April 2015

Dark Horse

Andrew George Aveyard
Worcester Polytechnic Institute

Kenrick C. Tsang
Worcester Polytechnic Institute

Samuel Brian Machlin
Worcester Polytechnic Institute

Timothy Nicholas Calvert
Worcester Polytechnic Institute

Project Number: DMO-AA14

Dark Horse

Development of a 2D Tactical Combat Role Playing Game

A Major Qualifying Project Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science
By

Andrew Aveyard, Tim Calvert, Sam Machlin, Kenrick Tsang

Advised by:

Professor Dean O'Donnell
Professor Jennifer deWinter

April 27, 2015

Abstract

Dark Horse is an Interactive Media and Game Development Major Qualifying Project in which a team of four students took a simple game concept through the stages of development, testing, and refinement to create a story rich, two-dimensional tactical combat game using the Unity2D engine. Most of our assets were hand drawn by our artist, while the rest of our art was either outsourced or built by our programmers. Our vision for Dark Horse was to design a turn-based strategy game with a narrative focus in order to create tactical gameplay grounded in a well-developed narrative experience, a rarity for the genre.

This report gives an overview of our project as well as details our multiple areas of development: design, programming, audio, art, and narrative. With design's chapter, we describe our initial processes of generating mechanics and content as well as our iterative process of playtest driven design. Our programming chapter goes into detail about the different features programmed into the game, how they were constructed, and how successful they were in completing their intended purpose. The audio chapter describes the custom music and sound effects we developed for the game, how we made them, and how they were used in our game. The art chapter similarly explains our customized art assets and how they were generated in regards to color coordination and its context within the story. Finally, the narrative chapter explains not only why we chose to develop a dark sci-fi story, but also the process behind it and how it affected the rest of our game.

Acknowledgements

Our sincerest thanks go out to our advisors Professor O'Donnell and Professor deWinter, without whom we would not even have a project to begin with. Their advice and constructive feedback has helped mold our game into being the best it could be.

We would also like to give a warm thank you to Alex Clemens, who made our digital portrait art assets as a freelance project. His diligent work brought our game's story much closer to a professional level than it would have been otherwise. Alex's work has been crucial to our final product's presentability.

Finally, we would like to acknowledge Sebastien Bini, the developer behind Game Character Hub; and Degica, the company that developed several RPG Maker Assets. Using Game Character Hub, we were able to use separated RPG Maker sprite assets assembled together to create custom sprites for our combat sequences. These tools were critical to improving the quality of our gameplay art, without which our game would have suffered immensely.

Table of Contents

Abstract 1

Acknowledgements 2

Table of Contents 3

List of Figures 6

Chapter 1: Introduction 7

1.1 Development 7

1.1.1 Design 7

1.1.3 Narrative 9

1.1.2 Programming 10

1.1.3 Art 11

1.1.4 Audio 11

1.2 Conclusion 11

Chapter 2: Design 13

2.1 Narrative 13

2.2 Environment 14

2.3 Combat 14

2.4 Polished Art 15

2.5 Modular Code 15

2.6 Gameplay Summary 15

2.6.1 Cutscenes 15

2.6.2 Missions 17

2.6.3 Units 18

2.7 Conclusion 20

Chapter 3: Technical Development 21

3.1 Scene Generation 21

3.1.1 Combat Maps 22

3.1.2 Cutscenes 25

3.1.3 Parsing 26

3.2 Grid 26

3.3 Units 27
3.4 Managers 28
3.4.1 Universal Managers 29
3.4.2 Cutscenes 29
3.4.3 Combat 31
3.5 Terrain 32
3.6 Pathfinding 32
3.7 Artificial Intelligence 33
3.7.1 Packages 34
3.7.2 Conditions 35
3.7.3 Actions 35
3.8 Input 36
3.9 GUI 37
3.10 Audio 37
3.11 Saving 38
3.12 Conclusion 39
Chapter 4 Art Development 40
4.1 Color Theme 40
4.2 Cutscene Backdrops 41
4.2 Sprite Maps 41
4.3 Conclusion 44
Chapter 5: Audio Development 45
5.1 Sound Effects 45
5.2 Original Compositions 46
5.3 Conclusion 47
Chapter 6 Narrative 48
6.1 Story Outline 48
6.2 Designing the Story 48
6.3 Conclusion 52
Chapter 7: Post-Mortem 53
7.1 What Went Wrong 53

7.1.1 Scope Issues 53
7.1.2 Art Pipeline Issues 53
7.1.3 Design Coordination Issues 54
7.2 What Went Right 54
7.2.1 Technical Success 54
7.2.2 Audio Success 55
7.3 What We Would Do Next Time 55
Chapter 8: Bibliography 56
Appendices 59
Appendix A File Format Examples 59
A.1 Level 59
A.2 Cutscene 61
A.3 Save 62
A.4 Attack 62
A.5 Stats 62
A.6 AI 63
Appendix B: Audio 64
B.1 Composition 64
B.2 Sound Effects 64
Appendix C: Art 65
C.1 Backdrops 65
C.1 Spritesheets 66
Appendix D: Design Documents 69
D.1 Characters 69
D.2 Classes 72
D.3 Mechanics 73
D.4 World Description 74
D.5 Story Synopsis 75

List of Figures

Figure 1: Paper Prototype 8
Figure 2: Playtesting 9
Figure 3: Level Text File 10
Figure 4: A Cutscene 16
Figure 5: Cutscene Design File 17
Figure 6: The Tox 18
Figure 7: Units 19
Figure 8: Map and Cutscene Generators 21
Figure 9: Text File Map 22
Figure 10: Text File Players 23
Figure 11: Text File Enemies 23
Figure 12: Text File Terrain 24
Figure 13: Text File Characters 25
Figure 14: Text File Stage Directions 25
Figure 15: The Grid 27
Figure 16: Text File Stats 28
Figure 17: Text File Attacks 28
Figure 18: Managers for Cutscenes 30
Figure 19: Managers for Combat 31
Figure 20: AI 34
Figure 21: Text File AI 35
Figure 22: AI Conditions 35
Figure 23: AI Actions 36
Figure 24: UI 37
Figure 25: Text File Save 38
Figure 26: World Map 40
Figure 27: Tox Background 41
Figure 28: Fire Emblem 42
Figure 29: Hase Lounge Spritesheet and Map 43
Figure 30: Vrell's Lab Spritesheet and Map 44
Figure 31: Table of Sounds 46
Figure 32: Composing in Sibelius 47
Figure 33: Document Detailing Main Characters 49
Figure 34: Document Detailing the World 50
Figure 35: Document Detailing the Story 51

Chapter 1: Introduction

Dark Horse is a 2D Tactical Combat Game in which players control various different classes of units in order to complete various different missions by utilizing both the individual strengths of the units controlled as well as the strategic layout of each level. During each mission, the units gain experience and grow stronger in order to complete more difficult levels. As the player succeeds, she will gradually watch a story of revolution, corruption, and betrayal unfold.

1.1 Development

Our game was developed following the core idea of creating a tactically complex turn based strategy game that closely followed a linear narrative. We sought to achieve tactical complexity initially through a multitude of different player mechanics, but due to time constraints and scaling issues during early development, we realized this would not work. During our playtest sessions, however, we noticed more positive responses to terrain changes than to statistic value changes. This led us to create opportunities for tactics using the layout of our maps instead of numerous mechanics, which proved to be both simple to implement and useful for adding tactical depth to our gameplay.

As for the linear narrative, we decided to borrow the cutscene style of other popular turn-based strategy games by using still image portraits to represent characters and a text box to represent speech. We made this choice simply because our research showed this to be the industry standard for the best games in the particular genre and style that we sought to represent with Dark Horse, but we were committed completely to the style as it turned out to be a fairly simple system to program and modularize.

Each of our different areas of production (design, narrative, programming, art, and audio) kept this guiding principle at the core of each of its iterations.

1.1.1 Design

Our process was started with a series of design documents, rather than a single master document. Drawing from professional advice as well as our own personal experience with designing previous games (such as the final project of a tabletop strategy game design course), we felt that this approach would allow for a faster diffusion of design details to the rest of the group and be an easier organization system to use than a master document. These documents were written before physical development began to serve as a starting place for the iterative design process. The documents cover the various aspects of our game:

- **Characters** - The names and bios of each of the main characters of our game.

- **Story Synopsis** - A brief description of the intended overall story progression of our game
- **Classes** - The different classes that appear in the game and their differences.
- **World Description** - Details a brief history of the game's narrative setting.
- **Mechanics** – Lists the different statistics assigned to characters as well as defines the type of combat we wanted in our game.

These documents can be viewed in their entirety in Appendix D.

The level design of each of the missions was handled differently from the mechanics and story, relying on physical designs rather than digital design documents. Initially, each level was produced as a paper prototype and went through a few cycles of playtesting using the predetermined game mechanics and AI behaviors. This was a continuing process that cycled between phases of creating, where the prototypes were actually made; playtesting, where we brought the latest prototype into the hands of playtesters; and finally debriefing, where we gathered feedback from the playtesters before returning to a creating phase. Below you can see Figure 1, which is one of the first prototype levels that we designed, and Figure 2, a playtesting session of Figure 1.

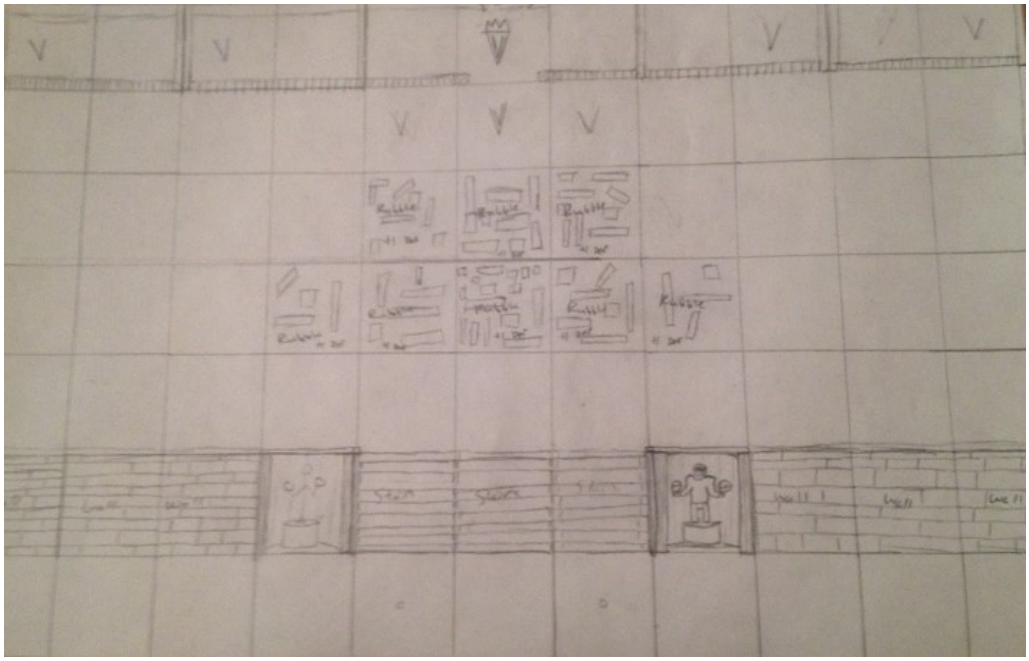


Figure 1: Paper Prototype

Figure 2: Playtesting

During early playtest sessions, our goals were primarily to collect feedback on level, mechanic, and content design. For example, many play testers expressed a concern that the levels felt too bare and lacking without different types of terrain. This in turn led to the development of several different terrain types. Later playtests were primarily concerned with balance in order to ensure the difficulty of the level was on par with the skill level of the players and the abilities of the characters.

As the designs grew more intricate and our technical development advanced to the point where we could construct digital levels, we dropped the designing of paper prototypes in lieu of purely digital level designs that were iterated by our group.

1.1.3 Narrative

Our narrative drafting process started at the outset of our project and continued to be iterated throughout the entirety of development. The story was first drafted using design documents to develop the characters, the world, and the story arc separately from each other. This approach allowed a much easier way of singling out and modifying key features of our narrative setting without having to sift through massive blocks of text.

As actual story writing began, we decided to take a “start big and pare down” approach to drafting our storyline. We originally intended to have a 10 chapter story with 10 battles, but this proved to be too herculean to accomplish within our limited time frame. We scaled this down to

a 5 chapter storyline with 5 battles, which we regarded to be the minimum viable product possible before having to alter the flow and major events of the narrative. Later into the drafting process, we began giving it to audiences in order to playtest its length and ability to capture an audience. A deeper look into the game's narrative is covered later in Chapter 6.

1.1.2 Programming

Through the entire creation of *Dark Horse* the philosophy behind programming was to have clean code that could easily be debugged or changed no matter who wrote the code. Our other goal was to have almost nothing hard coded, meaning any piece of code could be reused somewhere else in a different way. The figure below is an example of a map design file. The code takes the numbers in-between the "StartMap" and "EndMap" and converts them into a full map by associating each of the numbers to a specific art asset and then placing each asset in its proper position on the game screen as dictated by the design file. It reads and interprets the other information in the file in a similar manner.

```
# After that comes the map description, bookended by 'StartMap' and 'EndMap'
StartMap
2 6 5 1 5 12 15 12 6 3 6 5 4
18 30 29 29 30 29 11 30 29 30 29 29 7
39 26 25 32 25 11 10 11 26 25 31 25 38
16 25 31 25 26 10 11 10 25 31 32 26 42
41 32 11 26 25 31 25 31 26 25 11 25 17
39 10 11 32 26 25 32 25 25 31 10 11 40
37 26 25 25 31 26 25 31 25 25 11 31 17
16 32 25 32 26 25 32 25 32 25 26 25 42
14 13 13 13 24 21 19 20 14 13 13 13 24
14 13 23 13 23 21 19 20 22 13 22 13 24
29 29 30 29 29 30 30 29 30 29 29 30 30
EndMap

# Players
StartPlayers
    # Each player gets their own section
    # Need to specify name
    # Stats file
    # Level
    # List of attacks, comma separated
    # And position
    # The spacing in the position tuple *does* matter. There cannot be a space after the comma.
    # Things can be indented if it makes it easier to write
    StartPlayer
        Name: Faryll
        Stats: faryll
        Level: 1
        Attacks: melee
        # If there were multiple attacks It would look like this:
        #Attacks: melee,ranged
        Sprite: Faryll
        Position: (8,0)
    EndPlayer
    StartPlayer
        Name: Jered
        Stats: jered
        Level: 1
        Attacks: ranged
        Sprite: Jered
        Position: (4,0)
    EndPlayer
EndPlayers

# Enemies
# These follow a similar pattern as players
StartEnemies
    StartEnemy
        Name: Prisoner
```

Figure 3: Level Text File

1.1.3 Art

Dark Horse contains both pixelated and line drawn art. Each level, the purpose of the art was to incorporate a color theming with each background and map in order to create a more captivating world for the player.

For the most part, each backdrop was created with darker colors, so that the player can focus more on the characters instead of what lies behind them.

Regarding the map for each level, we wanted to create a way where we can quickly make edits to them without wasting valuable time. Thus, we decided to use spritesheets because they can be quickly modified and efficiently turned into maps.

Overall, we were able to create an immersive world for the player to experience.

1.1.4 Audio

The major goal of the audio was to create original sound effects and compositions in order to provide certain emotional content for the game.

Using different microphone placement techniques, we recorded each sound effect, and further made edits to them in various digital audio workstations. The reason we did the latter was so that the quality of the sounds can be further enhanced.

All musical pieces were composed in a notation program. This was the easiest and most efficient approach we could take because the software allows for quick changes to the music.

Audio is certainly an essential element of the game, and we wanted to make sure that each has as high a quality as possible for the player to hear.

1.2 Conclusion

While each of the aspects of production had varying development processes, we worked very closely together during every stage. The design team needed to know about the programming team's process in order to get an idea of what could and couldn't be developed. The art team needed to know about the design team's process in order to know what type of assets needed to be developed and how they needed to be constructed. The programming team needed to know about the art team's process in order to know whether or not a specific feature could even be communicated to the player through art. The narrative team needed to know about everyone

else's process to get an idea of how long or short to make the story last. Our development was varied and diversified, but also interconnected and inseparable.

Chapter 2: Design

From the outset of Dark Horse's developments, we have constantly strived to achieve certain design goals. These goals are:

- Draft and integrate an emotional and interesting narrative
- Create an engaging gameplay environment
- Implement a deeply strategic and intensely fun tactical combat system
- Generate and integrate polished 2-D sprite art assets
- Program clean and modularized code

Each of these goals played an important role in leading the development of our game in different ways.

2.1 Narrative

The story of Dark Horse was the first part of our game with any significant detail. Having a darker storyline for our game has acted as our compass in designing the environment, combat, and art of our game and has always been a key part of our designs. Other strategy games tend to have weaker, happier stories and we wanted to set Dark Horse apart from the crowd with a strong narrative.

The story design began with light character development (generally names, archetypes, and small back story blurbs) in order to get an idea of who our actors were. We set out to design a compelling game story, but to do that you need compelling characters first. As we refined our characters and their personalities, ideas for a story slowly formed.

An outline of the main story arc preceded the scene by scene write up. As the story was written before the development process got under way, we knew that it would most likely need to be pared down later on. Utilizing this foresight, we decided to create initial write-ups only for earlier scenes that were definitely going to be in the game. As prototype development and playtesting progressed, it became clearer what we could and couldn't have in Dark Horse and our story changed to reflect that.

In the end, however, our narrative was not able to make it completely into Dark Horse simply due to a lack of time to properly iterate and implement it and also have a playable version of our game.

2.2 Environment

In order to develop our narrative's sinister themes, we realized that we would need an equally sinister setting for our game. Creating the urban dystopia of Arkehlm was an effective method of mirroring these themes in our environment as the dystopia typically is a breeding ground for crime. Arkehlm was then dissected into different sectors to allow for a diversified cast of actors while keeping them all connected by the city's corruption. Additionally, the specific sectoring of the city made it easy to design immediately identifiable levels without requiring a large amount of assets to diversify them.

2.3 Combat

The combat system we wanted to implement resembles the turn-based strategy system from games like XCOM and Fire Emblem. Its two main components are the statistics of the characters and the level itself.

We wanted our statistics to bridge player and character, so we included a leveling system that would improve character damage, defense, movement range, and health as the character defeated more and more opponents. By doing so, we hoped to attach a sort of ownership to the characters and as a result, provide a stronger affective link to the game through its actors.

The statistics governed the actions you could perform with your characters during combat. To keep things simple, we limited character actions to movement and attacking and had them both use the same resource pool to be performed. This allowed us to focus on creating opportunities for strategic decisions through interesting level design that required players to utilize their limited options effectively instead of through the navigation of a laundry list of complicated mechanics. In short, we decided that simpler combat rules could and would be better for our game.

