RavenGuard

The Design and Evaluation of Combining Disparate Gameplay Systems

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science in

Computer Science

and for the

Degree of Bachelor of Science in

Interactive Media and Game

Development and for the

Degree of Bachelor of Arts in

Interactive Media and Game Development

Authors:

Charles West cpwest2@wpi.edu
Alex Marrinan ammarrinan@wpi.edu
Michael Weideman mrweideman@wpi.edu
Griffin Bowers gbowers@wpi.edu

Faculty Advisors:

Prof. Walt Yarbrough wyarbrough@wpi.edu
Prof. Michael Engling mengling@wpi.edu

Additional Contributions:

Rose Sand dsand@wpi.edu
Tate Donnelly tdonnelly@wpi.edu
Ian Tschida tschida@wpi.edu
Ben Levy belevy@wpi.edu
Ben Slattery benjamin.o.slattery@protonmail.com

Abstract

RavenGuard is a 2D tactical roguelike video game for the PC, putting the player in charge of the RavenGuard, a group of misfit mercenaries fighting to save the kingdom. Developed in Unity using C#, RavenGuard seeks to combine the quick run based experiences presented in roguelikes with the slower methodical gameplay found in tactical RPGs. Numerous design, technical, and artistic challenges were pursued to combine the key elements from these two disparate gameplay systems. This project researches and reports on the effect of blending these differing genres of games, detailing both successful applications, growing pains, and where more work needed to be done.

Acknowledgements

The team would like to thank our two advisors; Walt Yarbrough and Michael Engling for their invaluable support and advice throughout the duration of this project. We would like to thank Ben Slattery for his work on the music of the game, and for going above and beyond what the team initially asked of him. Finally, the team would like to thank everyone who helped playtest or give feedback during the development of *RavenGuard*.

Table of Contents

List of Figures

1. Introduction

RavenGuard is a single player, tactical RPG roguelike, where the player is placed in control of a small, evergrowing band of mercenaries as they fight their way through three areas. Each of these areas are filled with various challenges requiring thoughtful strategy and creativity from the player.

The project of *RavenGuard* was primarily designed around the idea of combining two different experiences, being tactical RPGs and roguelikes. This goal shaped many of the decisions made when designing the game, and led to the eventual focus on drawing out player creativity and creating intellectual challenge. This project also challenged the team to improve, master and showcase various skills they developed while at WPI, such as programming, 2D art, animation, and audio design.

This paper is divided into various sections that each detail different attributes about this project's development. The background section provides insight into the inspirations of *RavenGuard*, and where the team often looked for points of reference. Following that, the design section elaborates on the design and experience goals for the players, the thought processes used during design, and the eventual systems that would make up the final game. The gameplay implementation section discusses the technical work that went into making the designs for *RavenGuard* a reality. Art development and audio each discuss the creation, process, and implementation of their respective areas for the project. The testing section reviews the feedback the team received while working in the project, and conclusions that were drawn from it. And finally, the discussion section reflects on the project as a whole, elaborating on challenges faced, success of the project, lessons learned by the project team and individual members and what the future holds for *RavenGuard*.

2. Background

2.1. Game Inspirations

Different types of games often appeal to different styles of play, many of which can often be found at odds with each other. Long, methodical tactical RPGs and hectic, improvisational roguelikes are two which often contradict. With *RavenGuard*, we wanted to explore what would happen if we were to bridge these two opposites. While designing the mix of design pillars that would help establish the mechanics to make up *RavenGuard*, the team drew inspiration from many titles existing in the two respective genres that *RavenGuard* is designed to fall between. The series of *Fire Emblem*, was by far the largest inspiration from the side of tactical RPGs. For roguelikes, many came up in conversation, but we found ourselves going back to the intricacies of *Peglin* (Red Nexus Games, 2022), and the distinctness of *Vampire Survivors* (Galante, 2022) quite often.

2.1.1. *Fire Emblem*

The series of *Fire Emblem*, developed by Intelligent Systems and published by Nintendo, was one inspiration that stuck out to the team during development quite heavily. Excluding the more adventurous spinoffs that the series has spawned, the mainline games all find themselves at the center of the tactical RPG genre. With over 15 games in the main series, there is a lot to take inspiration from. However, a few notable games stuck out to us while creating *RavenGuard* and establishing the design pillars for the game. *Fire Emblem: Heroes* (Nintendo, 2017), is a spinoff game that sticks to the core tactical gameplay found within the mainline entries. But, it simplifies it down a lot to work better in the quick pickup and play setting often found with mobile gaming.

This simplification creates a core gameplay loop that emphasizes strategy and player control/build-crafting. The core loop also creates a faster-pace, with more emphasis on the abilities of singular units than armies as a whole, creating a sense of **satisfaction** and **pride** when a unit the player designed is able to perform well. These feelings were feelings we wanted to replicate in *RavenGuard*. The emotions found here functioned as the primary pillars when designing the skills and unit progression systems in the game. They were created in a way that would allow players to create units of their own, and be able to then go into these quicker combat sections and get that satisfaction, or a feeling of defeat, depending on how their build-crafting worked out.

The other games have also helped. Particularly older entries, such as *Fire Emblem: The Blazing Blade* (Nintendo, 2003) have played a large role as well. Due to being a full sized entry to the series, it focuses much more on deeper mechanics that play out over the course of a full game. The battles are slower and happen at a greater scale, which causes the focus to shift more towards the player's ability for **strategy** and **crisis management**. The progression systems found within *The Blazing Blade* also go on to allow for player **growth over time** and expansion of the player's options over time, because it has a full game to play around. With *RavenGuard*, we had wanted to capture that same feeling of strategic planning and crisis-management found within these systems. This inspired the way units were handled, with the goal being to replicate these emotions created by *The Blazing Blade*. This resulted in designing units in a way to make them valuable to keep alive, otherwise risking the loss of the investment made into them. It also inspired some long term progression features, such as the player being able to upgrade skills between runs, or directly manage the level-ups granted to their units throughout the run.

2.1.2. *Peglin*

Developed by Red Nexus Games and released to Steam Early Access on April 25, 2022, *Peglin* (Red Nexus Games, 2022) is a roguelike that features a goblin who plays pachinko in order to defeat foes and continue on an adventure. During a run of *Peglin* (Red Nexus Games, 2022), the player collects orbs that they can throw into the board. These orbs each have quite distinct functions, which makes finding them during a run, even after knowing what they do, capture a sense of **wonder** and **excitement**. This wonder and excitement brought forth from these orbs was a feeling we wanted to capture for the player while they go on their own quest in *RavenGuard*, so these feelings became core pillars for how we designed individual skills, units, and encounters the player can come across in *RavenGuard*

Peglin (Red Nexus Games, 2022) also presents a strong atmosphere for the player to get into. The game is very good at capturing the unknown excitement present in an adventure, and is able to pull the player into this sense of the unknown. *Peglin* (Red Nexus Games, 2022) is able to balance this more serious adventurous tone with humor too, as it's aware the concept is a bit unusual and plays into it quite a bit. *RavenGuard* is not a game where it would have the same amount of humor, but the same mix of some light-hearted jokes for various skill names while being able to lure the player into a sense of adventure was a feeling we wished to emulate. These feelings became core pillars for us as we designed and built the world map and other overall roguelike elements, since we wanted a run to feel like a bit more than just some way to get to fight enemies.

2.1.3. *Vampire Survivors*

An indie roguelike game remaining in development for quite a few years before being released on Steam in October of 2022. Developed primarily by Luca “poncle” Galante, *Vampire Survivors* lets the player pick one from many different characters, then play 30 minute rounds where they can level up and find new items to use.

In terms of *RavenGuard*, where the influence of *Vampire Survivors* (Galante, 2022) comes into play is mostly in the cross run progression, and the Paragon units. In *Vampire Survivors* (Galante, 2022), the player can pick one from a vast array of characters, each that have their own unique starting weapon, as well as a unique skill that changes how they play, often encouraging a different style of buildcrafting with each character. Throughout playing the game, the player can unlock more characters to use as well, further diversifying the types of runs they can have. The characters within this system often allow for players to make different strategies that reward the player, and can inspire some more creative uses of items that might have been lack-luster otherwise. We wanted to capture this same sense of **creativity** within the Paragon units found in *RavenGuard*, to have the units act as capstones for players to pick and explore strategic and creative choices based around their unique traits.

Outside of the characters in *Vampire Survivors* (Galante, 2022), the player can also get gold during and at the end of a run, and once they do, they can use it to unlock permanent upgrades or new characters to use while playing the game. Outside of just gold, it is also possible for the player to unlock new items, characters, locations, and even whole new mechanics to utilize by completing certain achievements. This gives a sense of **growth** and lets the game give the player a feel of making progress, even when the runs continually reset. The knowledge of this system and how it impacts *Vampire Survivors* (Galante, 2022) helped when figuring out how to capture the feel of progression and continual growth within *RavenGuard*, eventually coming to reality with the town players can visit outside of runs to unlock and upgrade skills.

