASIBILITY STUDY AND ANALYSIS

Feasibility Study

Schedule

Below was the proposed schedule:

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Development Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>Project Selection</td>
</tr>
</tbody>
</table>
Realistically, I could not follow that schedule so instead, here the schedule I ended up following:

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Development Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>Project Selection</td>
</tr>
<tr>
<td>Week 2 to half of week 3</td>
<td>Analysis &amp; Feasibility</td>
</tr>
<tr>
<td>Halfway through week 3 to end of week 3</td>
<td>Design</td>
</tr>
<tr>
<td>Week 4</td>
<td>Design &amp; Asset Gathering</td>
</tr>
<tr>
<td>Week 5</td>
<td>Implementation</td>
</tr>
<tr>
<td>Week 6</td>
<td>Asset Gathering &amp; Implementation</td>
</tr>
<tr>
<td>Week 7</td>
<td>Implementation &amp; ML Research</td>
</tr>
<tr>
<td>Week 8</td>
<td>Implementation ( &amp; Asset Gathering)</td>
</tr>
<tr>
<td>Week 9</td>
<td>ML Research</td>
</tr>
<tr>
<td>Week 10</td>
<td>Implementation &amp; ML Component Installation ( &amp; Asset Gathering)</td>
</tr>
<tr>
<td>Week 11</td>
<td>Implementation</td>
</tr>
<tr>
<td>Week 12</td>
<td>ML Implementation</td>
</tr>
<tr>
<td>Week 13</td>
<td>Testing</td>
</tr>
<tr>
<td>Weeks 14 to 15</td>
<td>Report, Defense, &amp; Presentation</td>
</tr>
</tbody>
</table>

The goal was to split the design and implementation into two parts such that after I finish a majority of the main game design implementation, I would start working on the machine learning portion of the project to ensure that if the machine learning did not work, I would still be able to present a working product.

Analysis

I researched the genre that I wanted to base my game on along with the games that fall under that genre that I have played before. I looked at what the critics and the players wrote in their reviews so I could see what I should keep in mind or avoid. I was inspired by the games Mario Bros. and Cave Story.
A *Kingdom Kidnapped* mainly follows the style of *Mario Bros.* such that it introduces the concept of “permadeath” or “permanent death”. The term “permadeath” here means that if the health of the character reaches zero, the player must either restart the game from the beginning or from a checkpoint (the latter not having been implemented yet due to the small number of levels). Also similar to *Mario Bros.*, *A Kingdom Kidnapped* allows only for one-way progression through the levels. That is to say, once the player completes a level by traversing through a door to the next level, they cannot go back.

Next, I researched AI in video game development and came up with a list of topics I could include in my project. I considered creating randomly generated tile layouts and creating enemies that would use the A* algorithm to find the shortest path to the character. However, the more interesting AI topic was proposed to me by my capstone supervisor, Dr. Rachidi. He proposed that I use Machine Learning to teach an enemy how to fight. I researched the many ways one could implement machine learning and found that the best way would be to use Reinforcement Learning (which I will be giving more details later in this report).

Software Process

As there are two distinct parts of this project, I made the decision to form a variation of the Agile development model that would best fit my project. The requirement specifications and the analysis and feasibility were performed once but the design, implementation, training, and testing portions were repeated. Then, when satisfied by those parts, the evolution phase would be carried out.

*Figure 3: Software Process Diagram*
DESIGN & ASSET-GATHERING

Diagrams
“Dummy” NPCs
“Slimers”

Finite State Machines

Finite State Machines

“Bonemen”

Finite State Machines

Traps (not yet implemented)

Finite State Machines
Figure 6: Finite State Machine for Traps

Main Character

Finite State Machine

Old FSM

Figure 7: Old Finite State Machine for the Main Character
New FSM

Collectable Objects

Finite State Machines

Figure 8: New Finite State Machine for the Main Character

Figure 9: Finite State Machine for Collectable Items
Main Character, Collectable Objects, and Enemies

Class Inheritance Diagram

![Class Inheritance Diagram](image)

**Figure 10: Class Inheritance Diagram for the Main Character, Collectables, and Enemies**

Asset-Gathering

The art, music, and sound effects, which were all free-to-use, were gathered off the internet (through the websites OpenGameArt.org and itch.io) and utilized with the express permission of the artists. The assets were gathered on an as-needed basis rather than all in one shot so as to avoid wasting time. Here are the assets I downloaded and used in my project:
Ground, Detail, and Background Tilesets