With our focus on level design, we strove to create tactical playgrounds for our players to explore. To this end, we first incorporated a grid system separated into tiles, forcing characters to move around the map tile by tile. With a grid in place, we could easily and neatly manipulate the strategic value of a level's terrain piece by piece. By designing different terrain types, we created different ways for players to actively experience and utilize the level (luring enemies into chokepoints created by impassable terrain, avoiding damaging terrain, guarding from defensive bonus terrain).

2.4 Polished Art

Asset generation was closely tied to our narrative theming all throughout the entirety of the development pipeline. We wanted to not only make each sector narratively distinct, but also artistically distinct. To do this, we used simple, yet powerful color associations with the different sectors of Arkehlm. This in turn allowed us to extend these associations to the characters in our game when designing their portraits and sprites. By doing so, we were able to have the art help the narrative with its story telling by developing the environment through visuals as well as through dialogue.

2.5 Modular Code

From the start, efficient and modular code was always going to be our priority over having varied infrastructure to support a multitude of mechanics. Having reliable, working code is the backbone of any digital game, and this can only be achieved through careful attention to detail when coding. Additionally, we wanted to develop code that could be re-used to create other games in the future if students wished to use it. This naturally called for modularization, as we couldn't rely on hard coding pretty much anything into our game if we wanted to achieve this goal of reusability in the future for other projects.

2.6 Gameplay Summary

Seeking to accomplish these design goals created our gameplay in its entirety as it is today. Our game entails one player in control of two to six specialized units in specific missions. The player's goal in each mission is to eliminate the enemy's forces. The player must utilize both the environment and the talents of her soldiers in order to achieve this goal. When a mission is completed, a cutscene or a series of cutscenes advances the storyline and provides context for future mission.

2.6.1 Cutscenes

The cutscenes are the primary communicators of our story. We chose to limit the scope of cutscenes to the perspective of our main character to limit the amount of information that would be relayed to the player. We believed that this would allow us to provide a more personal story by transporting the player into the main character's role, rather than an abstract godlike being watching over ants, a narrative pitfall that other strategy games tend to fall into. The relationship between the narrative and the cutscene design was intimate. For example, choosing to make the

cutscenes from a character's direct perspective completely changed the story from its initial form. Figure 4 below shows a cutscene interaction between Faryll (left) and Jered (right).

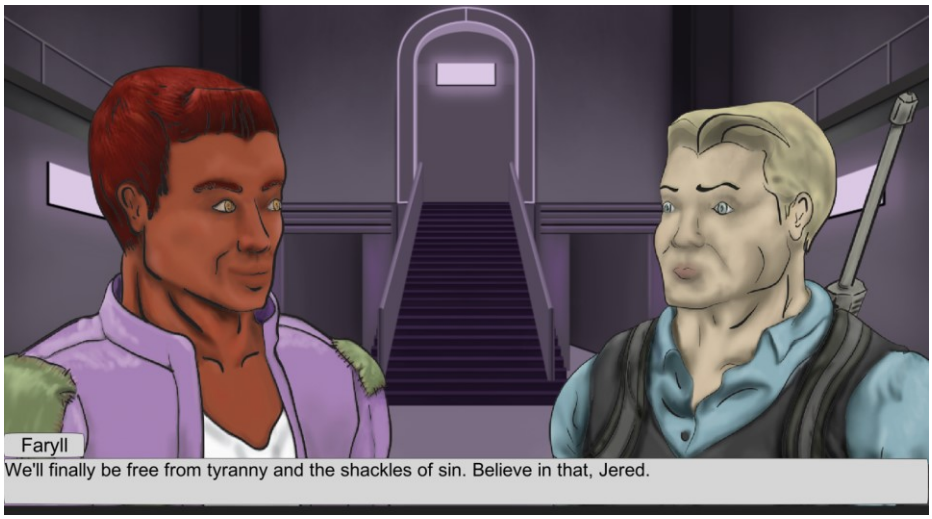


Figure 4: A Cutscene

Due to art and time restraints however, the story we wanted to tell could not be told. In future development, the game would have a stronger focus on narrative and have more ties to the story setting in combats as well as more developed backstories in cutscenes.

In terms of how we chose to make our cutscenes, we decided to model our design after Fire Emblem's cutscenes. This meant creating character portraits that would slide into and out of view, as well as fade in and out as the scene's current speaker changes. We decided to use this style because we believed that this was a key factor in how Fire Emblem gets you to care about its characters, and this is often the approach mirrored in many other strategy games (Valkyria Chronicles, Final Fantasy Tactics Advanced, Advance Wars, and others). While we wanted to be original, in this area we felt that this is simply the best strategy for design, art, and programming.

This resulted in setting up our cutscene tools as text files that resemble a play script. The files use "stage directions" for portraits, sound, and background music which are read by the engine and then interpreted as actions for objects within the game. An example of one of these files can be viewed below.

```
#Backdrop used: Basteon Holding Cells
Characters

Faryll Left
Jered Right
Wenden Right

# Background musicBackground Music# specified audio is relative to 'Sounds' folder# it sh

Scene

[Jered] MoveInToRight
[Jered] FadeIn
Jered: That's the last of them.
[Jered] FadeOut

[Faryll] FadeIn
[Faryll] MoveInToLeft
Faryll: Good. I'm still worn out from that stranger's attack before.
[Faryll] FadeOut

[Jered] FadeIn
Jered: That man...he swatted us away like flies.
[Jered] FadeOut

[Faryll] FadeIn
Faryll: He possessed incredible power.
[Faryll] FadeOut

[Jered] FadeIn
Jered: I hope he wanders into my crosshairs one day.
[Jered] FadeOut

[Faryll] FadeIn
Faryll: Redirect that anger to the mission Jered. We can't afford to get distracted now.
[Faryll] FadeOut

[Jered] FadeIn
Jered: Of course. The revolt is tonight after all.
[Jered] FadeOut
```

Figure 5: Cutscene Design File

Our cutscenes were designed specifically to precede or follow a combat or another cutscene and varied based on this positioning. Cutscenes that precede a combat have some sort of contextual explanation of the characters' objectives and their motivations in the coming mission. Cutscenes that precede combat give a recap of the outcome of the battle as well as its consequences on the future. Cutscenes preceding or following other cutscenes are arranged to transition between areas in a progression that makes sense to the player so that the cutscenes preceding combat can focus solely on explaining the task at hand.

2.6.2 Missions

Dark Horse uses a turn based system for missions, allowing the player to move her units as she wishes, then performing the computer's movement after the player has ended her turn. When units engage in standard combat, the unit being attacked takes damage while the initiating unit does not take any damage (unless otherwise noted). A standard mission is considered complete when all enemies on the map have been defeated, and a mission is considered failed when a player loses all of her units in a combat

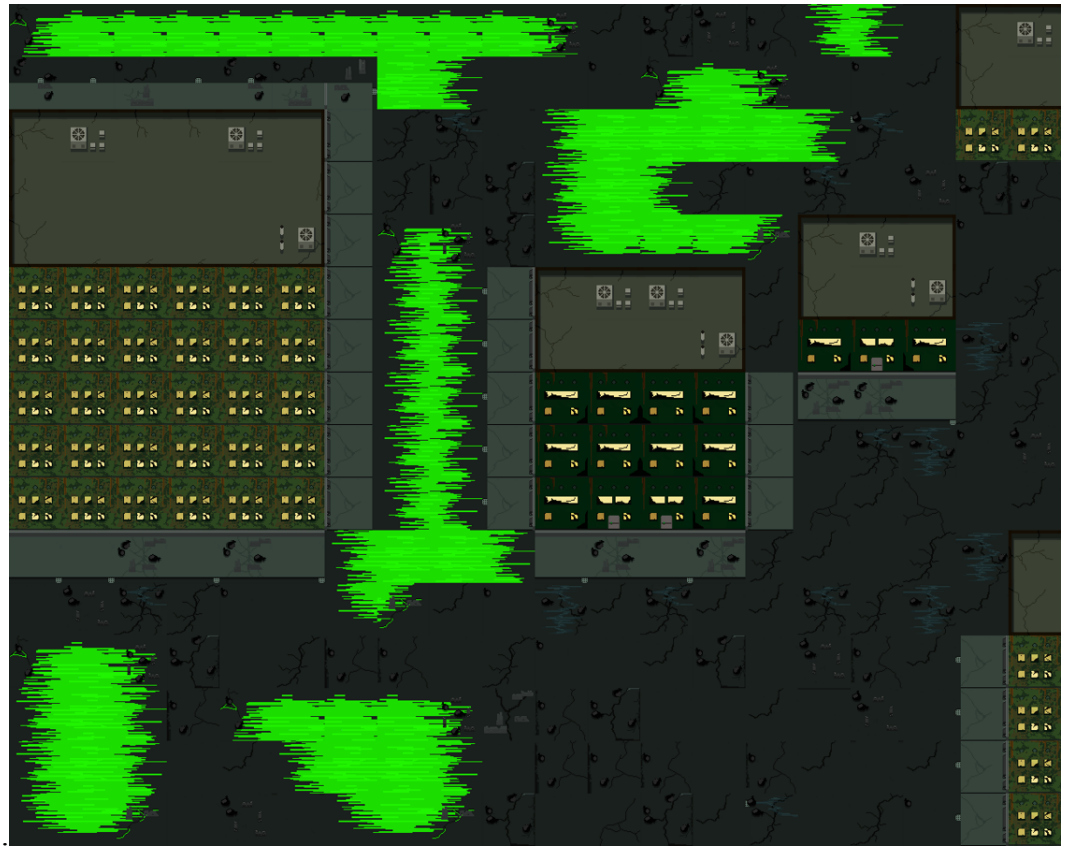


Figure 6: The Tox

Each mission is a completely new map arrangement with new tactics and strategic positions to utilize. These arrangements were first realized as paper prototypes. Paper prototypes were initially extremely useful, as they allowed for early balancing, designing new mechanics, and most importantly, playtesting. The power of paper was its mobility, allowing us to make any room with a flat surface and willing victims into a playtest session. These prototypes helped refine our mechanics and revealed different problems with our core design, both of which helped our project become a more enjoyable game.

Eventually, our levels were iterated enough to be transferred over to digital prototypes. In the digital realm, we began creating new prototypes much faster and easier due to the nature of our digital infrastructure. These prototypes were used primarily to reveal technical glitches and balancing issues, observe art responses, and gather narrative feedback.

2.6.3 Units

Both player controlled and AI controlled units have statistics based on unit type and level. Unit stats were all stored in text files for easy modification and balancing. An example of Unit stats

can be seen in Appendix A. The statistics are action points (AP), health points (HP), damage, and defense. Units have initial values set for these stats which increase as the unit's level increases.

Figure 7: Units

During the intended course of the game, a player would gain control of six different units:

- **Jered**: The main character of the game. You see things from his point of view in the cutscenes. An ex-soldier and a noble of Basteon, he is old friends with Faryll.
- **Faryll**: As a young man, he was enslaved for his refusal to become a Drake for many years until Jered bought his freedom. He leads the Dregs, a gang that seeks justice and the group the player controls.
- **Wenden**: A woman enslaved by the Drakes when they burned her family alive. The nerves in her hands were burned to nothing when she tried to put out the flames on her family's corpses. She became friends with Faryll when chance brought them together in the fighting pits of the Red Lantern.
- **Fhec**: Freed by Faryll's slave rebellion, he follows him around as he claims Faryll stole his purpose as a slave when he killed the nobles who owned him. He was a messenger during his time enslaved.
- **Aeria**: A soldier who rose through the ranks to gain control of an entire city sector. She begins as an enemy, but grows to respect and eventually join Faryll's gang.
- **Otto**: An escaped robot made only for war. He gained sentience and realized that his creators viewed him only as a tool of death and he fled to an abandoned sector to escape his fate. He joins Faryll's gang to help defeat a massive threat to Arkehlm.

You can read full descriptions of these characters in Appendix D.1.

We wanted a diverse cast of characters in order to provide more narrative interest through relationships. This led the design of our characters' backstories and their roles in both the story and the gameplay. By providing characters with varied personalities, we were able to integrate these traits into gameplay in order to provide well developed characters and tie these into their gameplay strengths and weaknesses. For example, Wenden has low damage but high health and defense and therefore can take more hits than the other characters. This borrows from Wenden's burned nerves in her hands, which as a side effect, lets her not experience pain when fighting with her fists. Thus, we created a contextual explanation for the units' gameplay roles which reinforce their individual personalities and develop their status as characters rather than simply units.

2.7 Conclusion

Our game's design focused on accomplishing 5 main goals. While many scaling and time constraints had a negative effect on our accomplishing all of these goals, many of them were completed. Our base combat provides a strategic challenge for players, our art was polished and evoked emotion in its beholders, our environment is carefully crafted to tie strongly into the narrative, and our code is clean, modular, and highly adaptable. We fall short of our narrative goal simply because we did not have the time to implement it in its entirety and iterate it to its best form. If development were to continue, we are confident that this goal could be reached.

Chapter 3: Technical Development

Dark Horse was created in the Unity game engine, using C# for scripting. Unity has very good 2D and sprite support making it a good fit for our game. Additionally, our technical members were all familiar with Unity. We used the NGUI package to create the user interface. We had two main goals when designing and writing our code. Our goals were:

- Create clean, modular code
- Give artists and designers the tools to quickly iterate

We accomplished our primary goal of clean, modular code by creating coding standards and using a manager system to organize responsibilities as explained in Chapter 3.2. Our secondary goal was completed by creating tools to allow the designers to specify all relevant information in text files as described in Chapter 3.1.

3.1 Scene Generation

We quickly found that our maps and cutscenes had a large number components that went into them. Because of this, creating maps and cutscenes by hand was tedious and error prone. However, the components that made up maps were frequently repeated between scenes. This presented a good opportunity to generate all of our scenes automatically from text files. To accomplish this, two Unity editor plugins were written to generate the scenes, as shown in Figure 8. One was responsible for creating combat scenes and the other created cutscenes. By making our scenes this way, we saved a lot of time that would have been spent building scenes and fixing errors.

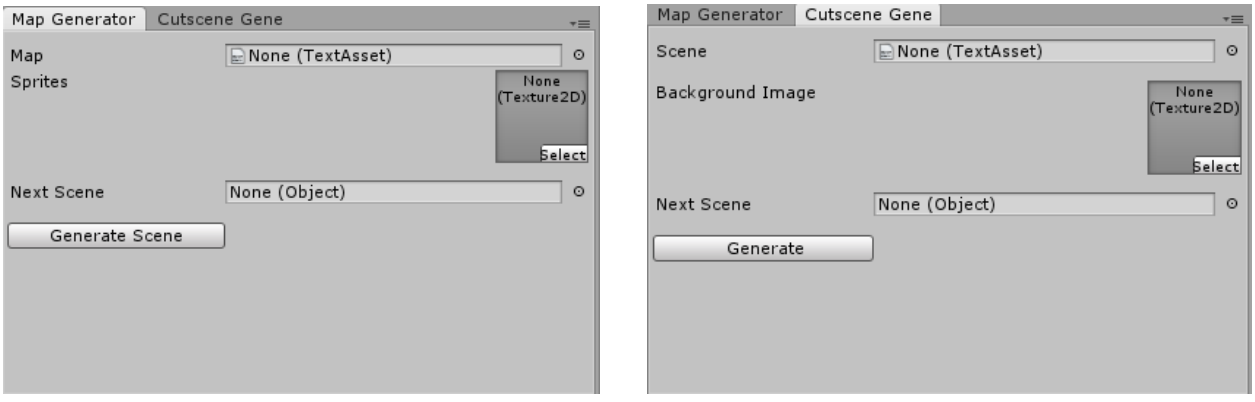


Figure 8: Map and Cutscene Generators

For an explanation of the file formats used see the chapters below, or see Appendix A for other examples.

3.1.1 Combat Maps

The editor plugin that generated combat maps used a text file that was filled with information that described everything necessary to build a combat scene.

```
Name: Basteon Prison
Size: 13,11

StartMap
2 6 5 1 5 12 15 12 6 3 6 5 4
18 30 29 29 30 29 11 30 29 30 29 29 7
39 26 25 32 25 11 10 11 26 25 31 25 38
16 25 31 25 26 10 11 10 25 31 32 26 42
41 32 11 26 25 31 25 31 26 25 11 25 17
39 10 11 32 26 25 32 25 25 31 10 11 40
37 26 25 25 31 26 25 31 25 25 11 31 17
16 32 25 32 26 25 32 25 32 25 26 25 42
14 13 13 13 24 21 19 20 14 13 13 13 24
14 13 23 13 23 21 19 20 22 13 22 13 24
29 29 30 29 29 30 30 29 30 29 29 30 30
EndMap
```

Figure 9: Text File Map

The first section of the text file as shown in Figure 9 has the name of the map, the dimensions, and the numbers that describe specific tiles from the map's tilesheet. The tilesheet is a single image that contains many smaller images that can be composed to create a seamless map. The numbers are used to pick out one of the smaller images. Together, the grid of numbers describes the entirety of the image that represents the ground of the map. The benefit of composing the map this way is that the artist only needs to create a single tilesheet and the designer can change the look of the map without having the artist do more work. The resultant map can be seen in Figure 15.

```
StartPlayer
    Name: Faryll
    Stats: faryll
    Level: 1
    Attacks: melee
    Sprite: Blue
    Position: (8,0)
EndPlayer
```

Figure 10: Text File Players

The next two sections describe where player and enemy units on the map. Each unit includes the name of the unit that is displayed to the player, the file that describes all basic stats for the unit, the level of the unit, a list of files that describe that attacks that the unit has, the sprite that represents that unit on the map and the position where the unit should be placed.

```
StartEnemy
    Name: Prisoner
    Stats: brawler
    Level: 1
    Attacks: melee
    Sprite: Drake_Goon
    AI: BasicEnemyR3
    Position: (3,9)
EndEnemy
```

Figure 11: Text File Enemies

Enemies have an additional field. They each have the name of the file that describes their artificial intelligence routines which are loaded when the game is started. More information about artificial intelligence can be found in Chapter 3.7.

By describing all of the units in this manner the designer can quickly tweak any aspect of the players and enemies on the map without editing the code.

```
StartTerrain
(0,1)
(1,1)
(2,1)
(10,10)
(11,10)
(12,10)
EndTerrain

StartStatus
Rubble (1,5)
Rubble (7,7)
Rubble (7,8)
EndStatus
```

Figure 12: Text File Terrain

The final section of the file lists how certain parts of the map interact with the player. Terrain is an impassible area of the map and is usually used to represent objects like walls. Status effects are aspects of a tile that alter the way the tile interacts with a unit. For instance, rubble provides extra defense to any unit standing on it.

Similarly to the player and enemy units, specifying terrain or status effects such as roofs or rubble within the text file allows the designer to alter the map design and functionality in a timely manner.

These text files were invaluable in the rapid iteration of our levels, allowing the designers to build levels and then alter them based on player feedback. For example, player feedback suggested that more varied terrain would make combat more interesting. The text files allowed the designer to easily add more terrain just by adding additional entries to the appropriate section in the file.