2.2. Art Inspirations

2.2.1. *Fire Emblem: The Blazing Blade*

The second title in the Fire Emblem series released for game boy advanced, *Fire Emblem: The Blazing Blade*, is one of the primary inspirations for art as *RavenGuard* is intended to be a TRPG like the Fire Emblem series of games. *Fire Emblem: The Blazing Blade*, was first announced in early 2003 and was released on April 25, 2003. The pixel art style and animations of the battle scene and combat in this game is what inspired our animations and style for protofest. We used a similar style of lower pixel sprites on the "board"/environment and higher pixel sprites for the battle scene and animation. Below is the higher pixel battle scene which we drew inspiration from.

Fire Emblem: The Blazing Blade (Nintendo, 2003) had a battle design style that we wanted to replicate for *RavenGuard*. The layout fits the design goals for what our game's battle scene should look and play like.

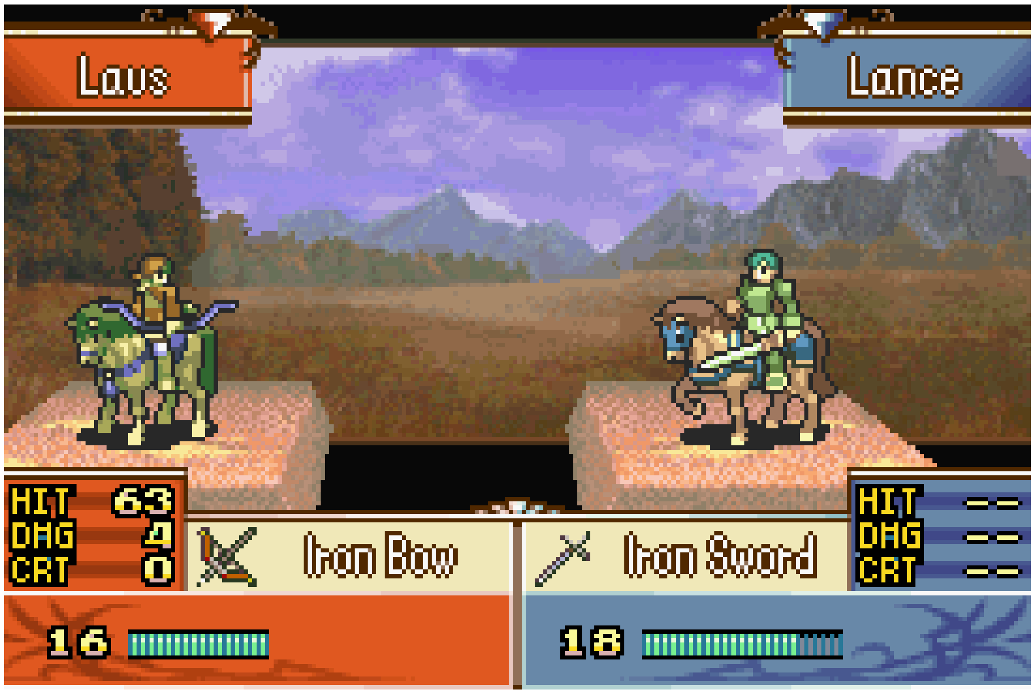


Figure 1: A battle scene from *Fire Emblem: The Blazing Blade* (Nintendo, 2003) Image source (WAve, 2020).

2.2.2. *Pokémon HeartGold / Pokémon SoulSilver*

Pokémon HeartGold Version and Pokémon SoulSilver Version are the 2009 remakes of the 1999 Game Boy

Color RPGs Pokémon Gold and Silver, also including features from Pokémon Crystal. The art style and color palette influenced the creation of the sprites color pallets and the simplicity of them while conveying the image. The goal was to replicate the way Pokémon uses few colors to make the image readable while also not being overcrowded with some colors only having one pixel. Pokémon also uses lower pixel sprites for the overworld and higher pixel sprites for the battle scene which also aided as a reference and inspiration for animation.

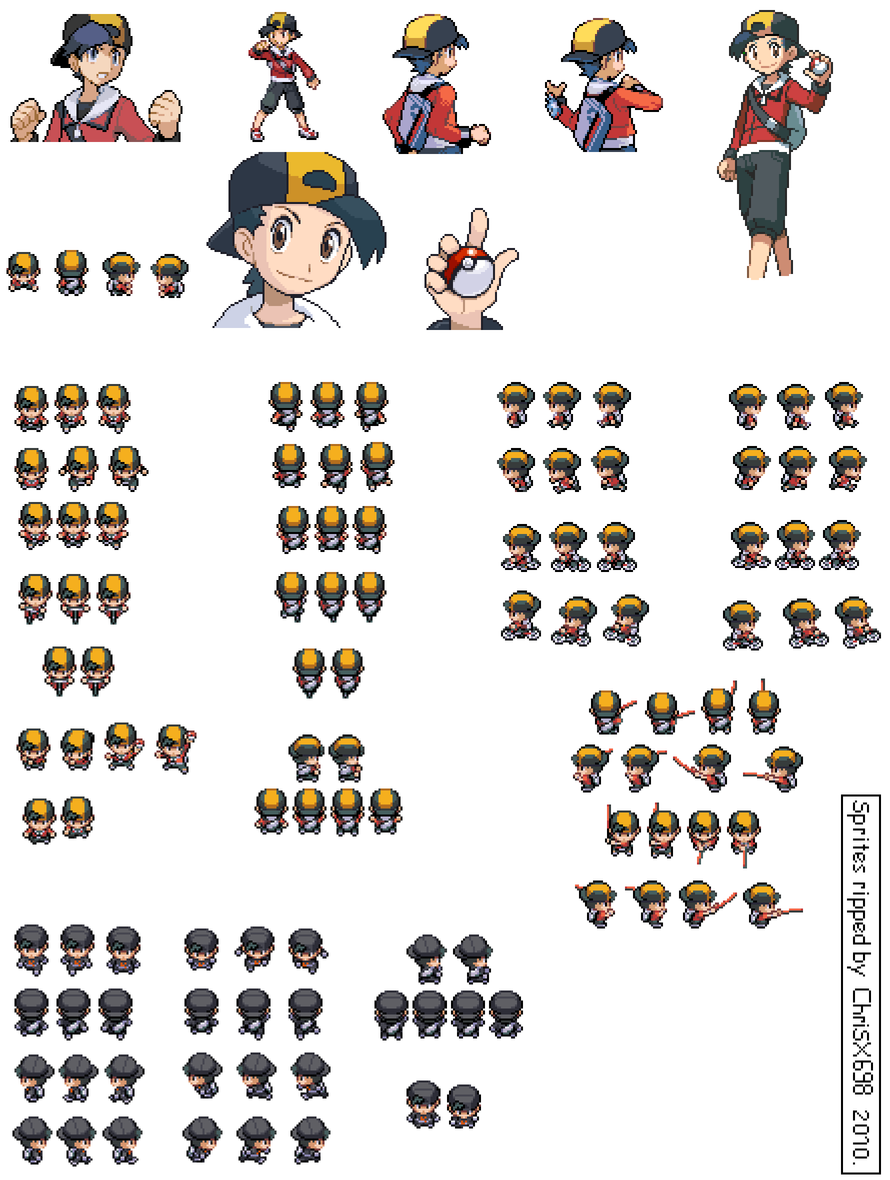


Figure 2: The spritesheet for the protagonist trainer in *Pokemon Heart Gold* and *Pokemon Soul Silver* (Nintendo, 2009), Image source (ChriSX698, 2010).

3. Design

This section documents the design and planning that went into creating the mechanics behind *RavenGuard*. It covers the goals, and higher level thoughts that dictated the resulting ideas. But also dives into these ideas themselves, the processes taken to create them, decisions made on these ideas, and the resulting structure of the final product.

3.1. Design Goals

The main design goal of the creation of *RavenGuard* was to successfully blend the gap between the more methodical, drawn out tactical RPGs and the more chaotic, improvisational roguelikes. We strived to create a sandbox where the combination of the two genres feels natural, and the randomness found in roguelikes empowers the player to be tactical and inventive, instead of just cutting back on scripted challenges. One of the main overlaps between these two styles of game is having an element of buildcrafting. We worked to ensure that the buildcrafting in *RavenGuard* not only allows the player to get progressively stronger, and itch their roguelike run-based build creation, but to also allow the player to hatch even more ingenious strategies on the battlefield as their runs progress. The end result of this goal was something we wanted to be capable of being enjoyed by players from either genre that inspired this game.

3.2. Experience Goals

For *RavenGuard*, we set experience goals for ourselves early on to understand what experience we wanted the players to feel when playing the game, and to ensure we took steps towards that goal regardless of adjustments made to key features and mechanics. By far our largest goals were capturing the feeling of strategicness and control for the player. The core gameplay loop was one of a Tactical RPG, and despite the random elements we wanted players to feel like their success revolved around their decision making. To help ensure this feeling of control and strategic ability remained with the player, players were given quite a lot of agency with what they do with the skills they acquire throughout a run and how they design their army. We also wanted the player to be able to feel an empowerment over time, and an increase to their options over time to allow for deeper strategies to emerge. These goals were tied quite closely to how we tuned the roguelike elements of the game to play out.