Ground and Detail

Figure 11: Fantasy Tileset by OddPotatoGift [11]

Figure 12: Extra Set of Fantasy Tiles by OddPotatoGift [1]
Backgrounds

Figure 13: Forest Background (used for Main Menu) by edermunizz [14]

Figure 14: Starry Sky Background by OddPotatoGift [1]

Figure 15: Tileset Used as Background for a Level by ansimuz [9]
Main Character Tilesheets

Figure 16: Main Character Tilesheet by rvros [3]

Enemy Tilesheets

Figure 17: Slimer Tilesheet by rvros [4]

Figure 18: Boneman Tilesheet by Jesse Munguia [36]
Collectables Tilesheets

![Image of collectables tilesheet]

Figure 20: Collectables by La Red Games [15] and OddPotatoGift [1]

Portals Tilesheets

![Image of portals tilesheet]

Figure 21: Portal Tilesheet by Elthen’s Pixel Art Shop [2]

Font

PIXELADE:

```
abcdefgijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890,:;'?"(?)+-*/=
```

Figure 22: Font Used by Astramat [34]
Level Design

Before creating a level in-engine, I would first design the levels by hand. The full-size designs will be provided in the appendix of this report.

IMPLEMENTATION DETAILS

Technological Enablers

The game engine that was used was *Unity v2018.3.6f1* and the scripting was done using the scripting language C# using *Visual Studio 2017*.

![Unity Logo](Figure 23: Unity Logo)

For the machine learning implementation, I first had to make sure that I had *Python v3.6.0* and *Anaconda v4.3.1* before attempting to install the other components [18], [19].

![Logos for Python and Anaconda](Figure 24: Logos for Python and Anaconda)

After making sure I had the right versions of Python and Anaconda, I cloned the ML-Agents repository from GitHub through command prompt and installed both the *ML-Agents* and *TensorFlow* packages using *pip* (through Python) [18], [19], [30]. To later perform the training, a virtual environment must be created through Anaconda [19].

Implementation

I split the implementation portion into two distinct parts: the normal C# portion and the machine learning portion. A majority of the time was spent on implementing the former.

Ready-Made Components and C#

*UI Elements*

When creating the UI elements, whether that would be for the Main Menu before starting the game or the Pause Menu that pops up when the player presses the Escape key, the first thing that must be created is a GameObject called a canvas. A canvas is used as a container for text, buttons, and graphics that will appear on top of the screen as UI. For the canvas to operate correctly (for the buttons to work, for example),

![TensorFlow Logo](Figure 25: TensorFlow Logo)
another object called an EventSystem is created the first time a canvas is created in a scene.

The Main Menu, of course, consists of a background and a canvas that contains the title of the game and three buttons: Play, Credits, and Exit.

![Figure 26: Main Menu Canvas When Editing and In-Game](image)

Each level contains three canvases. The one that is always on the screen was called the HUDCanvas, where HUD stands for Heads Up Display. This HUD canvas is where the health bar and the total amount of money collected is displayed and is never disabled.

![Figure 27: HUDCanvas When Editing and In-Game](image)

The second canvas was called the PauseCanvas. This is set to be disabled when the level starts by default but will be enabled when the player presses the Escape key on their keyboard and is disabled when it is pressed again. While the PauseCanvas is enabled, a small script runs that changes the time scale to 0 so that no time passes in
the game (which stops all movement). When enabled again, the time scale is returned to 1.

![Figure 28: PauseCanvas When Editing and In-Game](image1)

The third canvas was called the DeadCanvas. Like the PauseCanvas, this canvas is set to disabled when the scene starts running. It only becomes enabled when the character runs out of health. However, unlike the PauseCanvas, the activation of this canvas does not come with a change of the time scale and it cannot be disabled again unless the character restarts the game.