3.1.2 Cutscenes

```
#Backdrop used: Jered's Compound
Characters

Faryll Right
Wenden Left
Jered Left
Slaver Right

Background Music
Loops/Midnight_Hour_Loop
```

Figure 13: Text File Characters

Similar to combat maps, cutscenes were generated from text files created separately from the code.

The first section of a cutscene text file was used to describe the characters in the scene, their initial positioning off screen, and the background music that would be played for the duration of the scene.

```
[Slaver] FadeIn
[Slaver] MoveInToRight
Slaver: Got a shipment for Jered Boulderfist. That you?
[Slaver] FadeOut

== Effects/Plasma Slash

Slave: AAGH!
[Faryll] FadeOut
```

Figure 14: Text File Stage Directions

The next section, as well as the rest of the cutscene text file, is used not only to write the dialogue of the scene, but also to portray the stage actions and sounds effects that occur throughout. Dialogue is represented by the speaker's name and a line of dialogue. Stage actions have the moving character named within square brackets and represent movements across the screen. For example, [Slaver] MoveInToRight would move the slaver character from off the screen, to the right side of the screen. Sound effects can be played by specifying the name of the sound clip preceded by double equal signs.

The goal of creating cutscenes in this manner was to allow our designers to write cutscenes like they would write a play. This allowed cutscenes to be conceived in a convenient and understandable format that could be reviewed by others without a readability issue.

3.1.3 Parsing

In order to use the text files described previously, we needed to create a parser which was designed to read and interpret the information stored in the files. To handle this we created three different parsers that handled the main elements of the game. One parser was used to read files describing combat scenes, another for cutscenes and the final one was used to parse all other files such as those used for artificial intelligence, unit stats, save files and others. These additional files are all structured very similarly and so could be parsed in a nearly identical manner.

We chose to design our own format and write a parser rather than use an existing one (JSON, XML) because the format we used allowed the text files to be written in a simple way that did not required any familiarity with programming. JSON and XML require the editor to have some programming knowledge as they are highly structured and include specialized syntax. Our own format, however, is very straight forward and does not require any special knowledge to write.

3.2 Grid

We chose to use a grid for two reasons. First, our game is an homage to games like *Fire Emblem* which traditionally use grid systems. Second, grids are usually very easy for players to understand. The structure and affordances of the grid are easily discernable which makes learning to play the game simple for the player.

The central part of our combat scenes was the grid that we used to store information about the world. When a combat scene loads, all units, obstacles, effects, etc. are found and added to the grid. This allows us to easily track where everything is and query the grid for information. Additionally, it is much more efficient to take a click position on the screen, convert that into the x and y coordinates of a cell on the grid and then use those coordinates to inspect a cell, rather than use raycasts to identify contents of the cell. For every cell, a number of things are tracked:

- 2d world position of the cell within Unity
- The index of the cell (x and y coordinates of the cell in the grid)
- The contents of the cell (either empty, terrain, player or enemy)
- The status effect (none, rubble, tox, barricade or roof)
- The Unity object if something is there
- A status effect icon that can be shown when a unit is in the cell

The image below shows the scene in Unity. The green squares indicate the various objects such as units, terrain and rubble, that are brought into the grid. The actual grid is not visible to the player. It only exists because objects in the scene are locked to positions such that they form a grid.

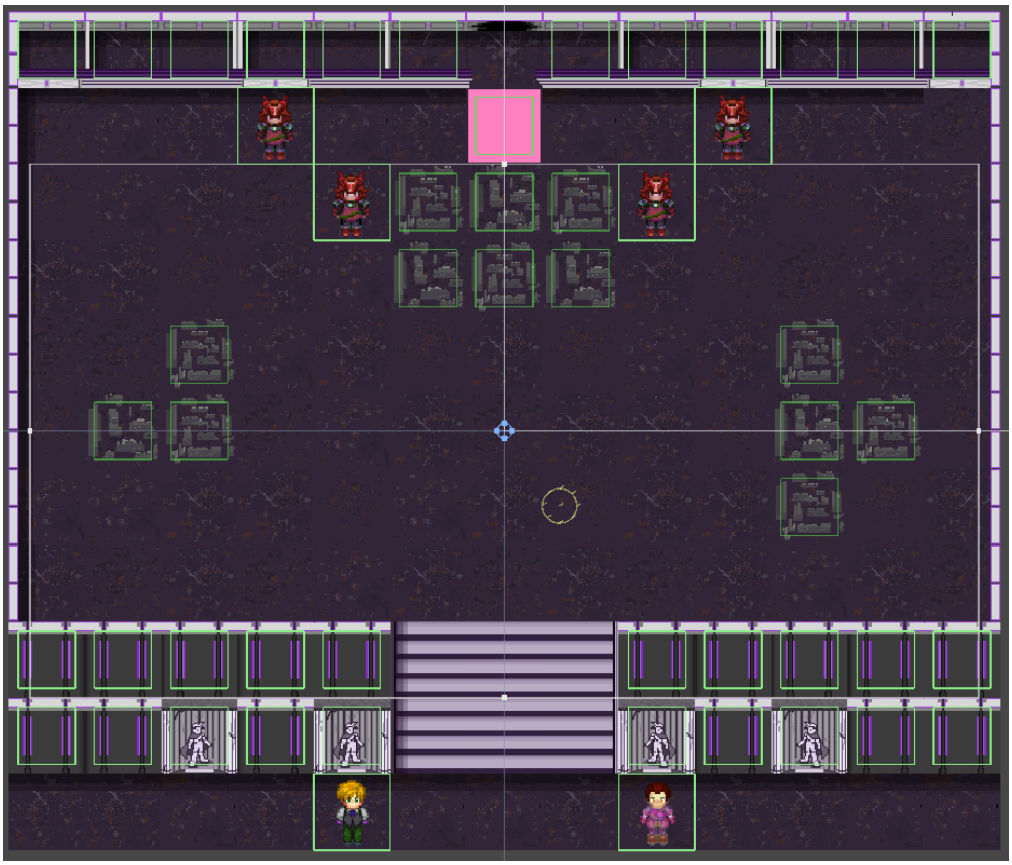


Figure 15: The Grid

By using this setup, we are able to quickly access information from the cell without performing the complex and expensive calculations involved in raycasting. The largest responsibility of the grid was to take the x,y coordinates of a cell and return all cells within a specified distance. There is a more used endpoint to this function that only returns open cells. This is used during pathfinding, identifying attackable enemies and anything else that requires knowledge of the surround area.

3.3 Units

All units have their base stats, current stats, and stats managed by the same piece of code. This made changing code in regards to the units very easy because changing the base class would change every unit at the same time. Initially the units have their base stats set based on the level

and unit type specified for them in the combat scene's text file (Chapter 3.1.1). An example of the stats for a certain unit type is shown below.

```
Stats
    Name brawler
    Health 8
    ActionPoints 3
    Strength 6
    Defense 1
```

Figure 16: Text File Stats

The text file also defines what attacks each unit has. The range, damage and sound for each attack is defined in its own text file as seen below.

```
Attack
    Name Melee
    MinRange 1
    MaxRange 1
    Damage 0
    Sound Knife Stab
```

Figure 17: Text File Attacks

The goal of our base unit class was to make it so the other managers could interface with it and tell units how their stats should be changed without having to know anything about how the stats are managed for an individual unit.

Units have health, strength, defense, and action points (AP). Units deal damage based on their strength and the opponent's defense. When a unit is hit with an attack it first removes the damage from its defense before subtracting it from health. When a turn ends the combat manager restores the defense of all units. Units themselves only handle modifying their stats. It is up to the combat manager to them when to modify it and by how much. There are multiple functions put in place to have units take damage, recover health, and recover AP.

3.4 Managers

We used a number of "managers" (also called a "controller" in the case of the master controller) to drive most of the game. The managers made it easy to organize our code and pass information around between various aspects of the game. By designing our code in this way, we were able to make it so each manager handles a certain part of the game and only interacts with other

managers through a clean, well defined boundary. This not only helped with keeping the code clean and modular, it made debugging a much faster process.

3.4.1 Universal Managers

There are three managers that exist at a higher level than all other managers, the master controller, the input manager and the resource manager. These exist in both cutscenes and combat scenes and are responsible for handling coordination between scenes, providing player input to other managers, and loading and parsing certain types of resources. The main purpose of hoisting certain functionality into these three manager was to make some of the more complicated activities simple to initiate. For example, loading resources and handling input are all relatively complicated things to get right. Providing a single interface to access them helps to simplify code and prevent possible errors.

The master controller is responsible for anything that takes place between scenes. It handles the process of loading the following scene when it is requested as well as saving and loading games.

The input manager is used as a shim between Unity's input mechanisms and our own game. We did this so we could provide input to other managers through event handlers and allow the player to remap keys. Additionally, it provides better control over what input is passed through. For instance, during the AI's turn we disable any combat related input so that the player cannot move.

The resource manager handles the loading and parsing of many of the text based resources used in the game. The files for unit stats, artificial intelligence and several others are loaded by this manager. It is used to provide a clean interface for loading these files and allows a manager to simply request that a certain file and leave the process of reading and interpreting the file, as well as handling errors, to a different manager.

3.4.2 Cutscenes

Figure 18 describes how our cutscenes work on a basic level. The input manager passes player input to other managers, the dialogue manager is responsible for displaying lines of text on the screen, the sprite manager handles presenting and moving sprites around the screen and the master controller handles the high level operations such as switching and starting scenes.

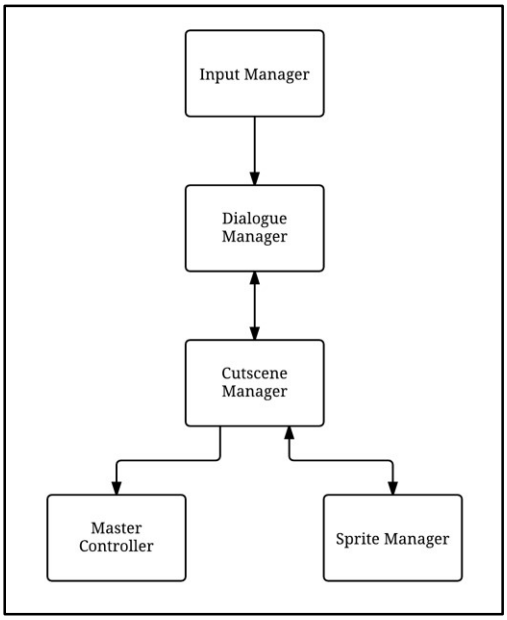


Figure 18: Managers for Cutscenes

The manager responsible for controlling the presentation of cutscenes is the cutscene manager. The primary responsibility of this manager is to parse the file describing the cutscene (3.1.2). After loading the file, the cutscene manager holds a list of all characters in the scene as well as a list of dialogue, stage directions, and audio cues in chronological order. It then runs through that list and handles the appropriate action. In the case of an audio cue, the cutscene manager plays the sound directly. For lines of dialogue and stage cues, control is handed off to the dialogue manager and sprite manager respectively.

Dialogue is handled by the dialogue manager. It takes instructions from the cutscene manager in the form of a speaker and a line of dialogue. The dialogue manager then proceeds to display the dialogue, printed out a character at a time to create a typewriter effect. It uses the input manager to allow the player to advance to the next line of dialogue.

The sprite manager is used to move around the characters in the scene. When the cutscene manager comes upon a stage direction, which is an instruction to either move a character around or fade out the character so that player focus can be moved to a new character, the information is provided to the sprite manager. It then finds the appropriate character and perform the necessary action before handing control back to the cutscene manager.

3.4.3 Combat

The combat manager provides most everything needed to run combat. All other managers defer back to it. Figure 19 shows the general flow of information between managers during combat. The sound manager provides an interface for playing sound effects. The items in the rectangles are the actual managers, the items in the ovals are auxiliary scripts or outside input that provide more information or more specific control to the managers.

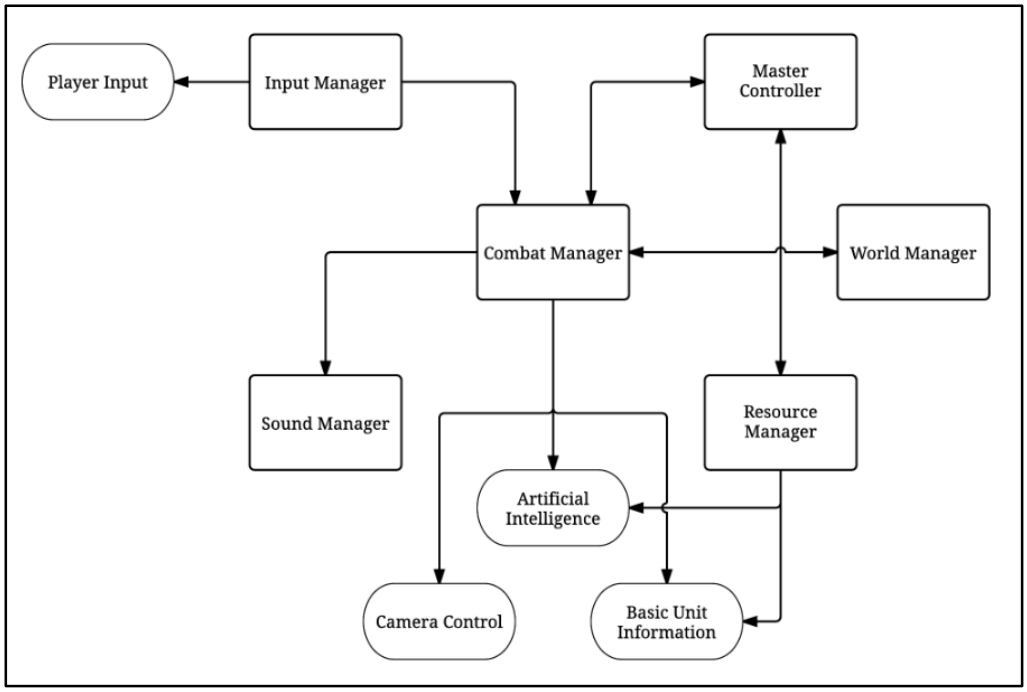


Figure 19: Managers for Combat

During combat scenes, the manager responsible for moderating gameplay is the combat manager. It holds information about all the units in the scene, manages turns, controls attacks and most everything else that combat requires. Anything that happens during a combat scene eventually finds its way into the combat manager. Having a central place for all important functionality makes adding features a lot easier.

All sound in combat scenes is routed through the sound manager. It provides a simple way for other components to request that a sound be played without having to deal with the additional issues of loading and playing it through the correct sound device.

The world manager stores all the information about the world. It has a grid that represents the map. The grid stores information for each cell, such as the contents and any status effects that might be on it. It is a representation of the grid as described in 3.2. It also provides a clean

interface to work with the data. For instance, the process of moving a unit from one cell to another is as simple as telling the grid to move whatever is in one spot to another spot.

3.5 Terrain

Terrain provided a much needed variation to combat scenes and servers an important purpose in structuring the environment and providing gameplay opportunities for the player. The primary piece of terrain was a simple obstacle. This was used to block off points on the map, allowing us to easily place restrictions on where units could move and therefore control the flow of the level. Another terrain type was roofs. Roofs helped to provide a vertical aspect to combat. Certain units are allowed to make the transition from ground to roof (and vice versa). Additionally, roofs limit attack opportunities. Only ranged units can attack across the boundary. Once a unit is on the roof combat occurs normally within the confines of the roof. The remaining terrain types provide small status effects to any units standing in the same cell. Tox pools slowly damage units standing on them, rubble provides a one point bonus to defense and barricades, which face a certain direction, damage any unit attacking into it. For instance, if a barricade facing north is placed in a cell, any character attacking from the adjacent cell to the north takes damage on attacking.

3.6 Pathfinding

Player movement is handled by clicking on the desired destination and letting the system find a path for the player. Enemies move in a similar fashion, though the destination is decided on by the artificial intelligence code. All paths are found with an implementation of the A* algorithm. A lightweight minimum binary heap and priority queue were developed to increase the speed of the algorithm. When finding paths, the path was allowed to go through, but not stop on, a cell currently occupied by a unit of the same type as the moving unit. For example, players can move through other players, and enemies through enemies. One of the biggest challenges was implementing roofs. Our pathfinding system was not setup to handle the complexities that roofs added to the landscape. To deal with them we needed to add certain special conditions to handle the various cases of roof usage (i.e., the unit's ability to climb roofs as well as the unit's current position, because units can be placed on roofs without the ability to climb down and therefore must move around on the roof). How any non-player units choose to move is explained in 3.7.

3.7 Artificial Intelligence

The nature of our game, a 2D, turn-based game, allowed for a significantly simpler AI system that what is normally found in game today while still providing a lot of detail and nuanced actions for the enemy units to take.

At the highest level, there is a simple store of information that all AI units can access which allows them to coordinate between each other. Anything that an AI unit would conceivable share with its allies can be stored there for easy access by other units. For example, when an AI spots a player, which is to say that its targeting routine found a target), then the unit registers that sighting with the information store. When another unit's AI is run, it may choose to look for players that were spotted by another AI, in which case it will access the store and try to find a target from the list of nearby player sightings. With information like this stored in an accessible location, the AI can seem more knowledgeable about its environment without introducing a lot of complexity to limit what should and should not be known about the state of the game. This helps to create a more challenging and realistic AI for the player to face.

Each enemy unit contains a script that performs high level duties of AI and wells as an unspecified number of scripts which perform the actual work and are added at runtime based on the text file describing the AI that is assigned to the unit. An example of such a text file can be found in Appendix A. When it is time for an AI to take its turn, this script is called to perform the turn as a whole. It identifies the first package that can be run by having the package evaluate its conditions. Once a valid package has been found, it instructs the package to perform its defined actions. When all the actions have been completed, the unit's turn is over and control returns to the Combat Manager which either passes control to another AI unit or switches to the player's turn if all AIs have completed their turns.