One key component that can be found within many tactical RPGs and Roguelikes is the ways each offers challenges that encourage the player to think creatively. tactical RPGs often offer a wide variety of methods for players to customize their forces, allowing armies to adjust to the type of tactics the player enjoys employing. Creating armies that fulfill these goals, and ones that can be effective against the foes encountered often requires and helps cultivate creative expression from the player. Roguelikes achieve this same sandbox in a slightly different way. A key feature across most of the genre involves collecting some sort of item, an item that offers specific upgrades or skills that slowly make the player stronger. However, the items the player *does* get a chance to get are usually uncertain, and in order to make it longer into runs need to be used as optimally as possible. This improvisation is what roguelikes often use to create this environment ripe for players' creative expression. Because of this strong overlap, properly capturing this feeling was something we believed was very important to make this blend of genres feel as natural, and enjoyable, as possible.

3.3. Audience

During the development of *RavenGuard* we defined our target audience based on the games and media that inspired the creation of the game. With our goal of creating a game as a middle ground between two genres, we aimed to design the experience in a way which would appeal to fans of either genre. We wanted the core gameplay and build-crafting to appeal to people who enjoy that style of tactical RPG and strategy games, such as the *Fire Emblem* or *XCOM* series. But, we also wanted the overarching gameplay experience to draw in people who enjoy roguelikes as well, drawing in people who enjoy those games, even if they may be new to this particular TRPG style of gameplay. It was also concluded that some of our audience might come from people interested in the game simply due to its medieval fantasy setting, but our efforts were focused on ensuring enjoyment from the audience of tactical RPG and roguelike fans.

3.4. Mechanics

The various mechanics of *RavenGuard* are all designed around giving the player creative and intellectual challenges while playing the game. During the game, the player controls a small squad of soldiers from the mercenary company of *RavenGuard*. Between the lead Paragon units and other soldiers the player will find, they each are very adaptable to the situations they find themselves in. During each run, the player is able to collect various skills that give tradeoffs to units, working as pieces to a greater whole. Thus strengthening the player's forces if used appropriately. The player is also able to grow stronger outside of runs, by either unlocking new skills and units to find or by upgrading the skills they can find on their adventures. But the skills are by far the most emphasized system within the game, serving as the main branch between the two genres that RavenGuard is trying to bridge.

3.4.1. Units

Units in *RavenGuard* are one of the key systems found in the game, and one which the player will interact with possibly the most. From starting a run of the game to planning their success, the player's choices always revolve around the units in some way. But grouping units as a whole into a single mechanic isn't really telling the full story. a centerpiece to *RavenGuard's* design, is that the units are made up of many different systems working in harmony. For example, starting a run requires the player to choose a Paragon unit to lead their forces, and four other units to support them. The units chosen will go to shape many of the player's future decisions. Or that while in a run, the player will find new skills to equip their units with, changing how they work in battle and how the player is able to make use of them.

At their core, a unit is made up of a few deciding factors. One of which is the unit's base stats. Level-ups in *RavenGuard* are very dynamic, allowing the player to pick which stats they feel are important for a unit to increase. The stats a unit starts with bear an effect on where their stats might end up. Regardless of what direction the player takes the stats of their units, the spread of them are key to keep in mind when outfitting units with skills.

Another key aspect to understand when making use of skills is a unit's class and weapon type. A unit's class determines their movement distance and what class-related skills they will be able to equip. Weapons determine the attack range a unit can engage in by default, what weapon related skills a unit can equip, which other weapons the unit is effective and ineffective against in combat, and in some cases, which defensive stat of a foe the unit will target. Combined, these factors make up the foundation of a unit, and give players varying starting points upon which to craft their army.

3.4.2. Paragon Units

Chosen at the start of a player's run, a Paragon unit is a type of unit that stands out from the rest. These units not only look more distinct than their standard counterparts, but also have access to a powerful Unique Trait, which will shape the goals and strategies a player may go for during their run.

The Unique Trait of a Paragon is by far the most important piece of their design. These unique traits are each designed with a focus on giving the player powerful abilities they would want to consider when creating their strategies during the run. Encouraging synergy, or flipping some fundamental mechanic on its head, are goals often aimed for when creating the unique traits paragon units will bring to the battlefield during a run. These are one of the only parts to their army the player is able to willingly choose, so it is important that the decision is able to not only feel and be impactful, but also go a long way in keeping their runs interesting.

Paragon units also have some smaller mechanics that help them to stand out. Due to them being a class of their own, they can equip skills regardless of class-based restrictions that they might have, allowing the crossing of skills that might otherwise be impossible.

These two skill-based abilities render Paragon units a powerful force on the battlefield. These units are the player's start, and often the centerpiece to their tactics. But, in order to keep the intensity that encourages the player to make tough decisions, or need to think deeply about their tactics, Paragon units needed something to offset their strength. The solution to this was making it so the player *needs* to keep the paragon unit alive. Other units, if they die you'll lose them and their skills, but be able to keep pushing forward. Paragons don't give that flexibility. If a Paragon unit is defeated, the run ends. This extra risk allows the paragon units to function at a higher level than other units, and avoid feeling like a one size fits all solution to any problem the players might encounter.

3.4.3. Skills

The skills in *RavenGuard* do many different things. Some grant buffs, some offer extra movement, and others can flip fundamental interactions on their head. Found during runs, skills are how a player can express their creativity, and prepare for harder challenges. Each unit is able to equip 3 skills. The skills themselves are divided into two main categories, active skills and passive skills.

Active skills are skills the player needs to manually activate, spending a unit's turn to do so. These skills can often offer extra movement options, strong buffs for allies, or offer damage over a wide area. But after their usage, they go on cooldown, rendering them inaccessible till their countdown ends. This cooldown, instead of being tied to turns, is tied to the attacks in combat a unit deals or receives, forcing a limited use for these skills if used by a unit that often avoids combat. The reasoning behind this choice is that some units can simply move further than others. Linking the active skill cooldown to the overall turn count, can reduce the need for a player to interact with their enemy and make choices, and instead may allow a more spacious playstyle, ultimately reducing the number of, and importance of strategic choices being made. These skills also offer no extra stat bonuses or ways to increase a unit's effectiveness outside of the very skill itself.

Passive skills are the opposite to their active skill counterparts. They each act as a new functionality a unit will innately have, and will activate automatically when their conditions are met. Each passive skill however, follows one similar pattern of design. Each skill involves some condition that it must meet in order for it to trigger and the unit to receive its benefits. If the skill lacks such a condition, it will instead come with a negative effect to offset the free positive. Skills are designed in this way to make them just as involved in creative planning as the other aspects of the game.

Initially, many skills didn't have these conditions or downsides, and thus functioned closer to direct upgrades. Deciding who to give a skill to was thus often less important, as whoever received it would always simply be better than previously, and there was little interaction between skills themselves. By adding these extra effects, it not only makes choosing which skills to equip onto a unit require more forethought, it also allows some skills to synergize strongly with each other, or enable a unit to cancel out negatives it receives from certain skills. In some cases, a negative effect from one skill might even help other skills activate. This design choice works to encourage players to find creative ways to piece skills together, and find skill combinations they enjoy using. We found that this extra difficulty makes it much more satisfying when the player is able to find a successful way to combine skills in a way that makes them more effective than they would have been on their own.

3.4.4. The Town Square

Progression within *RavenGuard* is not just restricted to in-run progression. We wanted the game to grow with the player, both within the span of a run and across runs, which we fulfilled with the creation of the town square. At the town square, players can spend coins in order to upgrade skills and unlock new paragons to use during their runs. These new and improved tools can make it easier to progress further into a run, or lead to more complicated builds or tactics for the player to exploit as they grow more experienced with the game.

Coins are not used directly during a run, allowing them to give the player a pure sense of overall progress when collected. Throughout a run, completing a battle stage successfully will grant these coins, with an even greater reward for successfully making it through all stages of a run, encouraging further progress.

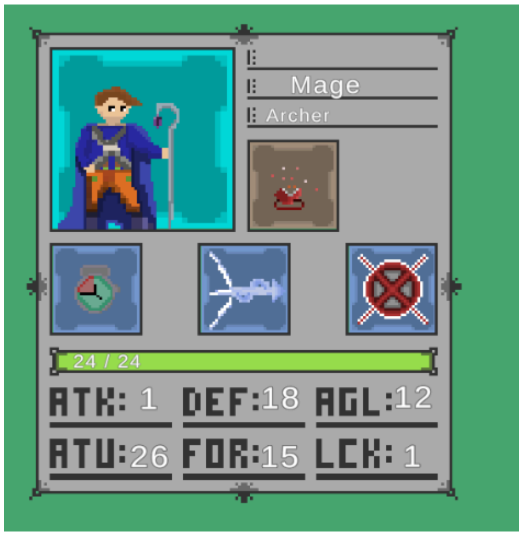
3.5. User Interface

When designing the user interface features for *RavenGuard* there was one main difficulty in which we were constantly working to overcome. *RavenGuard* is a game involving, and often requiring, an intricate understanding of a lot of details in order to achieve success. Communicating these ever changing details to the player clearly, and effectively was the main goal of the UI design in the game.

Out of the many different UI elements found in *RavenGuard*, there are two menus which come up the most often, and contain the most relevant information for the player. The Unit Stats Menu, and the Inventory Menu.