![Figure 29: DeadCanvas When Editing and In-Game](image2)

**Main Camera**

By default, the main camera is already initialized when a new scene is created. However, to have the camera follow the character around as they move, a script

![Figure 30: Camerafollow Script In-Engine](image3)
must be written and attached to it. When creating the basic script for the main camera, there are two things that must be accounted for: the target (or the object that the camera is following) and the camera movement speed.

```
using UnityEngine;

public class CameraFollow : MonoBehaviour {
    public float FollowSpeed = 2f;
    public Transform Target;

    private void Update()
    {
        Vector3 newPosition = Target.position;
        newPosition.z -= FollowSpeed * Time.deltaTime;
        transform.position = Vector3.Lerp(transform.position, newPosition, FollowSpeed * Time.deltaTime);
    }
}
```

*Figure 31: CameraFollow Script in Visual Studio 2017*

**Tilemap**

The tilemap is used to allow for easier creation of two-dimensional levels. When a tilemap is added to a scene, a grid is added on-top of the scene. I would create a different tilemap in the tilemap parent object for different layers within it. The most important tilemap is the *ground* tilemap which is the one that contains the tiles for the platforms (whether that be the ground or the walls). All the other tilemaps are there for non-interactable details within the levels.

*Figure 32: Screenshot with Grid Layout In-Engine*

*Figure 33: Example of a List of Tilemaps In-Engine*
The components that make the ground tilemap different from the others are the *Rigidbody2D*, *Composite Collider 2D*, and *Tilemap Collider 2D* components. The Rigidbody2D component makes sure that certain physical properties are applied to the GameObject that it is attached to. This usually includes the gravity scale and mass but as this tilemap should not be moving, the body type is set to *static* as it indeed is a static object. The Composite Collider 2D component here is used in conjunction with the Tilemap Collider 2D component. These components work together to make sure that the object is a solid object that can be walked on. A vital Boolean variable in Tilemap Collider 2D is the *Used By Composite* variable. When true, all the individual colliders that the adjacent tiles have are combined to form one collider. If set to false, the player would occasionally be able to force the character through the cracks and get stuck either between tiles or even inside the walls.

**Main Character**

The main character has many different components, such as a Rigidbody 2D, an Animator (a component that contains a finite machine that controls the various animations that the character can go through), and a Capsule Collider 2D (like the Tilemap Collider 2D but in the shape of a capsule or ellipse). In this section, however, I will mostly be describing the scripts I used that are attached to this GameObject.
The first script is called `PlayerBehavior`. This script contains the code that allows the player to move when the health is full and disables player movement if the health bar has reached 0. It also constantly checks if the character is touching the ground to make sure that they player cannot keep jumping in the air infinitely. The script sends the information (in float or Boolean form) to the animator controller so that it can decide whether or not the animations should change (if the right conditions are met).
Another script is called *Collection*. It is used to trigger certain actions when the player collects a coin, jewel, or heart. The script will get the set value that the coin or jewel is worth and then update the text located in the HUDCanvas with the updated amount of money that the player has collected. It also sends the preset value of the heart, when that heart is collected, to the script that handles the character health. Then, when all the values have been updated, a sound effect is played right before the collected object self-destructs.

The last script attached to the main character that will be discussed is the *LevelPortal* script. This script runs when triggered by the user when they try to progress through the levels. It holds the information about what level is next and gets the current health and current money variables from the other scripts and saves them in a text file called *CharacterData.txt* in JSON format right before changing levels. When the level changes successfully, the data is loaded from the file into the appropriate places. The way it is set up ensures that if the player restarts from the beginning (level 1), their data is not loaded into the game. When the player progresses to the next level, the save file gets overwritten. This method of saving and loading makes the previously mentioned concept of *permadeath* possible as the current level is not saved, and the saved health and money stats are not reloaded when restarting the game.

*Enemies*

Every enemy object, including the machine learning taught enemy, executes the *EnemyController* script. This script takes two values that I set beforehand, to set the maximum health and how health points are deducted from the player they collide. Its main purpose is to detect collisions, check if it collided with the main
character, and if so, sends the doDamage variable to the character’s PlayerHealth script to deduct it. It also holds commented code for me to develop further later for when the player will be able to attack the enemy objects.

“Slimers”

In addition to the Rigidbody2D, Animator, EnemyController script, and Capsule Collider 2D, there are two other components that these slimer GameObjects have that work together. The first is a script called SlimeMovement. This script, as the name suggests, is the script that dictates how the slimer moves. It is meant to move right until it hits a wall/object or reaches the end of a platform then flip around and repeat it infinitely. It uses a small Box Collider 2D that is placed in front of the sprite (the image of the slimer) as a trigger to prompt it to turn and move in the opposite direction.