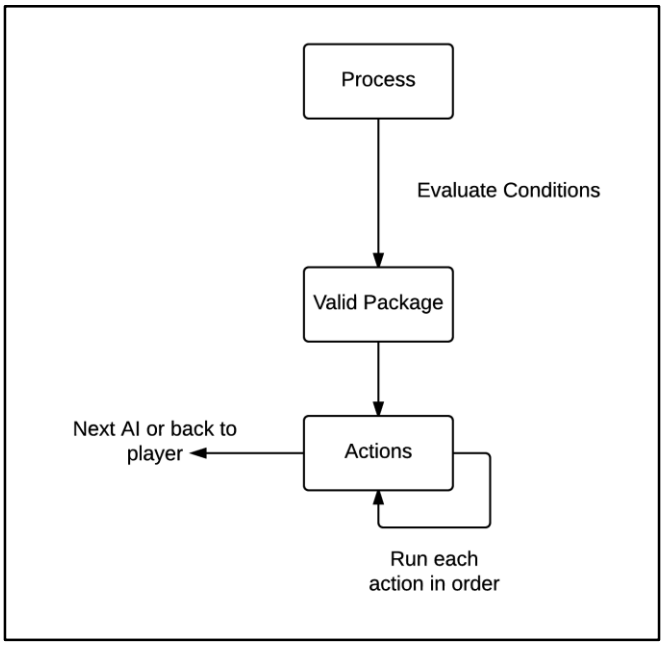


Figure 20: AI

We used such a simple AI system because it was significantly easier to create and maintain than more traditional systems such as hierarchical state machines, decision trees or behavior trees while still allowing us all the freedom to create whatever AI behaviors we wanted. It also let the designer easily understand what an AI would do by reading the corresponding text file. When creating the text file for the AI, what was written was almost exactly what would happen, with very little in the way of abstraction. This meant that the designer could create AI behaviors with little understanding of the underlying implementation.

3.7.1 Packages

Anything that the AI can do is contained with a package. An enemy can have any number of packages attached to it. Each package contains any number of conditions and actions (which will be explained below). In order for a package to be run, all of its conditions must pass. Once a package is found that can run its actions are executed in order. Packages are stored on a unit in a defined order and when finding a package to run, they are checked in that order. All units are automatically given a default package that can always run (has no conditions) and does nothing (has only the NoOp action). This package is inserted at the end of the list of packages so that it will always be the last action to run. An example of an AI file is shown below. For a more comprehensive example, see Appendix A.

```
AI
    Range 3
    Name BasicEnemyR4
    Package
        Conditions
                PlayersInArea 1
        Actions
                GetNearest
                GetLowestHealth
                GetRandom
                GetDestination
                MoveTo
                Attack
```

Figure 21: Text File AI

3.7.2 Conditions

Each package can only be run by having all of its conditions evaluate to true. Each condition checks something about the state of the game and most package contain a number of conditions to allow for more finely control package execution. Many conditions have parameters specifying minimum numbers or percentages that allow the conditions to be variable. Below is a list of conditions that a package might be predicated upon.

InjuredAllies [perc] [num]: Checks if there are [num] number of allies that are injured with [perc] health or less.
InjuredPlayers [perc] [num]: Checks if there are [num] number of players that are injured with [perc] health or less.
Injured [perc]: Checks if the unit has [perc] health or less.
AlliesInArea [num1] [num2]: Checks if there are at least [num2] number of allies within a range of [num1].
PlayersInArea [num1] [num2]: Checks if there are at least [num2] number of players within a range of [num1].
HasPreviousTarget: Checks if the unit had a target during its last turn.
HasPlayerSighting: Checks if there are any units that saw a player within a certain distance of this unit. The distance is specified on a per map basis.
Random [perc]: Checks against a random number. Passes if the random number is less than or equal to [perc].
OnBarrier: Checks if the unit is on a barrier.
OnRubble: Checks if the unit is on rubble.
NearTox: Checks if the unit is near a tox pool. That means that there is at least one tox pool in either of the cardinal directions of the unit's location.
BeneficialPosition: Checks if the unit is in a beneficial position. This is a combination of *OnBarrier*, *OnRubble* and *NearTox*.

Figure 22: AI Conditions

3.7.3 Actions

Everything that the AI might want to do, from identifying a player to attack, moving, attacking, or patrolling around the map, is stored as an Action. Most actions are a single, isolated action and complicated behaviors arise by combining a series of actions in a defined order. The only actions that work together are the ones used to pick targets. The first action executed identifies a list of potential targets based on its specific criteria. All subsequent actions use that list to narrow down

the choice of possible targets until one target is left. Below is a list of all actions that can be used by an AI.

GetNearest: Narrows the possible target by which player is closest to the unit.
GetFarthest: Narrows the possible target by which player is farthest from the unit.
GetWeakest: Narrows the possible target by which player has the least strength.
GetLowestHealth: Narrows the possible target by which player has the lowest health.
GetLowestDefense: Narrows the possible target by which player has the lowest defense.
GetStrongest: Narrows the possible target by which player has the highest strength.
GetHighestHealth: Narrows the possible target by which player has the highest health.
GetHighestDefense: Narrows the possible target by which player has the highest defense.
GetRandom: Picks a random target from the current potential target list.
GetPreviousTarget: Uses the previous target as the new target.
GetSighting: Picks target from the nearby player sightings.
MoveTo: Moves to the currently selected destination.
Attack: Attempts to attack the current target.
GetDestination: Gets a valid destination from the current target.
Patrol: Picks a destination from the unit's list of patrol points.
NoOp: Performs no action.

Figure 23: AI Actions

3.8 Input

Unity provides a decent system to access mouse and keyboard input but it did not fit particularly well with the way our code was structured. Two important things that needed be simple, but which Unity did not fully support were:

- The ability to remap controls
- The ability to disable input

To deal with that we added a layer on top of Unity's input system. It has a list of key mappings from keys (including mouse buttons) to actions that the player might make. This allows the potential to remap controls. The input manager polls Unity's system and looks for the state of any keys that are currently mapped to a control and store it. Any system that needs input registers a callback with the input manager based on the type of input required, which is broken down in controls for Combat, Cutscenes, UI, and Camera. Having the separate types of input allows blocking of certain types of input without blocking all input. It is further broken down into callbacks for KeyDown, KeyUp and KeyHeld. Whenever a key state matches a registered callback, the callback is called with the state of input.

3.9 GUI

The main goal of the GUI was to be clean and simple. Giving the player easy access to the information they needed without having it get in the way of them playing the game. We used NGUI for or GUI needs because it was a superior GUI system than the native Unity one. The most important information to the player is their own unit information and the enemy unit information. Knowing this we had it so the currently selected ally units information was always displayed in the top left, shifting to the bottom left only if the player moved their cursor to that area. This was to avoid the GUI covering up part of the map the player needed to see. Next we made it so that whenever the player hovered over an enemy, the enemy's statistical information would appear next to them. Disappearing again when the player moved the mouse away. The purpose of this was to keep the screen clear of unnecessary clutter and distractions, making it easier for the player to focus on the strategic elements of the game rather than comprehend the interface.

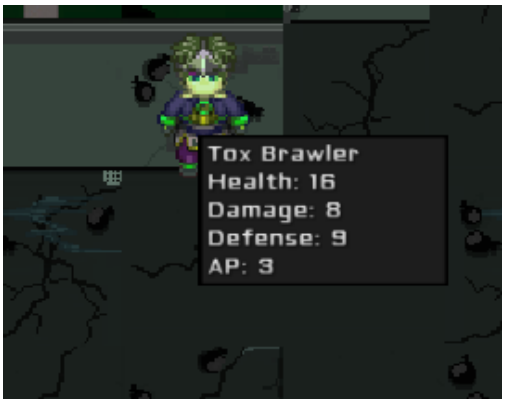


Figure 24: UI

Another important focus for our GUI was to make it properly adjust with different resolutions so there would be no issues with smaller or larger screens. Using a combination of NGUI's built adjustment functionally and our knowledge of adjusting in game elements based on screen resolutions, we made it so that the GUI elements always changed appropriately so that the player would never have an issue with it.

3.10 Audio

Audio was handled by using two different sources for the sound to come from. We used one source for the background music and/or ambient noise, and the other source to handle all sound effects. The main purpose of this system was so that the two types of sounds never interfered with each other, causing the other one to stop playing momentarily.

Background sounds were assigned in the Unity editor after creating a cutscene or battle scene. During combat scenes, sounds are tied to actions and the combat manager handles sending the appropriate audio to the sound manager whenever audio is needed. In cutscenes, audio cues are added to the script of the scene and the cutscene manager uses those cues to play audio directly. The audio will be more heavily covered in Chapter 6.

3.11 Saving

The master controller is responsible for performing saving and loading. Autosaves take place between every scene. The master controller performs a rolling save with five possible save files, overwriting the oldest autosave with the new one. Autosave files are saved under the file name "autosave_N.dhsav" where N is a number from 1 to 5. The user can also save manually. In that case, the save file is named after the level that it was saved on. Loading takes place from the main menu. Players only have the ability to load the most recent save so they cannot abuse the saving system to easily win levels. Saving was an important feature to add because players expect the ability to quit the game without have to completely restart all over again. An example of the save file format can be seen below.

```
save
scene 0-4 Faryll's Plan
player turn True
player index 0
experience
objects
1:(-20.0, -50.0, -5.0):11:2:2:3:0
0:(20.0, -50.0, -5.0):17:4:4:3:0
5:(20.0, 30.0, -5.0):10:3:5:3:0
3:(-20.0, 30.0, -5.0):10:3:5:3:0
2:(-30.0, 40.0, -5.0):10:3:5:3:0
4:(0.0, 40.0, -5.0):12:5:7:3:0
6:(30.0, 40.0, -5.0):10:3:5:3:0
tutorial
Stats:True
Movement:True
Attack:True
SwitchPlayers:True
EndTurn:True
Camera:True
AdvanceDialogue:True
SkipCutscene:True
```

Figure 25: Text File Save

3.12 Conclusion

We believe we accomplished our major goals which were to design modular code and to create development tools for the designer to make it easy to iterate on the game. Going forward, this code base and set of tools could be used to make new features and quickly implement them into the game without have to waste valuable time.

Chapter 4 Art Development

The art in *Dark Horse* is created for both the cutscenes and tile maps. The overall focus for this aspect of the game is:

- To provide visuals that can drive the story forward.
- To create a captivating world that the player can experience.

These were accomplished by utilizing a color theme throughout the backdrops and spritesheets of the game.

4.1 Color Theme

One major way we wanted to create an immersive world, with regards to art, is to introduce a color theme for each level. For example, the world map displayed on Figure 26 shows that each area is defined by shades of red, yellow, blue, green, purple, and yellow/white. By associating each section with a certain hue, it makes it easier for the player to realize where he is in the game.

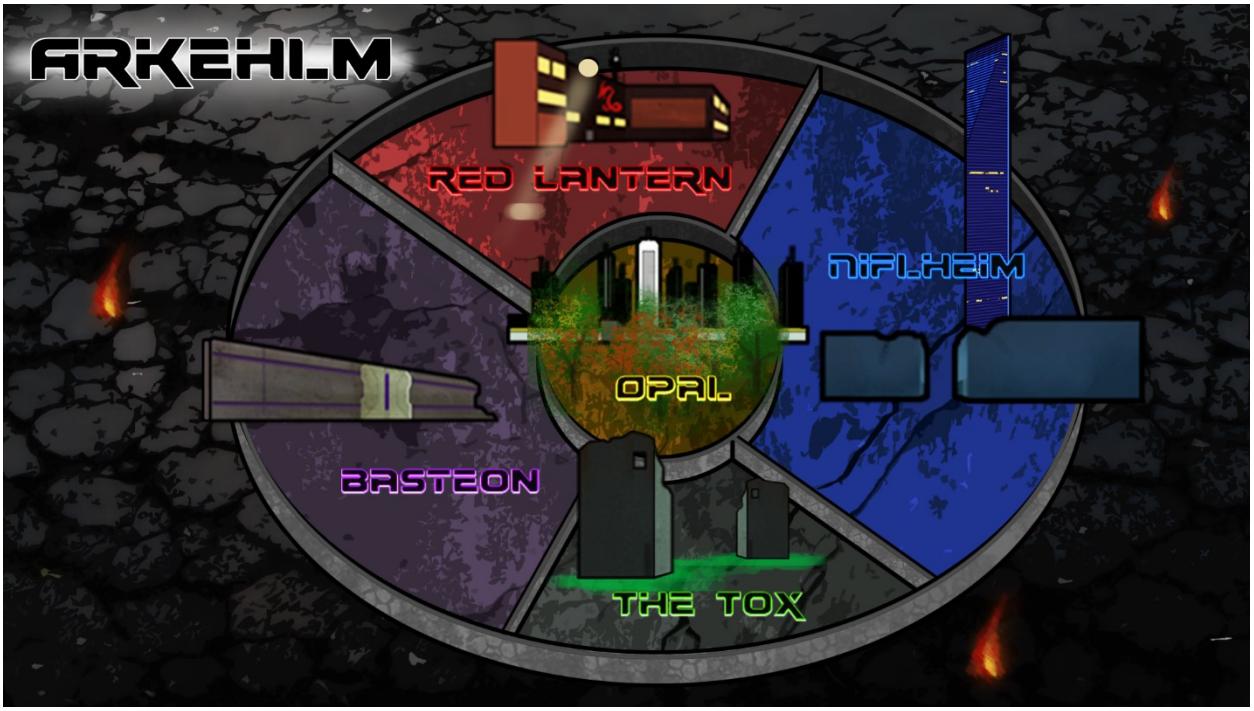


Figure 26: World Map

4.2 Cutscene Backdrops

Each cutscene backdrop was meant to set the mood for the game. The world in *Dark Horse* consists of five sectors. Each has its own underlying color theme, giving them their own feature and setting the tone for the scene.

The backdrops are drawn in a 2D art style in Photoshop. Using existing reference images and a perspective grid guideline, we made it so that each artwork stayed consistent with the appropriate color. Also, in order to add a touch of realism, we used texture images to blend into certain parts of the scene. Once the backgrounds were completed, they were saved as a high quality image file, and imported into the game engine.

Using The Tox as an example, we incorporated darker colors with less saturation in our artwork. By doing so, each cutscene was created in a way where the player can appreciate the artwork, but not enough to detract from the characters and their interactions.

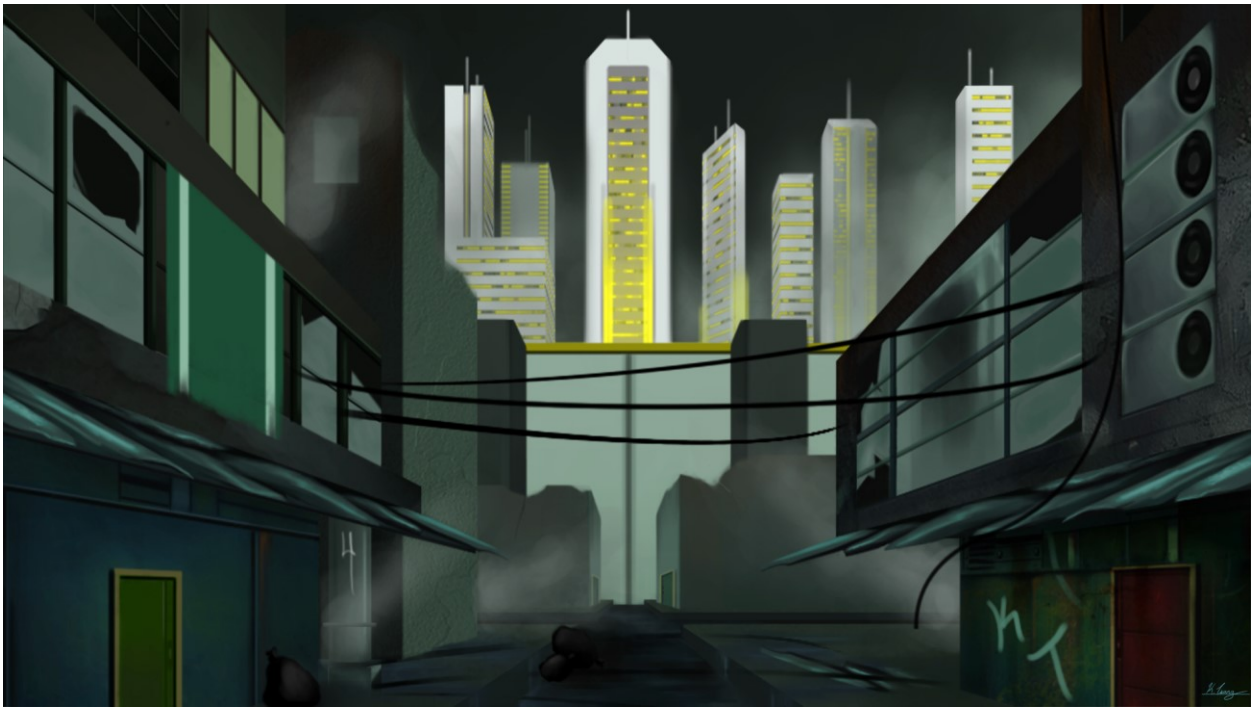


Figure 27: Tox Background

4.2 Sprite Maps

The game maps were drawn in a pixelated style as an homage to games, such as *Advanced Wars*, *Fire Emblem*, and *Final Fantasy Tactics*. The image below, Figure 28, was one of the references we used in order to get a better picture of what we wanted for the sprite maps.

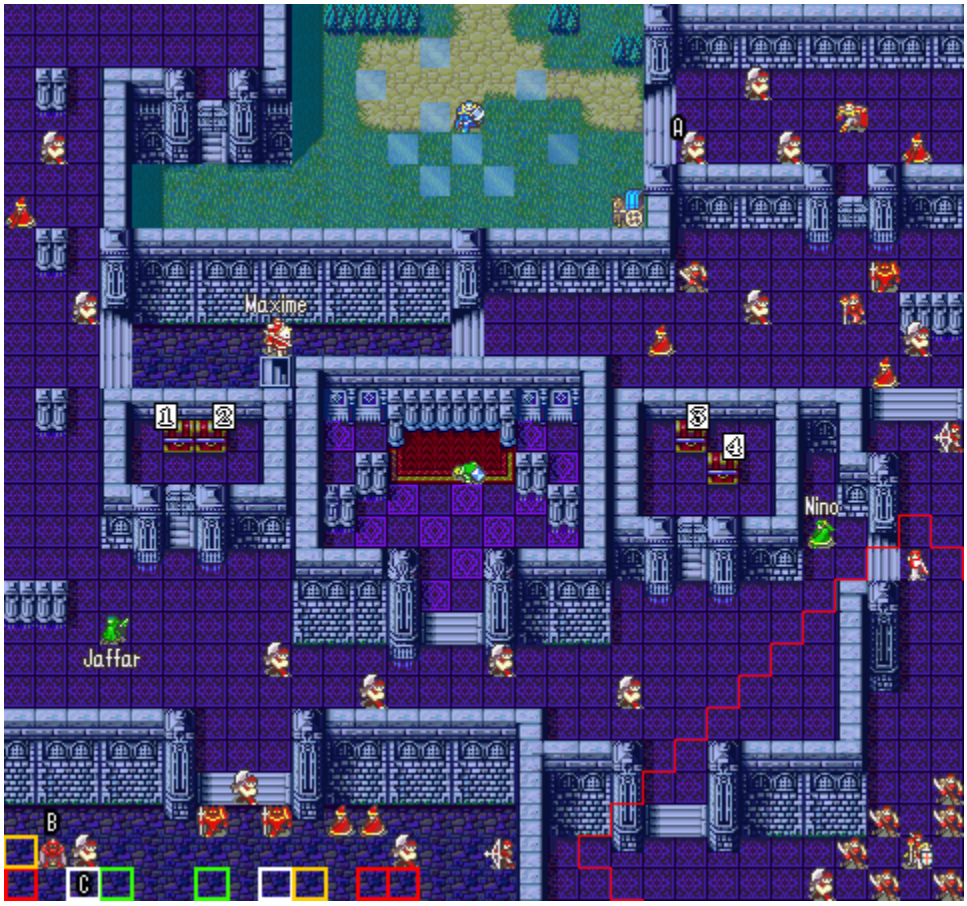


Figure 28: Fire Emblem

The guideline for the spritesheets that we had to follow was that they had to be consistent with the backdrops, which included possessing the same color theme and tone. Additionally, using a top-down perspective, every terrain consists of 64 x 64 pixel tiles, meaning that objects that are bigger than the measurement had to be split up into multiple pieces.