3.5.1. The Unit Stats Menu

The unit stats menu is an interface used to display every attribute and ability of a given unit. All 6 statistics, HP, equipped skills, and the name and look of the unit are all shown. The menu also displays any stat changes due to applied buffs and debuffs, with a green color indicating an increase and a red color indicating a decrease. If no unit is currently selected, the unit stats menu is shown for the current unit highlighted by the cursor. If a current unit is selected, the corresponding unit's menu will remain on screen regardless of cursor position. When a unit is selected, if the player hovers their cursor over a foe unit within range, the unit stats menu will create a predicted outcome of the battle that could occur between the two units. This prediction does account for the effects each unit might have due to skills, allowing for the prediction to accurately depict the result for players. The preview shown allows for players to make a mental note of the outcome, and focus their time on the strategy part of the experience, and less about crunching every number of every possible battle themselves. Below is an example of a single stats menu and two stats menus showing a battle prediction.



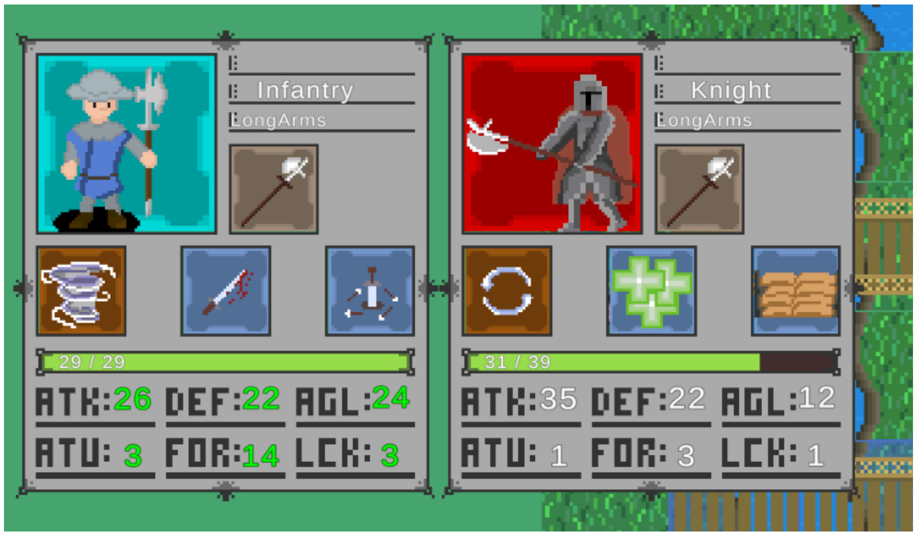
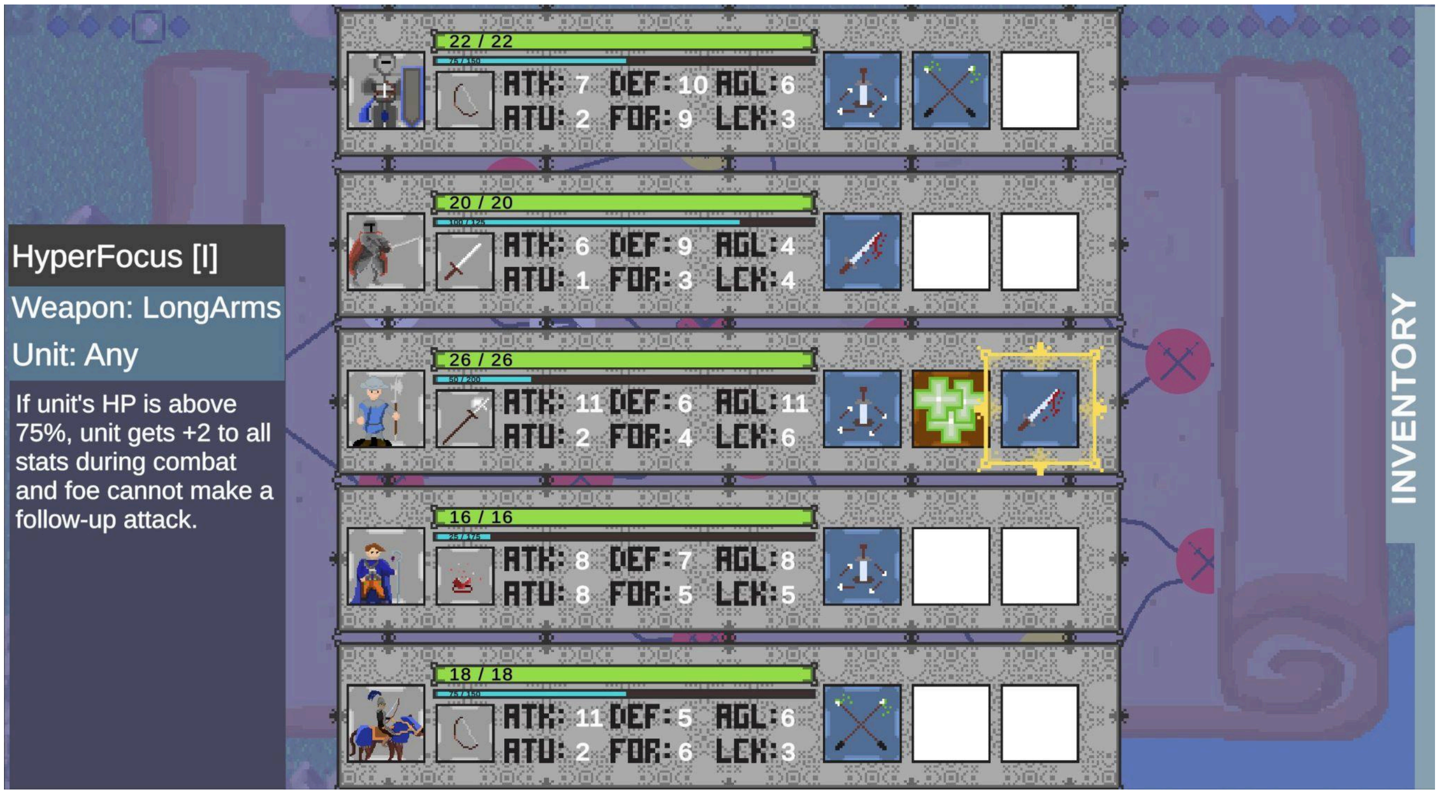


Figure 3 & 4: *RavenGuard*'s Unit Stats Menu, alone and in battle preview mode.

3.5.2. The Inventory Menu

The unit stats menu is where the player is able to view an overview of all their soldiers, along with being able to customize their units between levels.

During levels the inventory menu is utilized as an easy way for players to get an overview of every unit under their control. Each unit gets its own card in the menu, of which players can get all the information offered via the Unit Stats Menu. On top of this, any skills currently equipped can be previewed for the player to review what any given skill does. Skill information is displayed on the left side of the screen. Bouncing between each unit individually can get tedious, and keeping track of every stat while you look over all your units can become very exhausting. The overall overview provided by the inventory menu seeks to make it easier to view your army's current status as a whole. An example of the overview state of the inventory is shown in the figure below.



HyperFocus [I]
Weapon: LongArms
Unit: Any
If unit's HP is above 75%, unit gets +2 to all stats during combat and foe cannot make a follow-up attack.

Figure 5: *RavenGuard*'s inventory menu

The other main function of the inventory menu is the inventory and unit customization side. When on the map in-between stages, player units are able to be customized. By opening the tab to the right of the menu, the player can view their list of unequipped skills. By selecting these, or selecting empty slots on the unit menu, the player can equip skills to slots in which they are able to be applied.

Notably, the ability to customize the skills units have equipped is disabled during battles. One of the main reasons this is the case is due to how it alters the player experience around planning. Altering skills on a unit has no cost, so in a situation where skills can be swapped mid battle, it encourages a style of play involving constant skill-swapping to applicable units, instead of developing a long-term plan and role for how each of your units might need to be used. By instead restricting skill altering to be only done between stages, it still allows players to experiment with different set-ups, but also encourages a player to form a longer term plan and prepare for each unit to fill a certain role on the team.

3.6. Game Structure

The game structure of *RavenGuard* is focused around players repeatedly experiencing individual “runs” of the game. During a run, the player makes progress by navigating through a web of connected stages, picking between potential paths to strengthen their forces until facing the boss at the very end. If the player fails at any point, the run ends and they are forced to begin anew. When navigating a run, similar to existing titles such as *Inscryption* or *Slay The Spire*, the player can see what spaces and paths lay ahead of them, and must choose which way they wish to explore what lies ahead.

Outside of individual runs, as mentioned in section 3.4.4, *RavenGuard* fosters player progression via the Town hub. The town allows players to spend coins they earn during runs on upgrading skills or unlocking new paragons to lead their units into battle.

3.6.1. Stages

During a run in *RavenGuard*, there are three potential stages for the player to come across; Battles, Shops, and Boss. Each of these stages work differently, creating different incentives for the player to pick where they want to go when planning their route.