“Bonemen”

When creating the bonemen, I decided to make the colliders fit a little more accurately around the enemy by using a Polygon Collider 2D. However, unlike other types of two-dimensional colliders, its shape cannot be changed to adapt to each frame in the animation. This was fine for the idle and walking animations as the frames were pretty similar, however, this became a problem when implementing the attack animation. Thus, I went about creating a distinct collider for...
all nineteen frames and implemented a function that switches between each of them within the BoneMovement script. This script also tracks the main character and makes it move when it is within 0.6 meters and attack when within 0.15 meters while also implementing a function that flips the animation if it needs to face the other way.

Skulls

These enemies are the ones that are taught how to move and attack using machine learning. As the machine learning aspect of this project will be discussed later in this report, the remaining script that is not using machine learning, SkullOtherBehavior, will be discussed here. Along with flipping the sprites when moving in the other direction, this script also checks whether the skull in is contact with the ground or the other enemies and returns true while in contact or false when it is no longer in contact. This Boolean is used to help the training and decision-making in the machine learning portion of it.

Collectables

There are five distinct types of collectables in the game:

<table>
<thead>
<tr>
<th>Types</th>
<th>Worth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silver coins</td>
<td>1</td>
</tr>
<tr>
<td>Gold coins</td>
<td>5</td>
</tr>
<tr>
<td>Emeralds</td>
<td>10</td>
</tr>
<tr>
<td>Rubies</td>
<td>15</td>
</tr>
<tr>
<td>Hearts</td>
<td>25 (in health points)</td>
</tr>
</tbody>
</table>

Each of these collectables have a short script that holds the value of each item to be passed into the aforementioned Collection script.

Other

Doors to Other Levels

These objects are simply images with a Box Collider 2D attached to them, acting as a trigger. This trigger is used by the LevelPortal script. When the trigger is set off and the down arrow is pressed, the LevelPortal script starts the next level.
Portals and House Doors

The portals and house doors are identical in everything but which sprites they use. They both use a script called the Transport script which transports the character to the corresponding door or portal. It also moves the camera so that it also gets transported instantaneously with the main character. Finally, I set an interval of time within which the character would not be able to use the door or portal. While the time still has not passed, the variable canTransport is set to false. After it has passed, canTransport is set to true and the door or portal can be used again.

Bed

In the house in the second level, there is a bed. This bed is not really part of the background and can be interacted with. It is equipped with a Box Collider 2D component which acts as a trigger. This trigger is used in the script BedHeal which is used by this bed object. When triggered and the down key has been pressed, the character is returned to full health.

Machine Learning

Choosing a Form of Machine Learning

There are many kinds of learning that one could use when using machine learning a few being imitation learning, reinforcement learning, and neuroevolution. In the beginning of this project, I researched both imitation and reinforcement learning considering what kind of training I wanted the enemy agent to undergo. Imitation learning is a form of learning in which the agent is told exactly what it is supposed to learn while reinforcement learning is a form of learning that allows for the agent to figure out what behaviors or policies [26] it must learn through a reward system. I found reinforcement learning to be the more interesting way to train the agents.
Reinforcement learning is a form of machine learning in which Agents (the object to be trained), to perform certain tasks or adopt certain behaviors by allowing the Agent to observe its environment on its own and based on these observations, will make a decision based on how it was rewarded. There are two forms of rewards: positive and negative. As would be expected, a positive reward is awarded to the agent when it does something right and a negative reward if it does something wrong (a negative reward can also be considered a punishment). Positive and negative rewards are in the form of floats [24]. It is suggested that positive rewards go from +0.1 to +1.0 and negative rewards should be greater than -0.5 so that unintended learned behaviors will be avoided (for example, if I was teaching the agent to avoid touching the walls and ground and the negative reward was set too low, the agent might avoid going through a tunnel to follow the character because the walls are too close to the agent) [21], [23].

*Kinds of Brains*

To learn behaviors and make decisions, every agent must be equipped with a brain. The ML-Agents toolkit offers three kinds of brains: heuristic, player, and learning brains. The heuristic brain is used when hand-coding the logic used by the Agent, the player brain is used when controls are mapped to the actions performed by the Agents usually when testing the Agent’s code, and the learning brain is used to train a new model or use an existing one [16].