As each tile piece was completed in the spritesheet, it was placed on a mock battle map in order to get a general idea of how the level looks. There were many instances where we had to go back and rework a few tiles because the pieces did not go well with each other. Some needed to be matched better with its adjacent tiles. Others needed to be messier, meaning more detail or objects needed to be added to remove that feeling of cleanliness.

Creating spritesheets within Photoshop is beneficial in a few ways:

- One reason is because of convenience with regards to art. For example, consider that a floor tile on the map in Figure 29 had a few mistakes. All we need to do is go back to the

spritesheet and correct that specific tile. In turn, these updates will automatically be shown on the map.

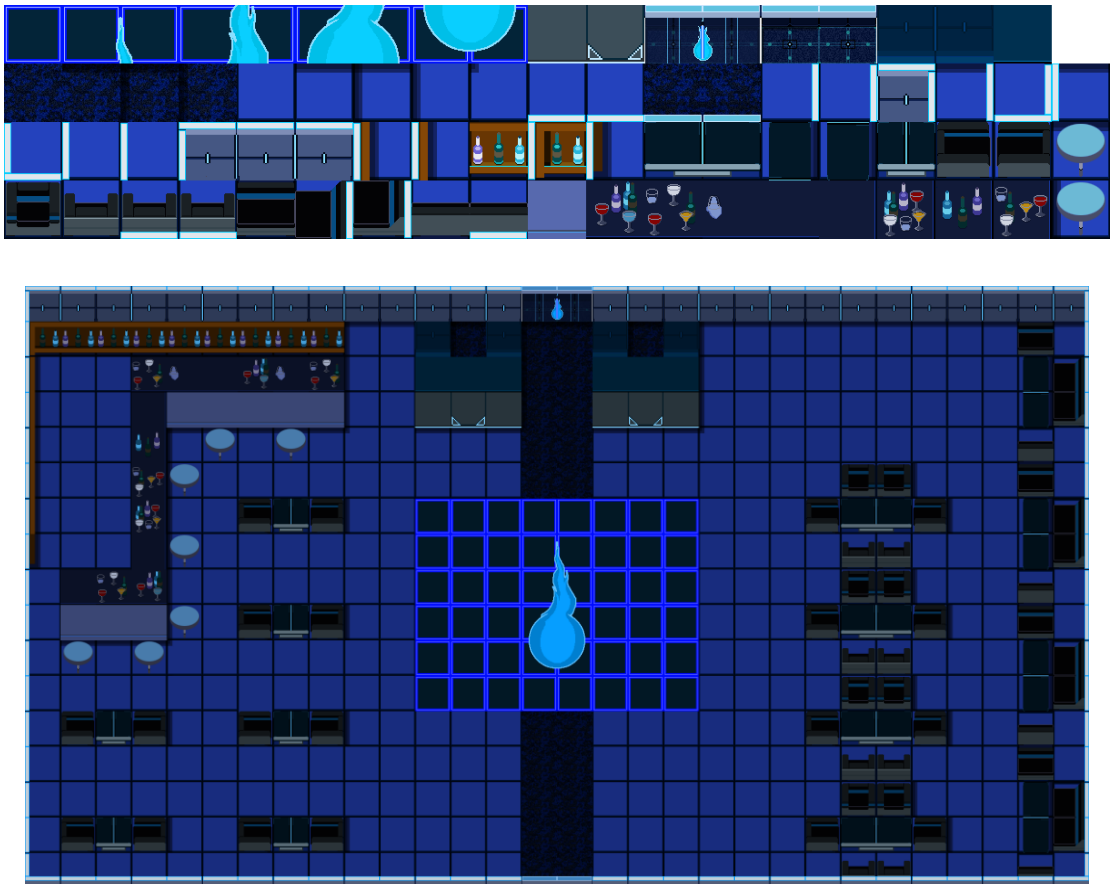


Figure 29: Hase Lounge Spritesheet and Map

- The other reason is because this method provides much more options in creating the look of the map. Each tile can be assigned to different places, so that, when it is done, we can see which one best represents the level.

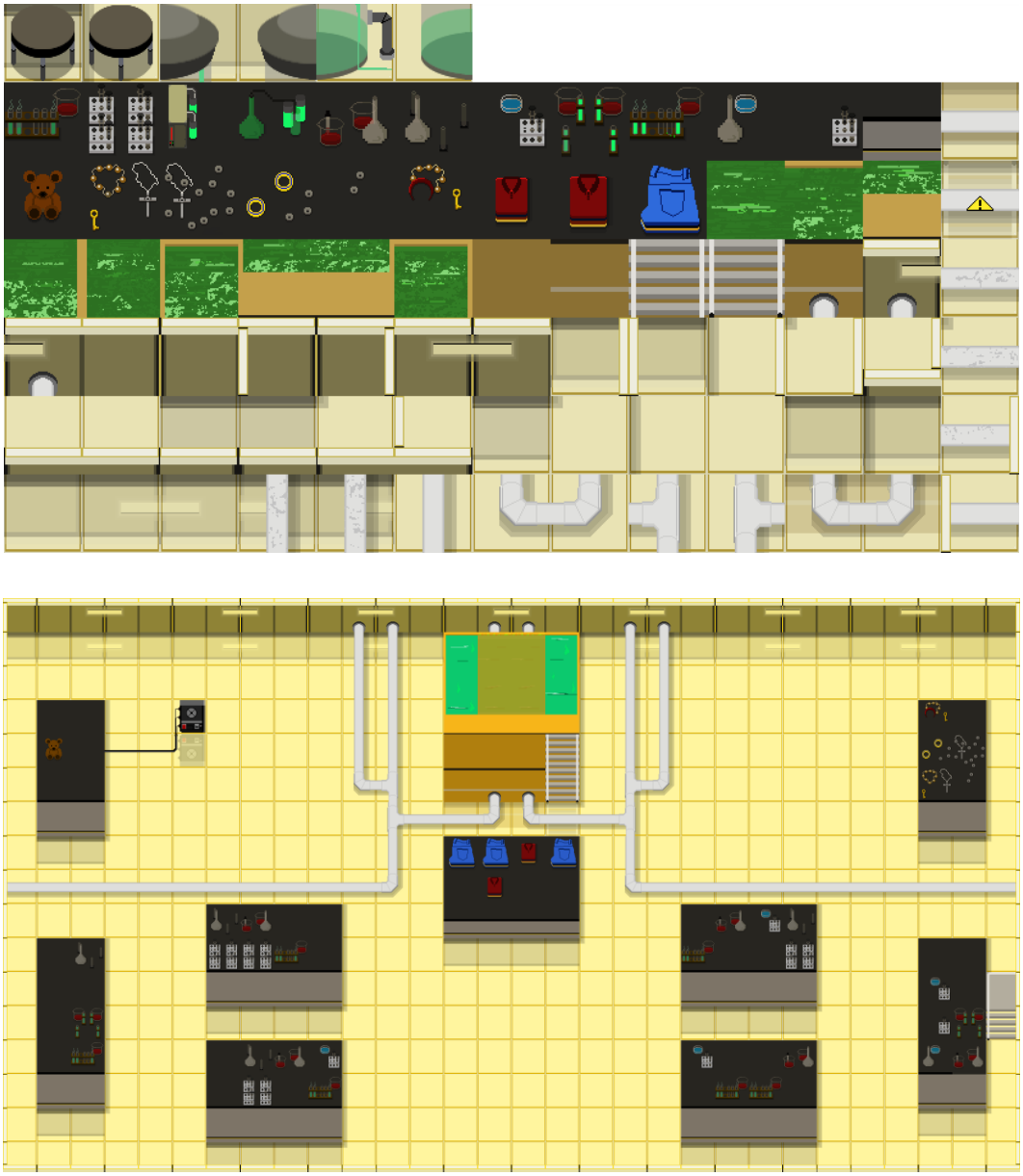


Figure 30: Vrell's Lab Spritesheet and Map

4.3 Conclusion

Overall, we felt that we successfully created cohesive and interesting art that flows well with the narrative and helps guide the player around the world. An important aspect across all of the art work was making sure the players could identify where they were and follow along with the story because of it.

Chapter 5: Audio Development

Audio is a powerful and important factor in the development process because it adds an additional dimension to the overall game. The goal of *Dark Horse*, with regards to sound, is:

- To supplement the visual aspects of the game
- To evoke certain emotional reactions from the player, such as tension during battle scenes and nostalgia in cutscenes.
- To provide a better fit for the overall theme of the game.

5.1 Sound Effects

The game contains sound effects that one would normally hear in conventional combat games. The reason being because we wanted the game to have those types of timbres that most players have heard already. So, in addition to the visual aspects of the game, the sound effects provide cues as to what is currently happening.

The actual recording process was a case of trial and error. Because the rooms being recorded are not acoustically treated, we had to place the microphone in different positions and numerous adjustments had to be made in order to get a clear and identifiable sound. In addition, live props, such as kitchen knives and fruit, were used and recorded multiple times in order to save the trouble of re-recording.

Sound Effects	Description
Ambient	Background sounds for the environments. Recorded with a portable microphone.
Energy Discharge	Fired from a plasma gun. Used existing sound as a base. Layered with additional effects in Ableton
Explosion	Created in Audacity and Ableton using sound filters and effects.
Footsteps	Recorded in various buildings at WPI using a portable microphone.
Gunshots (Rifle/Sidearm)	Used stock audio for the base foundation. Layered with additional effects in Ableton.
Knife Stab	Recorded a pomegranate being squeezed using a USB microphone. Added stock audio for extra layering and polish.
Slashes (Plasma/Sword)	Used tape for the electrical effects in the plasma slash. Used kitchen knives for the sword slash. Recorded using a USB microphone.

Figure 31: Table of Sounds

The final results of the recordings were further enhanced in Ableton Live. Overall quality was improved by utilizing different audio techniques, such as cutting out unwanted noises, raising the volume to appropriate levels, and equalizing to block out certain frequencies.

In conclusion, we wanted to provide the player with that sense of realism by having sound effects supplement the motions of the characters.

5.2 Original Compositions

The main goal of the compositions is to provide a fuller immersion into the game, meaning the player should be fully absorbed into the gameplay experience and feel a certain way as the story progresses. For example, battle scenes should convey a sense of thrill and cutscene dialogues should elicit a certain attachment to the characters.

In Sibelius, we chose one of the many orchestration templates that was available in order to save the trouble of manually selecting individual instruments to add to the piece. Using a MIDI keyboard, we decided that the first thing that needed to be done was to create the melody because it makes up the primary component of the piece. Additional instruments were sequentially incorporated to add even more notes, with each volume being adjusted accordingly. Further tweaking was done in Ableton Live in order to get a final polish.

The whole process creates a sense of balance and harmony to the overall structure.

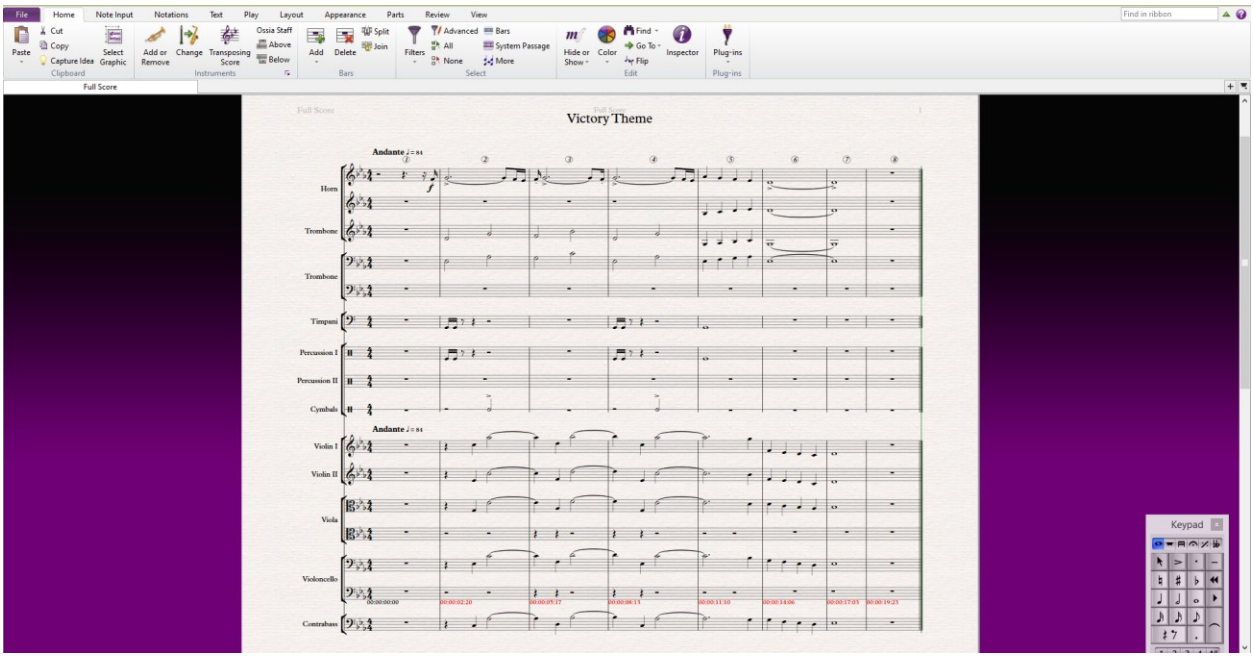


Figure 32: Composing in Sibelius

5.3 Conclusion

Audio is often overlooked in the game development process. We wanted to make our own audio so that we could better craft the overall look and feel of the game. In the future, we would like to expand on this by including better quality sounds and instruments in order to create an even more compelling world.

Chapter 6 Narrative

The narrative of our game was an attempt to create a story within a strategy game that had not only a darker plot central to the gameplay, but also a sci-fi setting. Looking at other games in the genre, there is a noticeable lack of both of these features. Fire Emblem games have plots central to gameplay, but they tend to be much happier and brighter toned stories and are set in fantasy universes rather than sci-fi. X-COM has a sci-fi setting with a dark plot, but its story isn't that heavily connected to gameplay. This showed us that there was an opportunity to create a game that stands apart from other strategy games not through its mechanics, but its narrative, and that is what we sought to do with Dark Horse.

6.1 Story Outline

Our story follows a man named Jered, a noble living in the corrupt city of Arkehlm and wishes to see it changed for the better. After many years of preparation, Jered frees his childhood friend Faryll from slavery in order to stage a revolt to take control of a city sector. The revolt succeeds, and together Jered and Faryll create their own "gang" to solidify a foothold of power in order to defeat the other gangs controlling the rest of the city and restore their world to peace. Their road to peace leads through sinister territory, however, and as they progress further towards their goal it becomes clear that even the righteous will have to stain their hands with blood by the end. A full synopsis can be read in Appendix D.5.

6.2 Designing the Story

In order to write a good story, we decided to develop each aspect of it piece by piece. These aspects were:

- Characters
- World
- Events

Each different aspect was handled in a separate design document, much like our mechanics were.

The characters design document focused solely on developing the major and minor actors in our game. This document was where we began the story design and the characters evolved over time from simplistic stereotypes to complicated people with relationships and backstories.

Main Characters:

Name:
Faryll

Bio:
Raised in the Red Lantern, he was sold into slavery for refusing to augment himself into a Drake when he came of age. Faryll used his situation to plan a slave revoltin order to overthrow the Boulderfists of Basteon. Working with his childhood friend Jered, a Boulderfist himself, and relying on the brutish strength of his fellow slave Wenden, Faryll was able to secure transport into Basteon and burn the old regime from the inside out. From the ashes of this uprising, Faryll, Serithe, and Wenden raised up a new power, which they named the Dregs, to run Basteon. A benevolent crime lord, Faryll turns none away, for that is the promise of the Dregs. Even a kinder crime lord is still a crime lord, though, and Faryll will do what he must to ensure the power of his syndicate.

Name:
Wenden

Bio:
By Faryll's side for most of the slave revolt, Wenden is an experienced brawler who harbors a burning hatred for the Drakes of the Red Lantern, who were responsible for slaughtering her family and giving her horrible burns on her hands and arms. Despite her injuries, or perhaps because of them, Wenden brutalizes her enemies with scarred fists, utilizing the dead nerves to relentlessly hasten her opponent's death with each strike.

Name:
Jered Boulderfist

Bio:
A childhood friend of Faryll's, Serithe has always harbored a hatred for the Drakes and the Red Lantern after they sold Faryll into slavery. Serithe would often visit his friend when accompanying his father into the Lantern. It was during one of these visits when they resolved to end Faryll's and all slaves' imprisonment. From this resolution, the Dregs were born. Jered trains to become a formidable sniper Years afterward, after his father's passing, Jered finally gains the power as begin putting his plan into action.

Figure 33: Document Detailing Main Characters

The world design document was only concerned with creating the city of Arkehlm and its various different sectors. These sectors are as follows:

- Basteon
- The Red Lantern

- Niflheim
- The Tox
- Opal

With each sector, we wanted to create a totally different environment and culture from the last, showing the disjointed nature of each sector even though they all live in the same city. Basteon has a purple color theme and focuses its architecture on mimicking the marble statues and halls of ancient Greece, a landmark of nobility. The Red Lantern worships the image of the dragon and uses lots of reds in its color schemes. Niflheim value the flame of knowledge and have a variety of blues in their palette. The Tox is an abandoned sector defiled by poison and disease; it uses the color green primarily. Opal is the golden center of Arkehlm and the last holdout of the government in the city. The design document responsible for detailing the narrative context of the sectors can be seen below and also in Appendix D.5.

World:

In the city of Arkehlm, a jungle of chrome and concrete, you are either found dead in a ditch or taken in by one of the many criminal gangs vying for power. What stands for the "authorities" have been helpless to stem the tide of blood and crime flooding out of Arkehlm's different sectors for years now and the city has fallen to ruins.

Opal is the central government sector and formerly the seat of power in Arkehlm. Twenty-seven years ago, the governor of Arkehlm recalled every government official and employee inside the walls of Opal and sealed its gates. Surrounding Opal is an area known as the Skids, split unevenly into four different sectors to the north, south, east, and west.