Battle stages are the most common stage and where most of the gameplay will take place. During these stages, the player is placed on a randomly chosen map from a total pool of fourteen, and tasked with the goal of wiping out all the foes inhabiting the said stage. Upon the successful completion of this stage, the player is rewarded with some coins, and a free skill, chosen out of a random selection of three. The end rewards of a battle stage aren’t the only benefit they offer. During these stages, chests can also be found scattered throughout the map. Moving a unit onto one will collect the skill that remains inside, allowing for more options for player strategy and build-crafting between levels.

Shop stages are less common than battle stages, and function in a much more straightforward manner. There is no fighting, no xp gain, and no coins. However, when entering the shop a large selection of five skills are presented to the player. Out of this selection, the player is able to freely choose whichever two they wish to keep. If the player's soldiers are comfortably leveled up, and the player is hurting when it comes to the skills they can equip their troops with, shop stages can be a powerful resource to knock a player's skill inventory onto the right track.

The final stage is the boss stage. The boss stage will only appear at the end of a run, and features a unique map that will only appear on this stage. This map does not spawn more chests for the player, but instead will spawn the Evil Tyrant boss unit. This unit is incredibly powerful, and functions as the last large hurdle for the player to overcome. Once successfully clearing out the tyrant and other foes on the stage, the player is rewarded with a large sum of coins and the run is successfully completed.

3.6.2. The Map

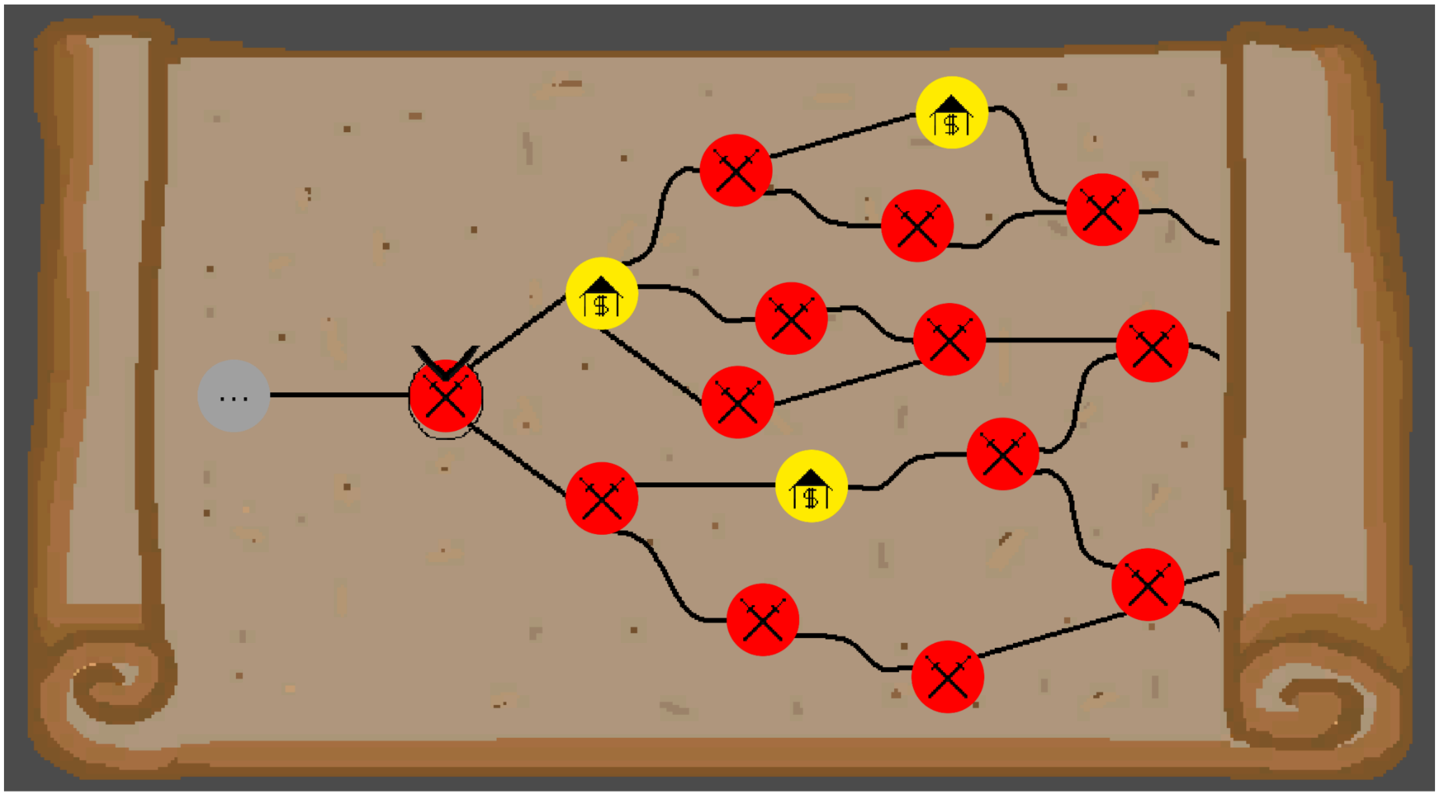


Figure 6: The map used to move between stages in a run in RavenGuard.

In order to traverse between stages during a run, the map is used. As shown in the figure above, the map is composed from a variety of nodes representing stages, and connecting paths.

The nodes on these paths will randomize between battle stages and shop stages, and it is up to the player to carefully pick which path they want to traverse as it affects which stages will, and won't be available to the player. Regardless of pathing, the player will traverse a total of 8 stages during a run. Come the end of the run, each of these will converge onto the boss stage for the player's final bout.

3.7. Level Design

The levels of RavenGuard are designed to strike a balance between offering an expansive area for the player to explore and strategize around, but also feel quick enough that playing through multiple levels during a single run of the game is not taxing to the player. Many of these levels also offer various ways to approach them based on the units the player might have access to, creating numerous potential strategies. Although, the levels were not always planned to be created in this way.

The way in which levels were constructed went through many iterations throughout the stages of this project. Initially, when creating levels for *RavenGuard*, the goal was to capture the feeling of bracing for the unknown typically found in roguelikes via procedurally generating the levels, ensuring the player would truly not know what area they'd need to plan around next.

The very first version of level creation was essentially purely random. But, as revealed by some of the earliest stages of playtesting, it was concluded that such a loose generation method was not only too unwieldy to create consistently playable levels, but also that it was unable to create spaces that would offer tactical challenge and intrigue to the player. The result of this was that any and all tactical thinking revolved entirely around the units themselves, and even with the introduction of some skills affecting nearby units, strategy involving formations and unit movement were still disregarded. This resulted in a weaker feeling of strategy and planning, which was something we didn't want to compromise on.

Thus the next attempt at creating procedurally generated levels was born. This attempt was significantly more controlled than its completely random counterpart. This version of generation is based on a system using bases and layers placed on top of those bases. By making use of predefined pieces, and allowing the generation to piece them together, it was able to bridge the gap between the fresh variety created with random generation, and the tactical intrigue able to be created by manually designed levels. More information on the technical implementation of this generation can be found in section 4.5. This system of random generation

At the time, this style of generation began to seem promising. It still had some issues, and often was too subtle to alter the strategic experience for the player, but it gave a level of control that would be capable of being manipulated into the experience we desired, and was able to produce consistently playable levels. Early examples of two levels created from the same base using the design of this system are shown below.

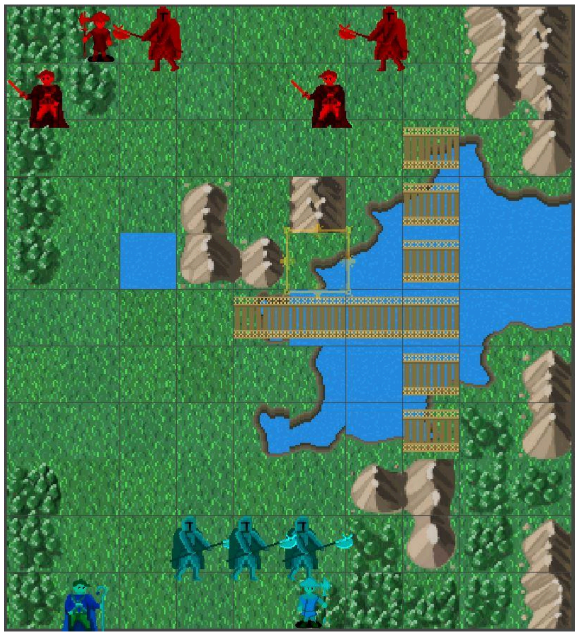
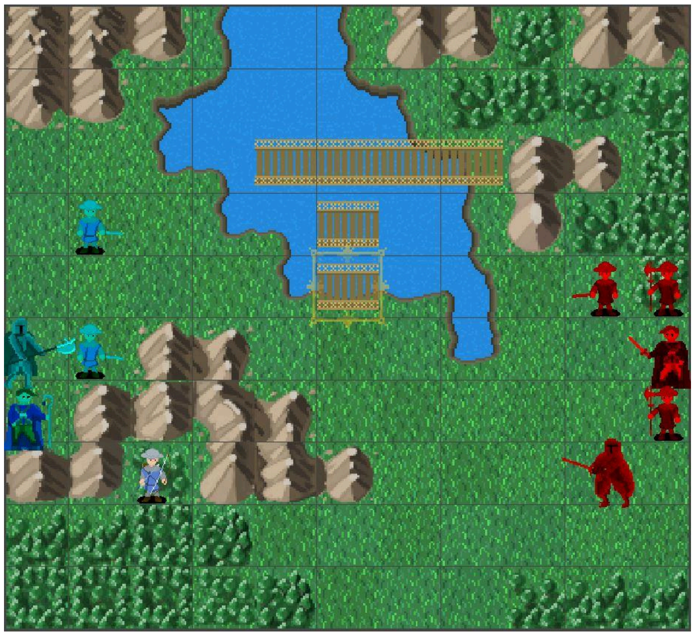



Figure 7 & 8: Two map layouts produced by RavenGuard's layer based procedural generation.