*Figure 52: Icons of Heuristic, Learning, and Player Brains*
Actual Implementation

For this project, I chose to create a single enemy that uses reinforcement learning to navigate and attack the player. This enemy agent was not bound by gravity as all the other enemies and the main character are so that it would be easier for the agent to navigate around the level.

After creating the character and attaching all the basic components and scripts to it, I started on the machine learning. First, I created a game object called Academy which I attached an empty Academy script to. The Academy script serves to take the base class Academy and attaches all the variables and behaviors in the Academy class already provided by the ML-Agents package [24]. The script in the game object is used to list the brains to be trained and to allow for the setting of certain parameters and variables according to how the agent should be trained. For example, I can set the time scale to 1 or higher which multiplies the current time scale by whatever value in inputted. This allows for the time to elapse faster in the game during training which allows for the training session to complete faster as a result. However, as the agent must learn to follow and attack the character, I would need to move the character around so that the agent would not spend all its time in one corner of the level. This means that the time scale cannot be too fast otherwise navigating around the level as the main character would be impossible. There was then the issue that that the training would move too slow if the time scale was too low. I settled on setting it to 10 which was slow enough for me to
move the main character around but fast enough that the training did not take too long.

Next, I created an agent script that takes the base class Agent. The code in this script is used to collect observations about the environment around it and make a decision based on those observations. It also attaches certain rewards to actions that the agent might take. In this Agent script, I set a positive reward of +1.0 to be awarded to the agent when it was less than 0.2 meters away from the character and a negative reward of -0.05 when it touched the ground, walls, or any of the enemies.

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using MLAgents;

public class SkullAgent : Agent{
    Rigidbody2D rBody;
    public Transform target;
    public float speed = 0.5f;
    public bool touchedObject;

    void Start() {
        rBody = GetComponent<Rigidbody2D>();
    }

    private void FixedUpdate() {
        touchedObject = GetComponent<SkullOtherBehavior>().touchObject;
    }

    public override void CollectObservations() {
        AddVectorObs(target.position);
        AddVectorObs(rBody.transform.position);
        AddVectorObs(rBody.velocity.x);
        AddVectorObs(rBody.velocity.y);
    }

    public override void AgentReset() {
        // Debug.log("AGENT RESET");
    }

    public override void AgentAction(float[] vectorAction, string textAction) {
        Vector2 controlSignal = Vector2.zero;
        controlSignal.x = vectorAction[0];
        controlSignal.y = vectorAction[1];
        rBody.AddForce(controlSignal * speed);

        float distanceToTarget = Vector2.Distance(this.transform.position, target.position);
        if (distanceToTarget < 0.2f) {
            SetReward(1.0f);
            Done();
        }
        if (touchedObject == true) {
            SetReward(-0.05f);
        }
    }
}
```

*Figure 56: SkullAgent Script in Visual Studio 2017*
When I finished writing the agent script, I created a learning brain for the agent. I attached this brain to the Agent script after updating the parameters (i.e. vector observation space size, vector action space type, and vector action space size). I then added this learning brain to the Broadcast Hub in the Academy script (in the Academy object) and set the control Boolean to true. When the control Boolean is set to true, this signals to Unity that the agent is set to start training. Right before training, I made sure to go into the player settings, added \textit{ENABLE\_BARRACUDA} (and not \textit{ENABLE\_TENSORFLOW} as the documentation says [23]) in the Scripting Define Symbols, and made sure to set Allow ‘unsafe’ code to true. This ensures that the learning brain will accept a neural network as the model.
To start training, I would open the command prompt, navigate to the cloned ML-Agents folder on my computer and activate the virtual environment that was created beforehand. Making sure that I had Unity open to the level I had the agent in, I passed a Python command in the command prompt to signal to the system that the training process will be starting. The general command with all the arguments I used was: `mlagents-learn config/trainer_config.yaml --run-id=<run-identifier> --train` [6].

- "mlagents-learn" is the Python command that initiates the training.
- "config/trainer_config.yaml" is the configuration file that lists all the extra data to be used to train the agent.