People call the northern sector the Red Lantern, for its notorious human trafficking and prostitution rings. The freakish Drakes have control here.

The Desolates inhabit the southern sector, named the Tox. The sector was previously named Opalhand, so called for the Loyals, a gang created by the governor himself which adhered to his command. The Loyals were wiped out, however, by a terrible epidemic in Opalhand years ago that killed most of the sector's inhabitants and mutated the rest.

In the east sector, Niflheim, the gang Azure calls the shots. They like to style themselves as mystics and sages, but in truth they simply get high on Hase, a drug that boosts the user's mental agility temporarily but leaves them nearly braindead when in withdrawal.

The western sector is called Basteon and is run by a relatively new gang called the Dregs. Formed from the rejects of the other sectors, the Dregs are a varied gang consisting of all kinds of people, from nobles to freed slaves. With the leadership of the gang's founder though, the Dregs have managed to carve a name for themselves as the best infiltrators, thieves, and assassins Arkehlm has ever seen. Our story begins with the beginnings of the Dregs, back when the Boulderfists still ran Basteon.

Figure 34: Document Detailing the World

The events of our game were described in the story synopsis design document. It detailed the overall narrative arc of our game, including when new characters were introduced into the story as well as what important events happened during cutscenes.

Prologue-

Characters Introduced:

Faryll
Wenden
Jered
Governor Vrell

Faryll and Wenden (a slave) arrive in Basteon at Jered's compound. Jered and Faryll reveal their friendship. Faryll and Jered go to investigate an explosion .

The two arrive in the holding cells to find a mysterious cloaked figure (actually Governor Vrell). After a brief exchange, Faryll launches an attack but is defeated, allowing his foe to escape. Despite being wounded, Faryll still manages to fightthe escaping prisoners with Jered's help (**Player Controlled Combat, PCC for short)**.

After the battle, Faryll and Jered discuss the plan to overthrow the Boulderfists' rule of Basteon using a massive slave revolt.

The revolt is bloody, and many die on both sides, but in the end the sheer physical strength of the slaves combined with Jered's resources allowed Faryll and his fighting force to wrest control of Basteon from the Boulderfists.

Together Faryll and Jered decide to take control as a new gang devoted to taking control of all Arkehlm's sectors in order to finally bring about an end to the endless conflicts between the gangs.

Chapter 1-

Characters introduced:

Fhec the Fleet

Fast forward 5 years after the revolt to the new Basteon, under the leadership of Jered and Faryll who have formed a gang called the Dregs. Faryll deals with espionage and combat issues, while Jered works to restore some semblance of order and politics to Basteon. After these years of stabilizing Basteon, Faryll now believes their forces are ready to take out the Drakes and gain control over the Lantern.

Under the front of a territory war, the Dregs launch a surprise attack against the Drakes, plunging the two gangs into brutal skirmishes all along the western border of Basteon. During the

Figure 35: Document Detailing the Story

6.3 Conclusion

While we were able to draft a full story synopsis, the full script that would be used by our game proved to be time consuming to write and even more time consuming to implement in our cutscenes. As a result, our goal to create a strongly narratively driven game was not fully reached. We learned that simply writing a good story was not enough to develop a story driven game. The narrative had to play a hand in molding many areas of development, and this connection worked both ways as the narrative also changed to reflect our progress with different game features. This realization improved our design process, but came late into development after a great deal of time and miscommunication had passed. If there was more time to develop, we would be able to implement the full story into our game.

Chapter 7: Post-Mortem

During the course of this year long project, there were many elements of the project that went right, and many that went wrong. In this section we will be talking about both the good and bad and how we overcame the issues as a team.

7.1 What Went Wrong

7.1.1 Scope Issues

Issues with the scope began during the concept phase before the game idea was fully decided on. We planned out a huge turn based adventure with a lot of story and tons of levels. We did realize this was a lot and began to cut it down, but what we didn't realize was that the real issue would be the art requirements. While we began to cut down the story and amount of levels and characters, we didn't take into account how much different art would be needed to make each level and the various characters in game. Not only this, but the cutscenes would require their own amount of art as well. To make matters worse, we planned it in a way that most art could only be used in one level or cutscene, having little crossover with the rest of the game.

It wasn't until two terms into the project that we actually faced and dealt with the scoping issue. To manage it we came up with the core amount of levels, cutscenes, and characters that would be needed to make the minimum viable product. This allowed us to focus in on the portions of the game that had to be there instead of waste time making nonessential features.

7.1.2 Art Pipeline Issues

During initial development, the art pipeline would often find itself clogged. We realized that the root of this problem was a matter of miscommunication and stemmed from a team member who vocalized neither his process nor his progress. While the rest of us were willing to actively participate and communicate, the team member often would not do the same. We tried to work around this problem instead of dealing with it head on, which turned out to be a mistake and ended up setting our project back significantly. We eventually had no other option than to remove the team member from our group due to his corrosive work ethic that threatened to damage our project even further.

While the removal of the group member expedited our art production process, we still had problems simply due to the sheer amount of art assets needed. Since we had only one artist in the group, and as the removal happened late into the year, we decided to cut many planned assets as

well as seek outside help from other artists in order to meet the necessary requirements for our base viable product.

While the problems could have been less dramatic if we had developed a solution quicker, we feel we were able to solve our issues adequately by revising workloads and enlisting assistance.

7.1.3 Design Coordination Issues

Mechanics and game content were often designed independently from the art and programming team, and while they took into account the limitations each team had to an extent, it was far from a comprehensive understanding of what could and couldn't be done with the art and code. As a result, some mechanics had to be either redesigned or scrapped entirely as the time that it would take to design these features would outweigh their usefulness in the larger scope of the game. For example, a certain level that revolved around a closed door system had to be redesigned completely due to the amount of work it would have taken to develop the system within our existing code framework. These issues not only led to decreased productivity but also wasted valuable time in a project that had already encountered large setbacks.

Luckily, we became aware of this inefficient process early enough where we could revise our communication so that everyone had a good idea of what everyone else's capabilities were.

7.2 What Went Right

7.2.1 Technical Success

The goal of the technical team was to create a clean, modular code base that allowed us to efficiently make any changes or additions that we needed. We definitely succeeded in accomplishing this goal. Our code was clean and understandable and adding features rarely caused any issues with other areas of the game.

A common feature addition was adding new types of status effects to our map. In a poorly architected code base, these additional might have been messy. However, because of the way that we structured our code adding new a status effect was very straight forward. It was as simple as adding a few lines of code to make sure that our level creation code could understand this new status effect and then any code necessary to make the effect itself happen which was also easily added.

Another big success from a technical standpoint was the use of text files to handle most of our game's information. Having much of our information in text files was advantageous as it allowed us to quickly create and iterate on design ideas. Levels could be tweaked in seconds instead of

minutes and we could create new attacks, unit class or AI routines by copying an old text file and making a few changes. The text files helped to greatly speed up development and allowed us to include more content that we would otherwise have been able to create.

Overall, we were able to create a code base that allowed us to quickly add any features that our game required and did not get in the way of creating the game that we wanted.

7.2.2 Audio Success

The focus of the audio was to create background music and sound effects that emotionally conveyed the theme of the game. That goal was achieved. We felt that there was a solid amount of sound effects and music with which to incorporate into the game.

Development of the sound effects went very well. Aside from finding the right props with which to use in the recording, there was not really anything else that was an issue and, because of that, sound effects were able to be generated fairly quickly.

Composing the background music was also very convenient because of the notation software that was used. The only major challenge was to create a piece that would appropriately capture the mood of the scene. Even so, the program provided enough user-friendly tools to make quick changes. After a few instrument changes and more note adjustments, we were able to incorporate music that were suitable to the game.

The audio development process was a significant achievement because they are all original. That endeavor was necessary in order to fit the overall theme of the game.

7.3 What We Would Do Next Time

The first thing that we did late in the project that we should have done very early, was deciding on what the minimum viable product was, and also taking steps to making it first. By focusing on the bare minimums for the game we were able to clearly see how much work would be needed to make various part of the game. From there we could decide how much more we could put into the game and also better plan out our development process down the line. Once the MVP was completed it was much easier to proceed from there.

Another big thing we wish we did early on was deal with poisonous and unproductive people. By putting off dealing with these problems for months, all we did was get really far behind and hurt the entire team. Facing the problem early would have let us quickly readjust our design and get back to work.

Chapter 8: Bibliography

Abletondaily. "#24 Ambient Sounds with Vocoder :: Ableton Live." YouTube. YouTube, 10 Nov. 2010. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=0CVmwv6z8g0>.

"AsphaltDamaged0047." AsphaltDamaged0047. Web. 28 Apr. 2015. <http://cgtextures.com/texview.php?id=25331&PHPSESSID=h7bv0vg8h2ofln165s9i8icmd1>.

Basementhum. "Creating Explosion Sounds for Ibb&obb." YouTube. YouTube, 14 Feb. 2012. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=N7Hvdy39Cfc>.

BoxOfficeSmashers. "Inception - Official Trailer [HD]." YouTube. YouTube, 12 Jan. 2010. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=8hP9D6kZseM>.

Cameron, Stephen. "Tutorial on Making Explosion Noises in Audacity." YouTube. YouTube, 21 July 2012. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=HRypuCJH9OI>.

Chagas, Filipe. "Lemon,Juicy,Squeeze,Fruit.wav by Filipe Chagas." Freesound. N.p., 10 Mar. 2010. Web. 27 Apr. 2015. <http://www.freesound.org/people/Filipe%20Chagas/sounds/91915/>.

"Cool Futuristic Door Design Ideas." LARCADE. Web. 28 Apr. 2015. <http://www.larcade.net/8525-stunning-design-and-good-functional-of-futuristic-door-design/cool-futuristic-door-design-ideas/>.

Firaxis Games. XCOM: Enemy Unknown. 2K Games. 2012.

FreeSFX. N.p., n.d. Web. 27 Apr. 2015. <http://www.freesfx.co.uk/sfx/firecracker?p=1>.

FreeSFX. N.p., n.d. Web. 27 Apr. 2015. <http://www.freesfx.co.uk/sfx/sizzling>.

FreeSFX. N.p., n.d. Web. 27 Apr. 2015. <http://www.freesfx.co.uk/sfx/whoosh>.

Gabovitch, Iwan. "Swoosh Whoosh Sword Air Swing Sound Effect." YouTube. YouTube, 13 June 2013. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=9eUFdAquFJs>.

"Game Boy Advance - Fire Emblem 8: The Sacred Stones - Euphraim - The Spriters Resource." The Spriters Resource. Web. 28 Apr. 2015. <http://www.spriters-resource.com/game_boy_advance/fe8/sheet/14425/>.

GAppears. "Xenoblade Chronicles Music - Time to Fight!" YouTube. YouTube, 10 June 2010. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=MUKzFYUGbg0>.

"Glossy Emblem Text Effects – Photoshop Tutorial." Hongkiatcom. Web. 28 Apr. 2015. <http://www.hongkiat.com/blog/glossy-emblem-text-effects-photoshop_tutorials/>.

Glue70. "Ableton Live Tutorial - Making a Laser Sound Effect." YouTube. YouTube, 31 July 2014. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=EJUD9zn_h1g>.

HGRRAC. "HD Plasma Gun Effect With Sound." YouTube. YouTube, 24 Apr. 2012. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=myAmiHOyoLo>.

Intelligent Systems. Advance Wars. Nintendo. 2001.

Intelligent Systems. Fire Emblem. Nintendo. 2003.

Jordaan, Cornelis. "Planetary Corona : An Orchestral Sci-fi Theme." YouTube. YouTube, 25 Jan. 2012. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=BhTtQlU1gFI>.

"King Tong Street YAO." DeviantArt. Web. 28 Apr. 2015. <http://ewkn.deviantart.com/art/King-Tong-Street-YAO-209992823>.

Kollar, Philip. "Home Review: There's No Place Like Hell." Polygon. 3 July 2012. Web. 28 Apr. 2015. <http://www.polygon.com/2013/1/24/3673776/home-review-theres-no-place-like-hell>.

Kunstler, Julianna. Web. 28 Apr. 2015. <http://juliannakunstler.com/images_art1/space/city10.jpg>.

Maniotakis, Bryan. "Tutorial - Making an Atmospheric Pad in Ableton Live." YouTube. YouTube, 11 May 2013. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=KINRvl8vwNc>.

"MetalAircraft0050." MetalAircraft0050. Web. 28 Apr. 2015. <http://cgtextures.com/texview.php?id=50013&PHPSESSID=u17g429i39khhc0r8bn8itvji3>.

MinMax Games. Space Pirates and Zombies. 2011.

New World Computing. Heroes of Might and Magic III. The 3DO Company. 1999.

NeoCore Games. King Arthur: the Role-Playing Wargame. Paradox Interactive. 2009.

Noisysid. "Cookin' With Noisy - Eps. #2 - Foley - Create A Knife Stab Sound Effect." YouTube. YouTube, 9 Nov. 2011. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=VVKYTcv3UV4>.

Patel, Amit. "Implementation of A* from Red Blob Games." Implementation of A*. Web. 1 Nov. 2014. <http://www.redblobgames.com/pathfinding/a-star/implementation.html>.

SoundEffectsFactory. "Knife Stab." YouTube. YouTube, 11 Feb. 2012. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=jt-p9sTpS9k>.

Square Enix. Final Fantasy Tactics Advance. Nintendo. 2003.

Thiefje. "Call of Duty Black Ops Victory Theme (HD)." YouTube. YouTube, 11 Apr. 2011. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=8zmtfhvlRd4>.

Trumpeter93. "Amazing Lightsaber Sound Effects Two!!!!!!!!!!!!" YouTube. YouTube, 14 Feb. 2009. Web. 27 Apr. 2015. <https://www.youtube.com/watch?v=8flafzgOrSI&src_vid=IhNx97LYg_8&feature=iv&annotation_id=annotation_113602>.

Unity Answers. Web. <http://answers.unity3d.com/>.

Appendices

Appendix A File Format Examples

A.1 Level

```
Name: Basteon Prison
Size: 13,11

StartMap
2 6 5 1 5 12 15 12 6 3 6 5 4
18 30 29 29 30 29 11 30 29 30 29 29 7
39 26 25 32 25 11 10 11 26 25 31 25 38
16 25 31 25 26 10 11 10 25 31 32 26 42
41 32 11 26 25 31 25 31 26 25 11 25 17
39 10 11 32 26 25 32 25 25 31 10 11 40
37 26 25 25 31 26 25 31 25 25 11 31 17
16 32 25 32 26 25 32 25 32 25 26 25 42
14 13 13 13 24 21 19 20 14 13 13 13 24
14 13 23 13 23 21 19 20 22 13 22 13 24
29 29 30 29 29 30 30 29 30 29 29 30 30
EndMap

StartPlayers
       StartPlayer
           Name: Faryll
           Stats: faryll
           Level: 1
           Attacks: melee
           Sprite: Blue
           Position: (8,0)
       EndPlayer
       StartPlayer
           Name: Jered
           Stats: jered
           Level: 1
           Attacks: ranged
           Sprite: Blue
           Position: (4,0)
       EndPlayer
EndPlayers
```

```
StartEnemies
       StartEnemy
           Name: Prisoner
           Stats: brawler
           Level: 1
           Attacks: melee
           Sprite: Drake_Goon
           AI: BasicEnemyR3
           Position: (3,9)
       EndEnemy
       StartEnemy
           Name: Prisoner
           Stats: brawler
           Level: 1
           Attacks: melee
           Sprite: Drake_Goon
           AI: BasicEnemyR3
           Position: (4,8)
       EndEnemy
EndEnemies

StartTerrain
(0,1)
(1,1)
(2,1)
(4,10)
(8,10)
(9,10)
(10,10)
(11,10)
(12,10)
EndTerrain

StartStatus
Rubble (1,5)
Rubble (7,7)
Rubble (7,8)
EndStatus
```

A.2 Cutscene

```
#Backdrop used: Jered's Compound
Characters

Faryll Right
Wenden Left
Jered Left
Slaver Right

Background Music
Loops/Midnight_Hour_Loop

Scene
[Jered] FadeIn
[Jered] MoveInToLeft
Jered: I can't believe this day is finally here. Don't worry my friend, your dream
shall be realized soon.
[Jered] FadeOut

[Slaver] FadeIn
[Slaver] MoveInToRight
Slaver: Got a shipment for Jered Boulderfist. That you?
[Slaver] FadeOut

== Effects/Plasma Slash

Slave: AAGH!
[Faryll] FadeOut

[Jered] FadeIn
Jered: Quiet down or you'll get more than a plasma lash next time.
[Jered] FadeOut

[Faryll] MoveOutToRight

[Slaver] FadeIn
[Slaver] MoveInToRight
Slaver: I see these troublemakers won't pay you no bother after all. Been a pleasure,
sir.
[Slaver] FadeOut
```

A.3 Save

```
save
scene 0-4 Faryll's Plan
player turn True
player index 0
experience
objects
1:(-20.0, -50.0, -5.0):11:2:2:3:0
0:(20.0, -50.0, -5.0):17:4:4:3:0
5:(20.0, 30.0, -5.0):10:3:5:3:0
3:(-20.0, 30.0, -5.0):10:3:5:3:0
2:(-30.0, 40.0, -5.0):10:3:5:3:0
4:(0.0, 40.0, -5.0):12:5:7:3:0
6:(30.0, 40.0, -5.0):10:3:5:3:0
tutorial
Stats:True
Movement:True
Attack:True
SwitchPlayers:True
EndTurn:True
Camera:True
AdvanceDialogue:True
SkipCutscene:True
```

A.4 Attack

```
Attack
    Name Melee
    MinRange 1
    MaxRange 1
    Damage 0
    Sound Knife Stab
```

A.5 Stats

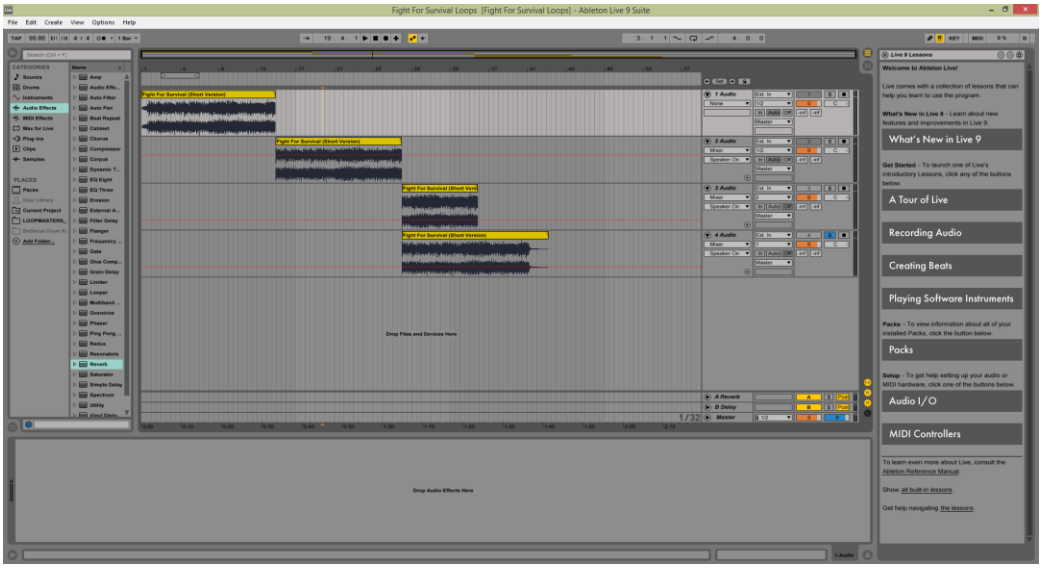
```
Stats
    Name brawler
    Health 8
    ActionPoints 3
    Strength 6
    Defense 1
```