However, during C-Term when analyzing what had yet to be accomplished, it was ultimately decided that this procedural generation system required too much time and effort to tune, bug fix, and get working exactly as we wanted, on top of the extra mass of art assets needed to account for every possible tile pattern. After this decision, effort in level design was shifted to focus on hand-creating levels that would be more interesting to the player, and just enough of them to keep them from getting repetitive too quickly.

Through a mix of iterations and playtesting feedback, the hand-crafted levels made some other changes to the experience goals of the levels. Already foregoing the goal to create a need for constant improvisation in the face of the unknown, these levels also generally upped their scale, creating more expansive areas with more enemies for the player to face down. The greater play-area also offered more space to add chests, which offered more opportunities for the player to gather skills. Previously, levels would offer an equal staging ground between the player and the AI. These newer stages instead opted to out-number the player, creating a larger need, and larger reward for successful tactical planning. The increase in enemies also offered more XP for players to gather to enhance their units.

In order to help the player feel strategic and rewarded for their team building, and to capitalize on the increased space, the new levels have been designed in a way that allows for different styles of team compositions to open up new ways to approach each level. For example, Cavalry units cannot traverse forests but can move much farther per turn than other units, so a few levels offer forested side-routes only accessible to non-Cavalry units, or dangerous zones safest when crossed quickly. Some other levels also offer various gaps and shortcuts that can be tactically abused via proper use of movement skills. As the player, these various paths help recreate the element of uncertainty that was initially lost when switching away from procedural generation. It isn't as much in the forefront as it once was, but these paths mean players must account for bringing skills to keep these options open, or foregoing them in favor of other skills. The image below shows an example of different routes theoretically opened up to the player provided the proper requirements are met.

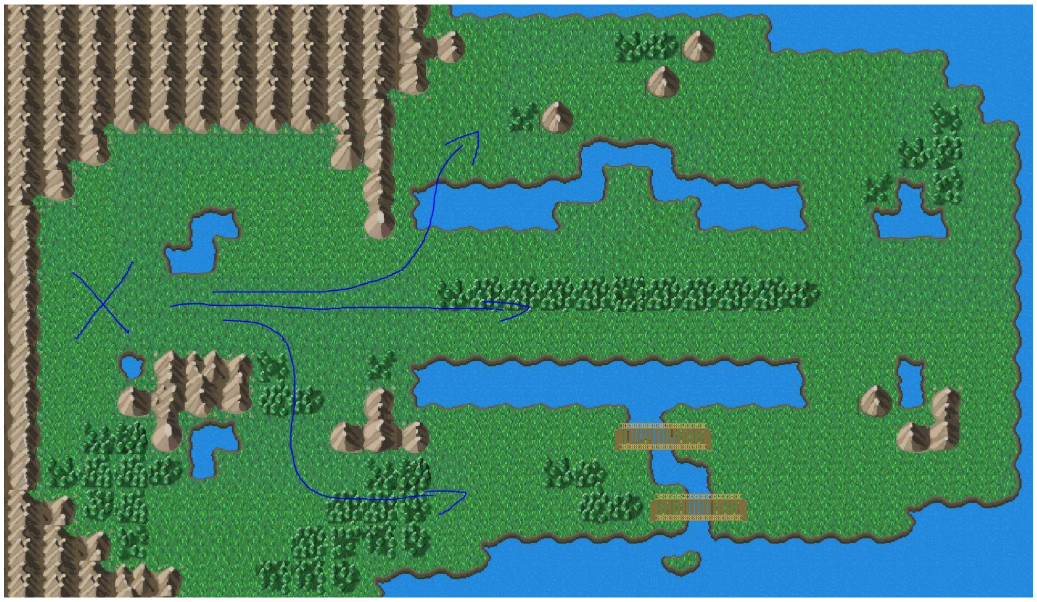


Figure 9: An example of different theoretical routes a player could take through a level.

3.8. Scope

Prior to the official beginning to the MQP, the team met on multiple occasions over the summer to establish a rough outline to the scope of what we wanted to create, and what we might be capable of doing. Once the project officially began, this rough outline got reviewed, and from it a modified MoSCoW prioritization document was born. With this document, the team was able to establish what systems needed to be created, and which would be nice to have but not critical for the game to work. This planning method and the design pillars we outlined would set the stage for what systems would end up in our final product, and which would unfortunately be left behind.

3.8.1. Dream Design Scope

Based heavily on the discussions we had over the summer, we established the scope we would like the game to live up to. When creating this plan, we had considered our capabilities with only 1 programmer, and that was one of the biggest hurdles we would need to overcome during the project. Within this plan, the highest level we wanted to reach would involve the creation of 9 different weapons for units to be able to use, and 7 different classes for units to possibly be. Paragon units and skills never got a fixed number. We figured 5 or so paragons might be a nice balance of variety and not a ton for us to create, but skills we had very little idea.

Outside of units, we wanted the game length to last for three world areas, with each area being a decent size, so runs felt like a satisfying journey even if they always had a fixed length by the end. At this point in production, we knew each world would need a form of final boss, and we hoped that if that went well, we could establish a mid-boss in the middle of the world, in order to keep things feeling interesting and also establish a change of environment during the world, instead of only between worlds. But at the time, we figured that might be a bit much to try and pull off, so the plans for mid-bosses instantly became a should do, not a must do.

Using the random events, tiny bits of dialogue, and some text narration we wanted to try and tell a very simple story too for the player to get into during their runs. *RavenGuard* was always a gameplay first game, so figuring out where to put the story has always taken a bit of a backseat. But it was something we have wanted to implement, even if it was only a little bit.

3.8.2. Streamlined Design Scope

A Term and B Term were mostly spent figuring out how to best approach certain key features, and laying the groundwork for the per-level gameplay loop. While working on these motions and figuring out our productivity, we began adjusting certain numbers we set. Weapon types were simplified early on to be a maximum of four, and unit classes were streamlined to be a maximum of five. One of these classes would end up getting cut during C-Term, reducing the final count to four.

Once the second semester of the project began, we were faced with creating the rest of the game outside of the gameplay for each level, easily concluding then that the scale needed to be rolled back further. With what we had left to do, procedural generation for levels was one of the first things to go. The reward achieved by getting it working properly, compared to the effects of spending that time on other systems wasn't enough to justify us continuing to persevere on getting it working. So we changed our plan to focus on hand-making levels. In regards to run length, the plan was streamlined down, removing random events and multiple worlds, as we felt they were more of a bonus feature and were not necessary to the gameplay experience we wished to create. During this time the plans for cross-run progression also got further designed and thought through, but ultimately would end up severely reduced from offering weapon, skill, and player upgrades on top of unlocking skills and units, to simply offering skill upgrades, and the ability for players to unlock paragon units.

When D-Term began, we were behind where we wished to be, so the scope of the game was reduced once again. This final scope reduction focused on refining what we had, and only finishing what systems were necessary for others to work. It was this reduction in scope that caused the town to focus on simple upgrades and paragon selection. This simplification combined with the larger level design and lack of time to create adaptive scaling also removed the ability for the player to find and recruit new units mid-run, instead allowing them to pick their army from the very beginning. Any remnant of possible story was also removed at this time, as very little planning had gone into it and what time was left was more important to allocate elsewhere.

Ultimately, compared to the lofty goals once set at the beginning of the project, the final result is significantly smaller and more streamlined, which isn't a huge surprise. However, despite losing many mechanics and potential features, we believe the remaining game still evokes the same experience desired and described via our design pillars.

4. Gameplay Implementation

4.1. Tools

4.1.1. Game Engine

Unity was chosen as our game engine because our, at the time, single programmer Alex had the most experience with it. As a single programmer, learning an entire new engine was simply too much extra effort on top of programming the entire game solo. It also fits within our scope of a 2D pixel art game better than other engines, such as UE5. This was decided long before the project officially started at the end of August, but due to some recent terms of service changes made at Unity, other engines such as Godot may have been considered. Though Alex's experience in Unity would likely keep us on that engine for this project.

4.1.2. Version Control

Our project was developed using Git on a public GitHub Repository. We also used Jira to manage tickets, issues, and git branches. Link: https://github.com/AlexMarrinan/RavenGuard

4.2. Controls

The game is designed to be played with a controller You move a cursor around a large grid/board to select units. Once selected a unit can be moved to another tile within their movement range, or use an ability that was previously assigned to them. Once the player chooses an action for a unit, that unit performs the action and their turn is done. The player repeats this until all of their units are done, and then the enemy units will do the same using AI instead of player controlled actions.