```yaml
default:
  trainer: ppo
  batch_size: 1024
  beta: 5.0e-3
  buffer_size: 10240
  epsilon: 0.2
  gamma: 0.99
  hidden_units: 128
  lambda: 0.95
  learning_rate: 3.0e-4
  max_steps: 5.0e4
  memory_size: 256
  normalize: false
  num_epoch: 3
  num_layers: 2
  time_horizon: 64
  sequence_length: 64
  summary_freq: 1000
  use_recurrent: false
  use_curiosity: false
  curiosity_strength: 0.01
  curiosity_enc_size: 128
```

Figure 60: Parameters in the trainer_config.yaml File
• “--run-id=<run-identifier>” names the folder that will hold the resulting Neural Network model and the summary of the results.
• “--train” is the flag that signals that it will be running in train mode. (“--train” can be replaced with “--load” if you want to go back to another training session and add to it). [6]

After entering this command, I was prompted to run the game from Unity to start the training. After that, it starts training itself in steps. The number of steps in a training session is set to 50,000 by default in trainer configuration file but that can be changed. I personally preferred to keep it set to the default. Next, all I had to do was move the main character around so that the agent could get used to a moving target. It is also helpful to note that, to have multiple instances of the trained enemy agent in different areas of the level, they must all be trained with multiple learning brains and cannot use the same neural network models that resulted from the training of the first agent. In the screenshots I will show, to save space, I cut the training short:

![Figure 61: Training Session Run in Command Prompt (part 1)](imageURL)
Figure 62: Training Session Run in Command Prompt (part 2)

Figure 63: Training Session Run in Command Prompt (part 3)

Figure 64: Training Session Run in Command Prompt (part 4)
After training the agents a few times, I was able to use TensorBoard [38] to compile some graphs that gave visual representations of total reward obtained by the agents after every step as well as which sessions were cut short. TensorBoard provides five graphs but the only one that I was concerned with was the first one.

![Figure 65: TensorBoard Graphs Accessed After Several Training Sessions with Emphasis on the First Graph](image)

Now that I had the neural networks resulting from the training sessions, I could attach them to the brains that they were trained with.

![Figure 66: SkullLearningBrain with the Neural Network Training Model](image)
REQUIREMENT VALIDATION

Before starting the project, I ranked the functional requirements, as numbered in the Requirements Specifications section, and prioritized them in that order, though not every aspect of those requirements could be fulfilled in their entirety. The only functional requirement I could not fully complete was the second one as I was not able to make the buttons in the options selectable by controller.

As for the non-functional requirements, I did not check back on them until I was testing the game. Anything that did not comply with the non-functional requirements list that I compiled were checked and altered to fit to the requirements later. As it turned out, with a few exceptions, most of those requirements had already been satisfied.

Requirement Checklist

This section will contain a list of all the requirements, which had been fulfilled, and which had not.

<table>
<thead>
<tr>
<th>Functional Requirements</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The avatar will be controllable with the keyboard and a controller:</strong></td>
<td>Fulfilled (The player would be able to move around and jump using a controller but run and fight was not implemented)</td>
</tr>
<tr>
<td>The user will be able to use the keyboard/controller to control where the avatar walks, runs, and jumps to and how the character fights.</td>
<td></td>
</tr>
<tr>
<td><strong>The user should be able to use the keyboard/controller for things other than controlling the avatar:</strong></td>
<td>Mostly Fulfilled (The player would not be able to select options with their controller)</td>
</tr>
<tr>
<td>They should be able to use the keyboard/controller to make the avatar gather collectables, bring up a menu, look through the gathered collectables, and to select options/buttons.</td>
<td></td>
</tr>
<tr>
<td><strong>The user shall be able to save their progress between levels (provided that the user did not restart the game from the beginning):</strong></td>
<td>Fulfilled</td>
</tr>
<tr>
<td>The game should automatically save the user’s progress at certain checkpoints and after levels.</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 67: Functional Requirements Fulfillment Status*
<table>
<thead>
<tr>
<th>Non-Functional Requirements</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usability Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game shall have a start menu with Start, Credits, and Exit.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>- The game may have a help section explaining how to play.</td>
<td>Unfulfilled</td>
</tr>
<tr>
<td>- The game should be intuitive and interactive.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Performance Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game should boot in less than 10 seconds.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>- The game should perform at a framerate of at least 30 fps.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Space Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game shall not take more than 1 GB in space.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>- The game should limit updates space to 2 GB.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Security Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game shall restrict player form using cheating software (Cheat Engine).</td>
<td>Unsure</td>
</tr>
<tr>