A.6 AI

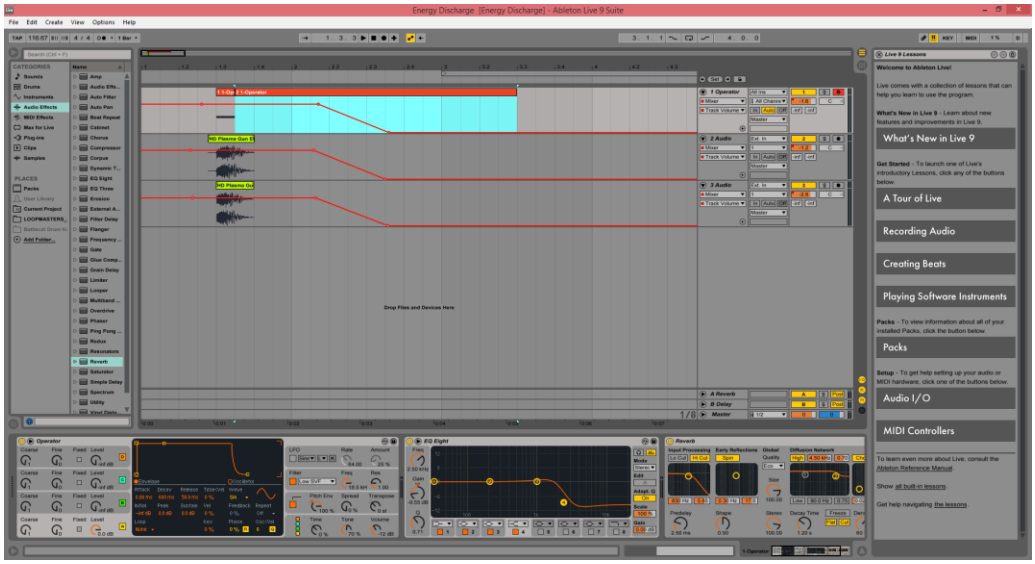
```
AI
    Range 3
    Name BasicEnemyR4
    Package
        Conditions
               PlayersInArea 1
        Actions
               GetNearest
          GetLowestHealth
               GetRandom
               GetDestination
               MoveTo
               Attack
    Package
        Conditions
               HasPreviousTarget
        Actions
               GetPreviousTarget
               MoveTo
    Package
        Conditions
        HasPlayerSighting
        Actions
        GetSighting
        MoveTo
```

Appendix B: Audio

B.1 Composition

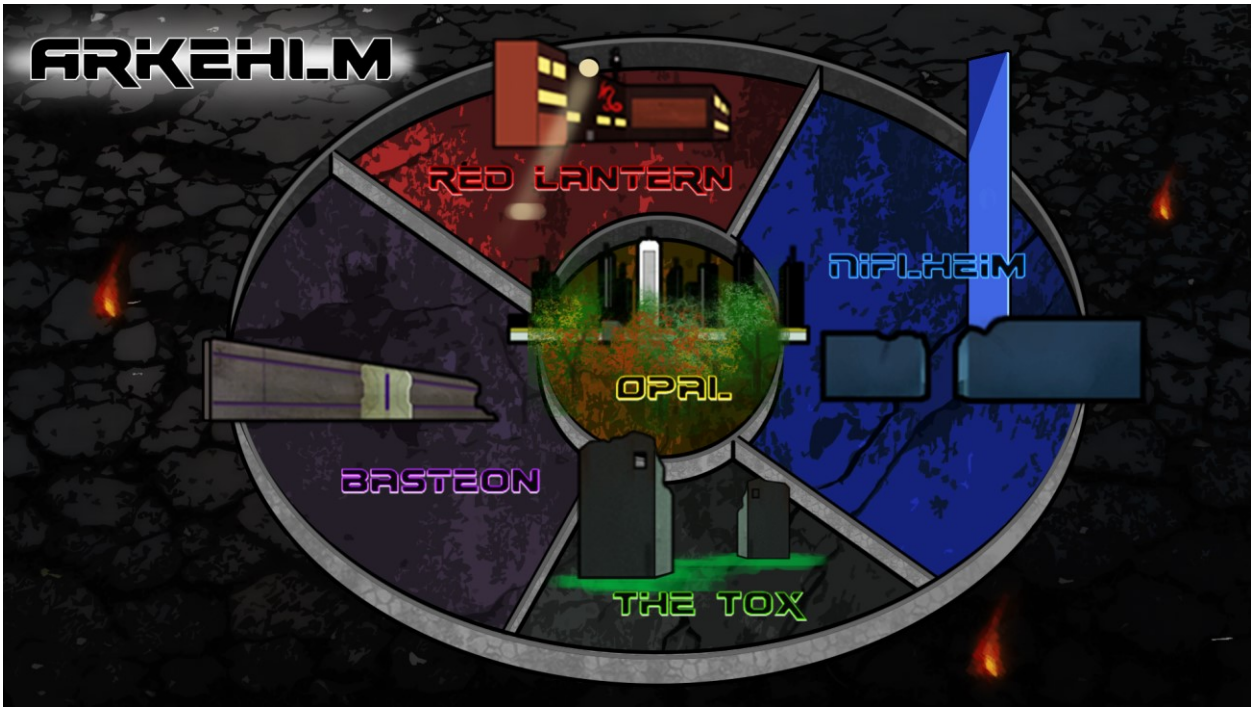


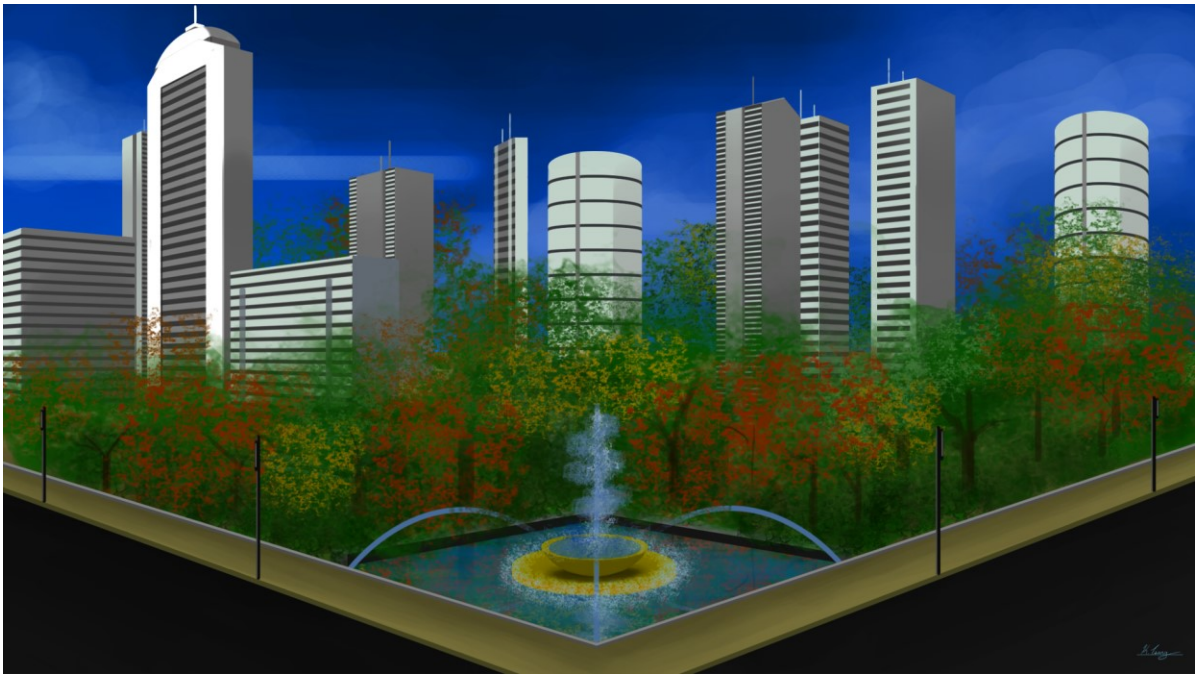
B.2 Sound Effects



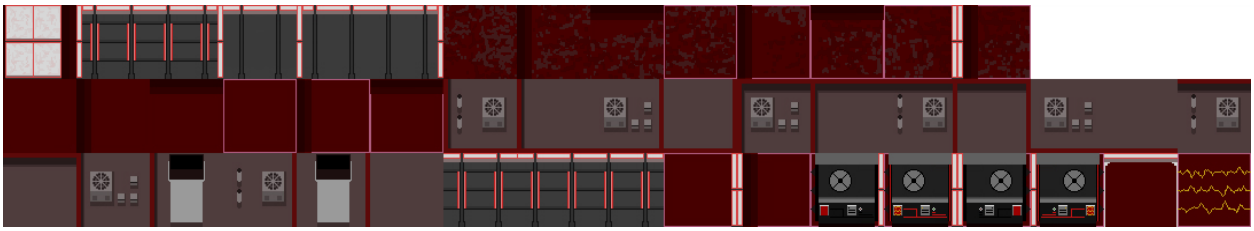
Appendix C: Art

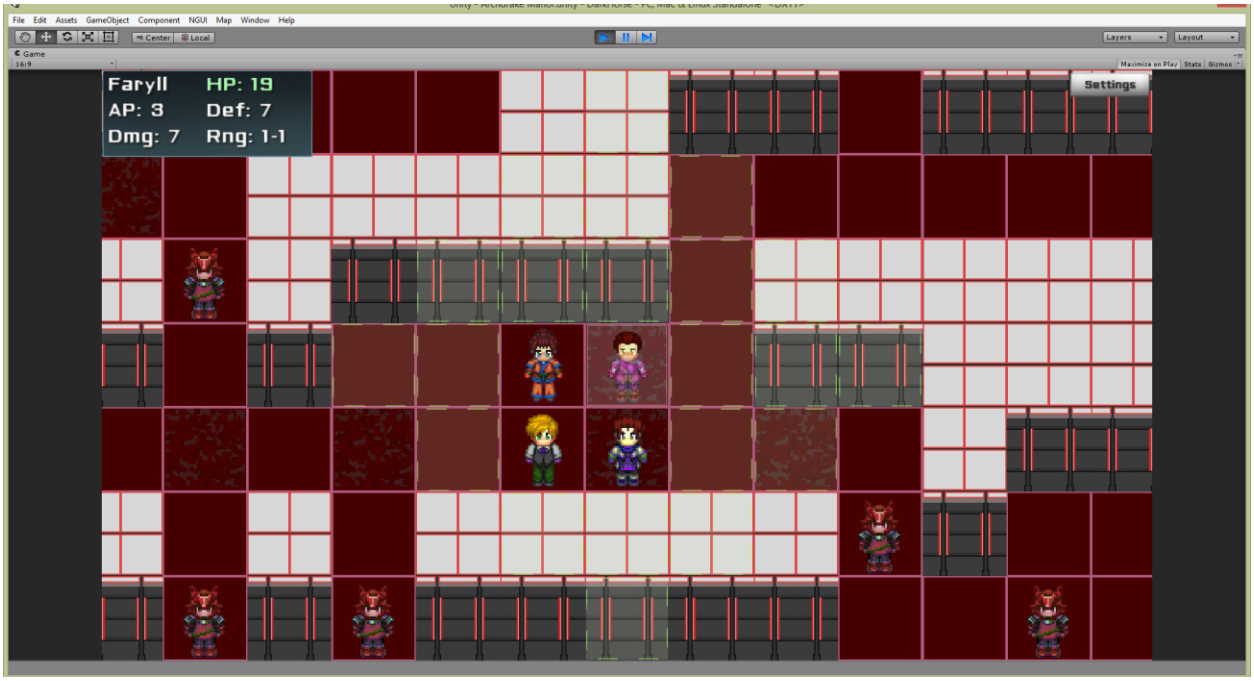
C.1 Backdrops

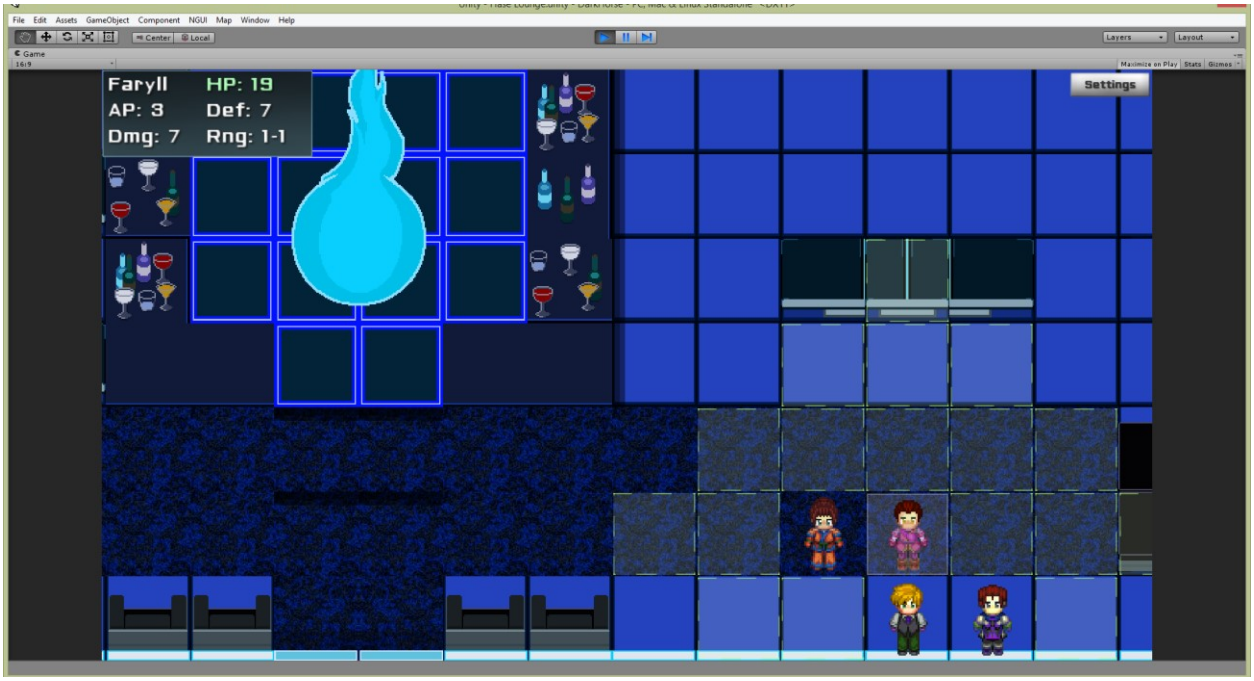


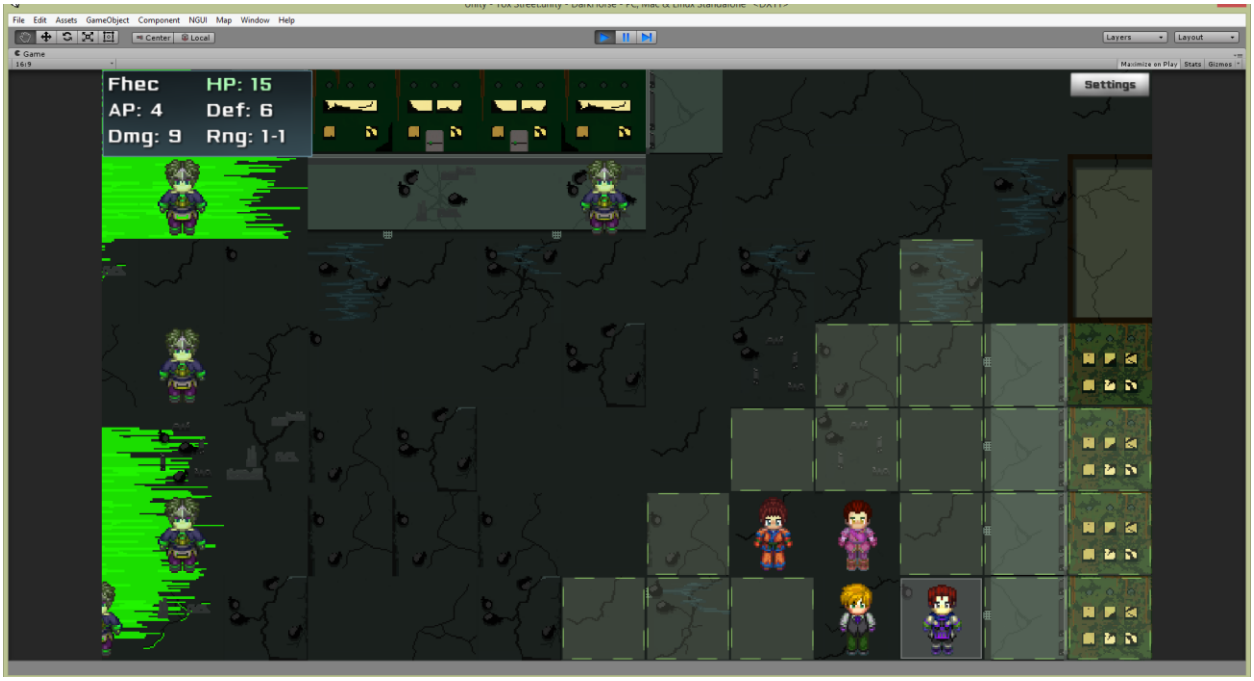


C.1 Spritesheets






File Edit Assets GameObject Component NGUI Map Window Help
Center Local
Layers Layout
Game
Faryll HP: 19
AP: 3 Def: 7
Dmg: 7 Rng: 1-1
Settings


File Edit Assets GameObject Component NGUI Map Window Help
Center Local
Layers Layout
Game
Maximize on Play Stats Gizmos
Fhec HP: 15
AP: 4 Def: 6
Dmg: 9 Rng: 1-1
Settings

Appendix D: Design Documents

D.1 Characters

Name:
Faryll

Bio:

Raised in the Red Lantern, he was sold into slavery for refusing to augment himself into a Drake when he came of age. Faryll used his situation to plan a slave revolt in order to overthrow the Boulderfists of Basteon. Working with his childhood friend Jered, a Boulderfist himself, and relying on the brutish strength of his fellow slave Wenden, Faryll was able to secure transport into Basteon and burn the old regime from the inside out. From the ashes of this uprising, Faryll, Serithe, and Wenden raised up a new power, which they named the Dregs, to run Basteon. A benevolent crime lord, Faryll turns none away, for that is the promise of the Dregs. Even a kinder crime lord is still a crime lord, though, and Faryll will do what he must to ensure the power of his syndicate.