4.2.1. Unity New Input System

Unitys "New input system" allows us to more easily define which buttons do which actions. Actions are defined by the types of variables that are read. This includes a simple press action that is either pressed or unpressed for selecting a unit, or a vector2 action that has a positive or negative x and y value to determine which direction the player wants to move the cursor. These actions are then assigned inputs, such as the south face button for select or the analog stick for move. We can add as many inputs per action as we want, so our game will work with any type of controller or keyboard/mouse setup as long as their system is able to use it.

4.2.2. Controller and Keyboard Input

Using a controller is currently the only way of playing the game. You move the cursor around each with either the left analogue stick or directional pad. You can select whatever your cursor is highlighting by pressing the south face button. You can then press the west face button to cancel any selection if applicable. To open the units action menu, press the west face button and navigate it with the previously mentioned move controls. To access your inventory, press the north face button, though you cannot move skills around when in a level. You can press either bumper to automatically highlight the next available unit. These controls work whether you are selecting units on the board or choosing options in a menu. This also means that only one menu can be focused on at a time. When moving the cursor, the camera pans to have the cursor stay at the center of the screen.

Keyboard controls work as an alternative method of input, relying on the same actions set by the controller with the Unity Input system.

4.2.3. Mouse

Using a mouse could have been a viable method of input had we had enough time to implement it. Instead of having a cursor that is tied to moving in a direction, a mouse allows you to highlight any unit by simply hovering that tile. You then press the left mouse button to select whatever unit you are hovering. This is the same for menus, you do not scroll through them like with a controller. but simply hover over the option you want. This also means that you are not locked to one menu, though certain menus like pausing the game will override the whole screen. So you can go from a unit's actions menu to the main board tiles without having to close the menu. But, based on how the cursor is designed for controllers, certain actions will need to be redesigned to accommodate a mouse. Panning the camera to center on the highlighted tile will not work, as the pan will move the highlighted tile off the mouse, thus highlighting a different tile, panning to that tile, etc. So when using a mouse, it will only pan to center a tile that is selected.

Due to the vastly different control schemes between these two input methods, mouse controls were disabled beyond our initial builds to focus on polishing the primary control method. There were also many bugs that appear when swapping between the two control methods that we did not have time to fix. Because of these reasons and others, mouse controls were cut entirely to focus all of our efforts on using a controller or keyboard.

4.2.4. Unit Possible Move Calculation

When a unit can move, it has a given value of the max distance it can move to. This number is then fed into a recursive depth first search algorithm that finds all the possible tiles a unit could move too. This accounts for opposing units that can be attacked and certain tiles that block a unit's path.

Melee Units must be in an adjacent tile to an enemy in order to attack. A sword isn't very long, nor does it shoot projectiles. Stats that affect the attack are based on the units class, equipped weapon and skills etc. These units are simple to implement, just make them move next to the unit they are attacking and you're golden.

Ranged units are a bit more complicated. These units have a minimum range where they are too close to an enemy to attack, and a max range where they are able to attack. Out inital idea was to hen pressing on a unit to attack it, the unit must be moved to a location where they can attack the opposing unit. These are similar mechanics to melee units, but instead of moving them to an adjacent tile, you move them to one that is within their min/max range, which can be many different values. The exact details on how this auto move selection works is still TBD.

4.3. User Interface

4.3.1. Heads-Up Display

For the protofest board scene, there is a tile highlighted by a yellow square acting as a cursor to select units and menu items. When a unit is selected any tile they can move or attack to is highlighted, in blue or red respectfully. When a move is selected the unit is unselected and their moves hidden. There are also mini health bars below each unit that scale regardless of camera zoom/position.

For the protofest battle scene, the heads up display is just the menus shown before the battle, and the two health bars for each unit in the battle. The sprites and their associated animations are not part of the unit but the game world itself. This is important for reasons that will be explained later with the art implementation process.

For alpha fest, we had integrated the battle and board scenes together, so a lot more info needed to be displayed to the player. We added the Units Stats Menu, as it is described in the design section. It is an overview of the units to the top left of the screen. This displays the stats, weapon class, unit class, and equipped skills of the highlighted or selected unit. Two of these will appear if the player is about to initiate combat. The health and stats will change to reflect a prediction of the battle that will occur, more on that in 4.7 Battle Scene.

Beyond alpha fest various improvements were made to the HUD. Red and blue dots were added to the uppermost corners of the screen, to indicate units on either side that were alive with their turn, alive but turn has been used, or dead. A cursor also highlights which dot corresponds with the current highlighted unit.

4.3.2. Menus

Menus are built as a series of icons and buttons. These buttons can be navigated using the move action on controllers or by hovering over them on a mouse. They can be horizontal, vertical, a 2d grid, or a more complex layout to allow for flexibility in design. For example, a unit action menu is a few icon based buttons arranged in a horizontal line at the bottom of the screen in order to not obstruct the view of the board. The pause menu, on the other hand, is 3 large text based buttons vertically arranged and covers most of the screen because the board is not in focus. The inventory screen is a 2 different 2d set of icons representing items in the players inventory and the items equipped by units, and can be navigated on the horizontal and vertical axis'.

4.4. Enemy AI

Enemy artificial intelligence is a feature that is required for our game to be fun in any way. Without enemies to fight, there is no reason to even play. But AI is also a balance, too difficult and it feels like the game is cheating the player, too easy if there's no challenge or thought needed to succeed. So a lot of thought is needed to develop good enemy AI. The AI should feel smart, but not smart enough where the game is never finished. It also should feel unpredictable, not that it is making bad choices, but that the choices it makes are not easily expected and exploitable. At the same time, we need to balance the perceived difficulty with what is actually feasible to implement. Our design pillar here is to clearly communicate choices to the player and make the player want to replay the encounters and try different tactics. In no case do we want the player to hate us, the developer, for making exploitable or impossible enemies.. AI is important but it can be relatively simple, as we had for our protofest board demo.

4.4.1. Initial Implementation

For protofest, we designed a very simple system for our AI. The enemy gets all of its possible movement options. From this list, if a player character is found, go to that player and attack, otherwise choose a random tile to move to. This was implemented at the very last minute and had bugs but was done so that players were not attacking enemies that were just standing still. This proved better than no enemy actions at all, but needed a real solution in the future.

4.4.2. Decision Tree and Rating Moves

After protofest, we started designing AI based around a large decision tree and weighing possible moves. The AI at this point could make 2 types of moves, a simple move to a tile within its range, or to a tile to attack an enemy adjacent to it. These lacked ranged attacks and active skills, which would be implemented at a later date. For every possible move an AI can make, it determines a rating for the move. This rating is based on a lot of determining factors, such as how much health would the unit be at, how much health the opponent unit would be at, would either unit die, would there be other ally or enemy units in range, etc. Once every move was analyzed, one of the top rated moves is chosen and the unit acts accordingly. With some tweaks to our initial weight values and some tuning of the code, this system provided AI that could make fairly strategic movements, albeit with only the simple types of actions. This was the system we used when showing off our game for Alphafest in the middle of B term.

After Alphafest, there were two major ways we wanted to expand upon this system, the first way was to allow units to have specific personalities. Units are given specific personality values based on specific parts of combat, such as aggression, defensiveness, supportiveness, etc. These values change the value of the ratings calculated, giving certain units a higher likelihood of choosing moves that are more in line with their given personality. This feature was intended to be developed in time for alphafest, but a lack of time meant we would need to add it later.

The second improvement is adding the ability for the AI to perform active skills. Unlike passive skills, these skills require you to use your action to perform them, and can possibly be performed on specific tiles. Active skills can also perform many types of actions, such as dealing damage, changing stat values, restoring ally health, etc. Given this, there are a lot of possible places an active skill could and should be used, so an entire new set of weights need to be designed accordingly.

However, unit personalities and active skills for AI would have required a lot of work. Even without these features, the AI was deemed to be in a good enough state for the time and scope of our project . It would not be iterated on much in the future beyond bug fixes and some general tweaks.

4.5. Procedural Level Generation

Like enemy AI, procedural level generation can vary a lot depending on how much time and effort could be allocated to it alone. It also requires many of the core features of the game to work, such as unit movement and combat, in order for us to properly test the generation. When the idea of procedural generation first came up, our first thought was how would it even work. We began researching ideas on just exactly how we could have pseudo randomly generated levels. Our initial design idea was to split parts of the generation into multiple steps. First we would use a noise map to create natural looking terrain, these are the grass, mountain tiles, ponds, rivers, etc. Second we would place natural structures around, many of which are small such as trees, bushes, rocks, etc. Finally we place down artificial structures, many of which define what level you are playing, such as houses, moats, bridges and others. For each step, we also go through a process to make sure that the terrain generated is playable, so that every unit can reach every other unit. If this design proved too difficult or out of our scope, there are other simpler designs we may choose to go down later.

4.5.1. Initial Implementation

This was easily the most difficult process to nail down. Our design went through many iterations before settling on the one seen in the final game.