<td>- The game should be protected from all attacks and especially patches and harmful programs.</td>
<td>Unsure</td>
</tr>
<tr>
<td><strong>Environmental Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game should not be used to damage the environment in any way.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Development Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game shall be implemented using a version of Unity 2018.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>- The game should respect C# and Python restrictions.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>- This game be built for Windows and Linux.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Regulatory Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game shall allow the interaction of the user within the boundaries of the game rules and game physics.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Ethical Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game should be available for free and to the open public.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td>- All due credit will be given in the Credits of the game and in this report.</td>
<td>Fulfilled</td>
</tr>
<tr>
<td><strong>Privacy Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- Any personal information of the player that may be used shall be encrypted and fully secured.</td>
<td>Unsure</td>
</tr>
<tr>
<td>(No personal information is needed to play the game)</td>
<td></td>
</tr>
<tr>
<td><strong>Safety Requirements</strong></td>
<td></td>
</tr>
<tr>
<td>- The game will not in any way harm anyone who plays it or installs it.</td>
<td>Fulfilled</td>
</tr>
</tbody>
</table>

*Figure 68: Non-Functional Requirements Fulfillment Status*
TESTING

Testing was performed in-engine every time a C# script was implemented as well as after every training session. This was not only to catch the bugs in the scripts but to also try to see if the game was not performing as it should despite having written the code correctly. After completing the training sessions and doing all the initial testing myself, I built the game and distributed it to a few of my friends. They gave me excellent feedback and notified me of some of the problems they encountered. There were two issues that everyone agreed on:

1. The Skull agent is not threatening enough of an enemy because it is not particularly good at attacking the character.
2. Sometimes the character just phases through the platform and gets stuck inside.

The first issue can be remedied with more training. The second issue will have to warrant a little more research and testing to understand why it happens.

Another final issue that I had witnessed was an issue with the jumping mechanic. On some Windows systems, the main character would not be able to jump most of the time the jump keys were pressed. On one of the machines, the issue would somehow resolve itself when the game was run in Administrator mode but in the other, it continued to be an issue. Though I am not sure what exactly triggers this bug, I believe that I could fix it by changing the script I wrote to enable jumping.

CONCLUSIONS

Though I have created games in the past, they were nothing like this one. This game, by far, has been the most intricate game that I have ever even attempted making. I surpassed my wildest expectations and accomplished more than I thought that I could. Through it, I was able to not only create another game to add to my growing portfolio but to explore a fresh style of game while also trying to implement machine learning for the first time. It aligned with my interests and career goals and so I am thoroughly proud of the work I have done on this project. That being said, I do not see this project as being complete. I intend to take this project and develop it further. Once fully completed, I will publish the final version for free online.
Future Perspectives

There is a lot that I hope to add to my project after this. I would like to create more levels as well as introduce new enemies that would be increasingly harder for the player to avoid, creating a more interesting game experience. Here are some of the additions or changes that I intend to incorporate in this game in the future:

- More levels; after every five levels, the player will be able to save their overall progress.
- Levels with boss enemies that the character could either wait out, fight, or bribe.
- Levels where the controls are inverted (example: the left key makes the character go right and vice versa), upside-down, or underwater.
- Levels with more than one exit (each leads to a different level) and some where you can go to the previous levels.
- Levels where the door to the next door will only open when a mission or achievement has been completed or acquired (examples: defeating an enemy, collecting a certain amount of money, finding a hidden key, etc.)
- Various kinds of collectables (examples: a food item that can make the main character invincible for five seconds or a disposable item that can be used to freeze all the enemies in a level).
- More non-machine learning taught enemies.
- More machine learning-taught enemies (some taught using reinforcement learning and maybe a few using imitation learning). For example, I would like to have an enemy that learns to navigate around the level while being affected by gravity.
- Fixing the issues encountered, starting with the collisions by writing scripts that would replace the ready-made components that I used.
- The usage of other forms of artificial intelligence (example: having enemies use pathfinding technology to find the shortest path to the main character).
REFERENCES


[33] P. de Byl, Pushing the Character Forward. Udemy.


APPENDIX A: LEVEL DESIGNS

Figure 69: Level 1 Design
Level 2

Figure 70: Level 2 Design
Figure 71: Level 3 (part 1) Design
Figure 72: Level 3 (part 2) Design