Name:
Wenden

Bio:

By Faryll's side for most of the slave revolt, Wenden is an experienced brawler who harbors a burning hatred for the Drakes of the Red Lantern, who were responsible for slaughtering her family and giving her horrible burns on her hands and arms. Despite her injuries, or perhaps because of them, Wenden brutalizes her enemies with scarred fists, utilizing the dead nerves to relentlessly hasten her opponent's death with each strike.

Name:
Jered Boulderfist

Bio:

A childhood friend of Faryll's, Serithe has always harbored a hatred for the Drakes and the Red Lantern after they sold Faryll into slavery. Serithe would often visit his friend when accompanying his father into the Lantern. It was during one of these visits when they resolved to end Faryll's and all slaves' imprisonment. From this resolution, the Dregs were born. Jered trains to become a formidable sniper Years afterward, after his father's passing, Jered finally gains the power as begin putting his plan into action.

Name:
Aeria Dornell

Bio:

The true leader and Commander of the Azures, she rules from the shadows by manipulating the mentally damaged Magelord Kar. From a young age, Aeria has trained both her body and mind to their limits, and on numerous battlefields she has proven the strength of both. She has long wished for an end to the endless battles between the gangs, however, and has risen to the highest seat of power in all of Niflheim to pursue that wish. After the Basteon slave revolt, Aeria saw a true chance for a lasting and united rule over Arkehlm, one where real progress could finally be made in Arkehlm. As Faryll's exploits grew and grew, Aeria watched and inch by inch, her interest grew to hope. Before long, Aeria found herself falling in love with a man she'd never met. She found herself entrusting her dreams to a man whose face she'd never even seen for herself. Only one question now remains to her, does he have the nerve to do what needs to be done?

Name:
Governor Elias Vrell

Bio:

The appointed governor of Arkehlm, it was under his watch that the city turned to crime and ruin. For years, Vrell tried to fight this growing disease of crime in his city by using all the old powers of politics he gained from the years before the New Dawn. His old ways of thinking weren't able to keep up with the new era, however, and soon organized crime began to grow strong in his blind spots. When he finally saw the threat to his city, it was too late to save it. The criminal powers had taken control of the streets through intimidation and bribery, and there was nothing Elias could do to stop it. The final nail was the Tox Outbreak, which killed hundreds and thinned his security forces significantly. It was then that the governor realized that Arkehlm could no longer be saved in this state. The Tox incident signified how dirty Arkehlm had become, and in Vrell's eyes it was time to clean the slate and rebuild anew. As one last act as governor, Elias built the Quarantine Wall to close off the Tox from the rest of Arkehlm. Soon after the wall's completion, Vrell and all of his subordinates suddenly disappeared within the walls of Opal, and haven't been seen since. In their wake, they left only speculation and mystery.

Name:
Danek Stone (**CUT**)

Bio:

A young Boulderfist soldier and slave sympathizer, he was drawn to Faryll and Serithe's cause but bound to fight against them by honor. In the end though, when his superiors sent him and his fellow comrades out on a suicide assault, Danek realized that his honor is tarnished more by serving the corrupt tyrants of Basteon who care so little for human life than by abandoning his sworn duty. He joins Faryll's revolt and goes on to not only help train but also lead the freed slave warriors using his knowledge of Boulderfist tactics and formations. He is now the Dregs War Captain, in charge of the training and leading of the military might of the Dregs.

Name:
Fhec the Fleet

Bio:

A slave messenger who was treated very well by his masters for his esteemed delivery service. When his masters were killed, Faryll was greeted not with joy but with disdain from Fhec who now claimed it was Faryll's duty to provide the amenities lost due to his masters' deaths. As a result, Fhec constantly berates Faryll for the paltry conditions under which he lives but refuses to leave Faryll's group due to the debt Fhec believes is owed him. Deep down, however, Fhec fears being on his own, and truthfully would have a difficult time surviving without Faryll's support and guidance.

Name:
Automaton Model 6 (Otto)
Bio:

A combat robot built by the Azures, Automaton Model 6 is a vicious and calculating murder machine. Its advanced intelligence programming has given it sentience and it realizes that the Azures only see him as a slave to do their bidding. Using its mastery of destruction, Automaton Model 6 carves a bloody path out of Niflheim into the safety of the Tox, where no living thing dares tread. He calls himself Otto for short.

NPC's:

Archdrake Hassin Averas- Leader of the Drakes
Magelord Kar- Surrogate Leader of the Azures
Soanshi Averas- Trueborn daughter of Hassin
Naiah Averas (formerly Naiah Mantai)- Adopted daughter of Hassin
Areon, Fael, and Daerk Averas- Sons of Hassin

D.2 Classes

Classes:

Melee (1 space attack range)-
Gang Lord- Faryll
Dicer
Brawler- Wenden
Azureblade- Aeria
Automaton- Otto
Edgelord- Vrell

Ranged (>1 space attack range)-
Azure Acolyte
Sniper- Jered
Lancer
Spitter

Cavalry (high movement, 1 space attack range)-
Streetsurfer- Fhec

Class Descriptions:

Gang Lord: Faryll only, 1 space attack range. Melee class.

Lancer: 2 space attack range. Ranged class. Uses a cannon to fire javelins at foes.

Azure Acolyte- Long Range, 2 spaces only. Uses TekGaunts to fire plasma blasts.

Dicer- Close, 1 space claw attack.

Streetsurfer: Close range attack, 1 space. Moves far in a single turn, can move through enemy squares, but cannot land in them. Also can move up walls if he lands in a non-walled space.

Sniper: Long Range Attack, 2 space range only.

D.3 Mechanics

-Base Player Stats-

Note: Bullets are hidden formulaic stats, determined on the spot based on the parent stat itself plus any bonuses. In combat it is the stat and bonuses versus the enemy's opposing factors (terrain, DEF, buffs, etc.).

STR
- Physical Damage
- Equipment Restrictions

DEF
- Physical Defense
- Physical Effect Resistance (Bleed, Cripple, etc.)

-Other Player Stats-

HP
- Hitpoints

AP
- Action Points
- Determined by AGL
- Limits what each character can do in a given turn

EXP
- Gain 100 to gain a level.

Level
- Caps after a point.

-Turn-Based Combat-

Top down combat fought area by area rather than one continuous open world. Each side's units will be allowed to move during their turn. A player turn ends when the player manually ends the turn. Computer turns end after computer has finished making the best moves for their units.

D.4 World Description

In the city of Arkehlm, a jungle of chrome and concrete, you are either found dead in a ditch or taken in by one of the many criminal gangs vying for power. What stands for the “authorities” have been helpless to stem the tide of blood and crime flooding out of Arkehlm’s different sectors for years now and the city has fallen to ruins.

Opal is the central government sector and formerly the seat of power in Arkehlm. Twenty-seven years ago, the governor of Arkehlm recalled every government official and employee inside the walls of Opal and sealed its gates. Surrounding Opal is an area known as the Skids, split unevenly into four different sectors to the north, south, east, and west.

People call the northern sector the Red Lantern, for its notorious human trafficking and prostitution rings. The freakish Drakes have control here.

The Desolates inhabit the southern sector, named the Tox. The sector was previously named Opalhand, so called for the Loyals, a gang created by the governor himself which adhered to his command. The Loyals were wiped out, however, by a terrible epidemic in Opalhand years ago that killed most of the sector’s inhabitants and mutated the rest.

In the east sector, Niflheim, the gang Azure calls the shots. They like to style themselves as mystics and sages, but in truth they simply get high on Hase, a drug that boosts the user’s mental agility temporarily but leaves them nearly braindead when in withdrawal.

The western sector is called Basteon and is run by a relatively new gang called the Dregs. Formed from the rejects of the other sectors, the Dregs are a varied gang consisting of all kinds of people, from nobles to freed slaves. With the leadership of the gang’s founder though, the Dregs have managed to carve a name for themselves as the best infiltrators, thieves, and assassins Arkehlm has ever seen. Our story begins with the beginnings of the Dregs, back when the Boulderfists still ran Basteon.

D.5 Story Synopsis

Prologue-

Characters Introduced:

Faryll
Wenden
Jered
Governor Vrell

Faryll and Wenden (a slave) arrive in Basteon at Jered's compound. Jered and Faryll reveal their friendship. Faryll and Jered go to investigate an explosion.

The two arrive in the holding cells to find a mysterious cloaked figure (actually Governor Vrell). After a brief exchange, Faryll launches an attack but is defeated, allowing his foe to escape. Despite being wounded, Faryll still manages to fight the escaping prisoners with Jered's help (Player Controlled Combat, PCC for short).

After the battle, Faryll and Jered discuss the plan to overthrow the Boulderfists' rule of Basteon using a massive slave revolt.

The revolt is bloody, and many die on both sides, but in the end the sheer physical strength of the slaves combined with Jered's resources allowed Faryll and his fighting force to wrest control of Basteon from the Boulderfists.

Together Faryll and Jered decide to take control as a new gang devoted to taking control of all Arkehlm's sectors in order to finally bring about an end to the endless conflicts between the gangs.

Chapter 1-

Characters introduced:

Fhec the Fleet

Fast forward 5 years after the revolt to the new Basteon, under the leadership of Jered and Faryll who have formed a gang called the Dregs. Faryll deals with espionage and combat issues, while Jered works to restore some semblance of order and politics to Basteon. After these years of

stabilizing Basteon, Faryll now believes their forces are ready to take out the Drakes and gain control over the Lantern.

Under the front of a territory war, the Dregs launch a surprise attack against the Drakes, plunging the two gangs into brutal skirmishes all along the western border of Basteon. During the confusion of war, Faryll leads an elite force consisting of Wenden, Jered, and Fhec the Fleet to break through a weakly defended Drake checkpoint (PCC).

With the outpost captured, Faryll's force rests overnight to regain their strength after the fierce battle, waiting for Faryll's Red Lantern contact to arrive with disguises, allowing them to enter the Lantern undetected.

Chapter 2-

Faryll discusses their mission is to infiltrate Archdrake Manor to kill Archdrake Hassin, the Drakes' leader, and put an end to the Drakes. Jered is initially displeased at Faryll's shady and bloody strategy, but concludes that Faryll is doing the right thing.

After the fight, the group meets Faryll's contact, a female Drake named Soanshi Averas, and her younger sister Naiah. Soanshi is the eldest daughter of Archdrake Hassin, but she has 3 brothers ahead of her in line for the rule of the Lantern. In return for bringing her to power, Soanshi has agreed to ally herself with Basteon and adhere to the regulations and rules of the Dregs.

With the disguises, Faryll, Jered, Wenden and Fhec are able to traverse the city unharmed, acting as Soanshi and Naiah's guards.

The manor has too vigilant of guards, however, and their disguises will not fool them. Soanshi uses a hidden route to get them around the front gates. Once inside the manor grounds, though, the company is on their own and has to fight their way through Archdrake Guardians in order to take out the ruler himself (PCC).

With the Archdrake slain, Soanshi and Naiah are the last two successors of the late Archdrake's lineage; and with a swift strike, Faryll kills Soanshi. Naiah is devastated and halfheartedly attacks Faryll, but she is easily overpowered and stopped. Faryll then tries to dissuade Naiah from her hatred, but he fails and is forced to take her life as well.

Outside, the rest of Faryll's comrades are crestfallen as they see Naiah's blood on Faryll's blade. None raise their voice, nor can look Faryll in the eye.

News of the Archdrake's assassination reaches the front line of battle quickly. Leaderless and demoralized, the Drake forces are soon overwhelmed and routed by the Dregs.

With their army crushed, the Drakes have no choice but to submit to the will of the Dregs as they ransack the Lantern, finishing off any remaining resistance.

Chapter 3-

Characters introduced:

Aeria Dornell

With control of the Lantern firmly in Dregs hands, Faryll explains that the path to Niflheim, a purveyor of addictive and debilitating drugs, is now open. The Azures are more prepared, however, and their defenses are too strong to slip through like with the Drakes. The Dregs have no choice but to launch an all-out assault against the Azures, with Faryll leading the vanguard.

Bursting through a hole in the Azures' front lines, Faryll and the rest of the vanguard push onwards to the heart of Niflheim in hopes of winning quickly by crumbling the enemy leadership.

With the main forces of the Azures seemingly holding the line against the Dregs, the vanguard has a clear shot to the Cerulean Summit; the skyscraper from which Magelord Kar rules Niflheim. Unlike the Archdrake, however, Kar has no heirs and so the line of succession dies with him. Faryll isn't as lucky as he thinks, however, as he meets heavy resistance just before the Magelord's loft (PCC).

After fighting through a legion of Azures, Faryll finally reaches the top floor of the Summit and confronts Kar, only to find him a mindless vegetable. Suddenly, a woman appears from the shadows and introduces herself as Aeria Dornell, the true leader of Niflheim ever since the Magelord destroyed his mind by overdosing on Hase. As Faryll tries to move against her, Azures and defenses appear to stop him. The vanguard has flung head first into a trap.

Aeria offers him a deal. A new Azure technology was stolen and taken to the Tox, and Aeria wants it back. Her men have thus far failed to retrieve it, and she needs a fresh set of eyes with her. If Faryll helps her bring the weapon back into Niflheim, she will agree to an alliance between the Dregs and Azures, and a peace can finally be achieved in Arkehlm. Backed into a wall, Faryll has no choice but accept Aeria's terms.

Chapter 4-

Characters introduced:

Otto

Faryll's vanguard and Aeria enter the disease ridden Tox by rappelling down the side of Niflheim's Quarantine Wall. Not long after they land, however, they're attacked by Desolates, the remnants of the old gang that ruled the Tox before the outbreak, reduced to insanity and hideous deformity because of the disease (PCC). The vanguard dispatches the enemy in front of them, but as they let down their guard, a Desolate launches a sneak attack at Faryll from behind. Before the Desolate can get to Faryll, however, a cloaked figure appears and kills it.

Aeria instantly realizes who the cloaked figure is and leaps to detain him. Faryll and the others move to attack but Aeria removes the hood to reveal Automaton Model 6, a robot designed for war and the only one of its kind. She continues, saying that it malfunctioned and tore a bloody warpath into the Tox. Aeria congratulates Faryll on drawing it out. The robot cries out, saying that he did not malfunction. That he was treated as a slave. That his name is Otto, not Automaton Model 6. He begs with Faryll to help him. Jered and the others are sympathetic, but Faryll knows that to choose his conscience here would be to doom the lives of many Dregs. Before he can answer however, they are surrounded by soldiers dressed in government uniforms. The group is outgunned and completely surrounded, so they have no choice but to surrender and are taken as prisoners into Opal.

Chapter 5-

As the group is brought into Opal, they see that it looks almost exactly the same as it did when they were children. Opal has been thriving while the rest of the city rotted away.

In their cells, Governor Vrell comes to visit Faryll personally, and Faryll recognizes him as the cloaked figure from Jered's holding cells just before the slave revolt. After a brief exchange, Vrell departs the prison leaving Faryll infuriated.

Otto is able to break free of his cell, however, and frees the entirety of the vanguard, but pauses at Aeria's cell. After a brief exchange, he tells Aeria that he is sorry for the lives he ended, but he had to fight for his freedom. He then frees her. To keep her locked away would reduce him to her level.

Faryll explains that Governor Vrell is planning something, and needs to be stopped.

Faryll's forces manage to locate Vrell in a facility beneath Opal. As Faryll enters the central research area, Vrell explains his plan since secrecy has become pointless now. The machine before him can create the Tox's disease. During the epidemic years ago it killed most of the Tox's inhabitants, but left enough survivors for the sector to live on. Vrell saw an opportunity in this, and after building the Quarantine Walls, started to research what caused the disease and how to harness it. He syphoned the disease little by little from the Tox and put his top scientific minds to work studying it. They found it was a bacteria that not only could be vaccinated against but also grown. Since then, he has been growing enough bacteria to wipe out most of Arkehlm all at once in order to wipe Arkehlm's slate clean and start over. After his speech, Vrell orders his guards to bring him Faryll's head (PCC).

Vrell is defeated, and in a final showdown, Faryll drops Vrell into his Tox machine, killing him. As Faryll looks around the facility, however, he sees potential. With this powerful bacteria, he could ensure an indefinite peace with the threat of destruction. Jered sees his descent into darkness, though. He tries to reason with Faryll. Begs with him to think about what this will do, to think about the real peace he sought from the beginning. Faryll doesn't listen. He's lost his ethical battle. Jered relinquishes and agrees to help him. As Faryll turns away, however, Jered raises his gun and fires, blasting Faryll into the Tox pit alongside Vrell's corpse. In the end, to secure a peaceful Arkehlm, everyone's hands had to be bloodied.

Epilogue-

With Vrell and the immediate threat of his disease gone, Faryll's dream is almost complete. The remnants of Vrell's officials either surrendered or died fighting against the Dregs, as the gang attempted to reopen Opal to all the sectors of Arkehlm. Vrell's bacteria is destroyed, but the people are vaccinated for safety. The Tox is renamed Redemption and the Dregs begin repopulating and restoring the sector to be livable once more. They decide to convert Redemption into an agricultural sector, since the streets and buildings have cracked and started to be reclaimed by nature. The large amount of decomposed bodies has also helped fertilize the ground, making it easier to grow crops there. All government officials are pardoned of their acts beneath Vrell and are welcomed into the new society of Arkehlm under the condition that they never speak of Vrell's plan to anyone. The secret tunnels beneath Arkehlm that connect to Opal are repurposed as trade routes between the sectors. All the gangs are dissolved. The practice of slavery is outlawed, and all those living as slaves are freed, and offered jobs to help them regain their freedom fully. Faryll's group of warriors disband to follow their own paths in Arkehlm. Jered assembles representatives from each of the sectors to be part of a council that will help Arkehlm unify itself under one standardized set of laws. Wenden founds the Peacekeepers, a council sanctioned group devoted to the sole purpose of keeping Arkehlm unified peacefully. The drugs of Niflheim are strictly regulated, and while are not banned completely, they are forced to be made much safer, reducing their mind enhancing abilities, but also their danger.

Aeria spearheads this drug regulation, and helps open clinics throughout Arkehlm. Fhec helps newly freed slaves gain their footing in Arkehlm by helping train them in useful skills. Otto leads Arkehlm's strongest fighters on scouting missions outside the walls of Arkehlm in order to find more sources of food and look for other cities. Together, the group helps erect a statue of Faryll, the hero who fell at the height of his dream becoming reality. Arkehlm's hard-fought peace has finally arrived, and Jered plans to honor the one person who had the vision to see it as possible.