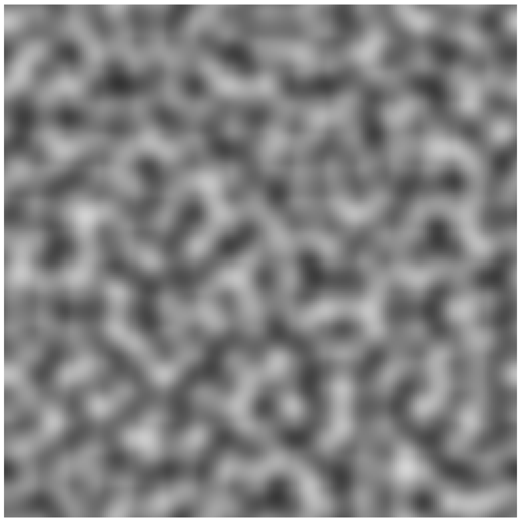


Figure 10: Perlin Noise Map

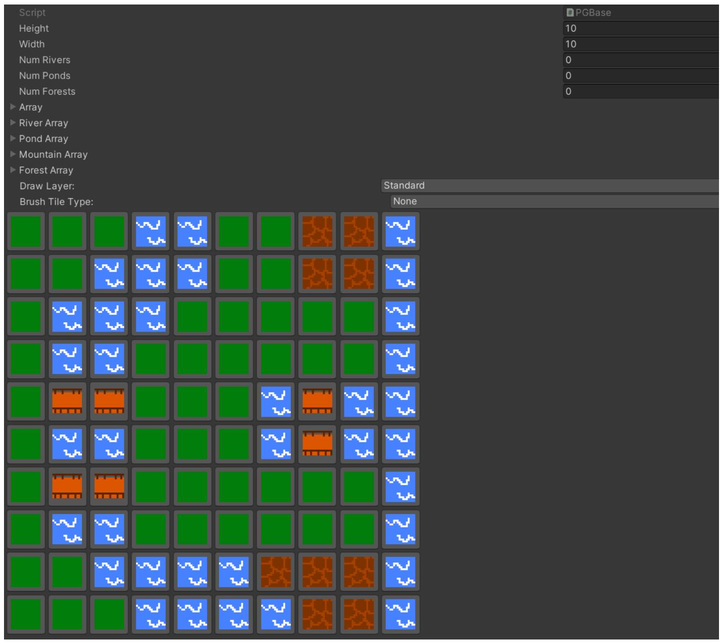
We initially generated a Perlin Noise map, as seen above, which is a large 2 dimensional array of values ranging between zero and one. This map gives us random, but natural looking transitions between the low points (0's) and high points (1's) in order to generate smooth but natural looking areas. We take a very small portion of this map, and use the values to determine our floor and wall tiles. This step was implemented in a basic form for Protofest in October and Alphafest in November.

While the terrain generated seemed to resemble natural looking terrain, it had many issues. The first of which was creating levels that were unplayable, such as a giant wall blocking the two factions from reaching each other, levels that were not beatable. Besides beatable, this random generation also created levels that were playable but not fun. We needed to find a balance between control in how we were creating levels, and randomness to generate levels that felt varied.

4.5.2. Bases and Layers Editor

A noise map, or any kind of pure randomness would let us have very little control over our levels, so we thought about a way for us to define certain aspects of levels that could then be put together. We eventually designed a new system using pieces of already defined terrain to generate our levels. These pieces could be entire level sized, or just a few tiles. With these pieces, we generate a level by algorithmically layering them on top of each other, as well as performing rotate and mirror transformations.

To create these bases, we created a Unity Editor tool to define them within the engine. Below is an example of a base as it is defined in the inspector:



```
Script              PGBase
Height              10
Width               10
Num Rivers          0
Num Ponds           0
Num Forests         0
▸ Array
▸ River Array
▸ Pond Array
▸ Mountain Array
▸ Forest Array
Draw Layer:         Standard
Brush Tile Type:    None
```

Figure 11: Level Editor Inspector Tool

This tool allows us to draw out possible layouts as well as define areas around the level where other layers could possibly be placed. Each tile is given a type, such as grass, water, forest, mountain, bridge, etc., and once all layers are placed and calculated, the correct tiles to render the level are calculated in game. This allows us to combine layers of any type with each other and not have to worry that the tiles wouldn't look right connecting to each other. An algorithm that calculates that a level is playable will also be implemented. While this system doesn't provide a great level every time, it does make sure that each level is made up of parts that are great in a vacuum since we defined each piece ourselves.

4.5.3. Pre-made Levels

Procedural generation was going at a steady pace, but soon many issues came up. The current system could create playable levels, but more often than not these levels were boring or not very good. We estimate an entire MQP could be done to create a worthwhile level generation system, so we cut the feature around B-term of the project. We then shifted towards pre-made, hand crafted levels with a few random elements instead. These decisions allowed for our designers and artists to put a lot more detail into the levels, since the art could be handmade. The largest problem with the hand made levels came about because the art and the actual logic of the levels were now separate. This was good for allowing flexibility of art assets used for specific tiles, but meant we would have to also manually palace the logic tiles on top of the art assets. The logic tiles were what the game actually used to determine where units were and should be able to move. Placing these manually was very tedious and could take a long time just for a single level. However, developing a system to automatically place these tiles based on the art used at the same location would have taken a lot longer and taken focus away from the many other technical features we needed to implement.

4.6. Skills

Skills are abilities that playable and non-playable characters can have in their arsenal. These abilities can be active, needing to be used in combat as an action, or passive, abilities that happen automatically or only after certain criteria are met. Skills can be used to deal damage to enemies, increase your own units stats, decrease other units stats, give status effects to yourself and enemies, etc. Based on this description, it was clear from the beginning that skills would be an extremely daunting abstract task, Any skill could theoretically need to keep track of anything, and do anything, to potentially any unit. They also needed to be use by both the player via the user interface, and by the enemies via the AI. Needless to say a lot had to be considered to get even some of the basic skills implemented.

The first steps to getting skills working was to divide the designed skills into categories. We have active and passive already, but there are many different versions of each base type.

4.6.1. Passive Skills

For passive skills, we have to check at specific points whether a skill should be activated, the most basic of which is when does a passive skill activate in the turn order. We define five major points, at the beginning of a faction's turn, when a unit has moved, when a unit has finished its turn, during combat, and after combat. At these points we check what skills each unit involved has, and check the remaining conditions to see if it should be used.

4.6.2. Active Skills

For active skills, these are simpler to find when to use, we just check if the ability is on cooldown and whether there is a tile it can be used on. This gives us our possible active skill types: use on a tile, use on any unit, or use on a unit of faction. These define the valid tiles we could use the active skill on, from there we can define the shape of the tiles around it to also be affected, which could be just a single tile, a rectangle, a radius, etc.

4.7. Battle Scene

4.7.1. Battle Prediction

When a unit is highlighted in the context of getting attacked, the unit stats menus display a prediction of the damage each unit will take and what their stats will be at during combat. This prediction accounts for all skills and status effects applied to either unit, including combat specific stat changes, which units attack and when, etc. This logic is used during an actual battle, besides critical hit chance, and by the enemy AI to calculate the viability of an attack.

4.7.2. Animations

Animation data is stored inside the unit object. When a battle begins, the unit's animation data is used to display the attack animation during the battle scene. There are two Battle Units on each side, that are translated and sprite animated accordingly. However since each unit faces the middle of the screen, we ran into a problem early on: flipping the animations. Flipping the sprite renderers was easy, as the animation would flip accordingly pixel for pixel. But the translation aspect was more complicated, since the translations would only work for one facing direction. The naive approach was to make two animations per action, which is what we did for protofest when we had a very limited number of units. But a more permanent solution was necessary after. Eventually it was found that having the sprite renderers be child objects, and rotating their parent object, the translations would appear flipped. This worked because the translations were being performed correctly on the child object's local position, but that actual position was now rotated 180 degrees.

4.8. ISP Work

4.8.1. Town Hub World

Developed primarily by Tate Donnelly, the Town required an entire set of new disconnected systems from the base game. We wanted the player to be able to freely move their paragon unit around a little town, while also being able to interact with their environment by speaking with NPCs, entering buildings, and purchasing upgrades. Tate developed an interactions event system that allowed for us to tie any code to an interaction event, such as the previously mentioned examples. They also made a UI system for buying upgrades for skills, and customizing weapons. This allowed us to create the two final buildings used in the game: the blacksmith used for upgrading skills and the tavern used to recruit more paragon units.

4.8.2. Overworld Map

Developed primarily by Ben Levy, the overworld map is used by players between stages to select what stage they want to go to next. The map was originally going to be randomly generated, but Tate could not figure out in time an algorithm that allowed us to create lines with nodes that did not let lines cross over each other. So, when Ben joined in D-term, we tasked him with creating an alternative design. His version relied on preplaced nodes and lines, but each node could be of a random type: either a normal stage or a shop stage. The system also makes sure that no more than 2 shops appear on any single path on the map. The map is also animated, combined with masking to appear as if a scroll is opened up every time it appears on screen.

4.8.3. Particle Effects

Another feature Ben was tasked with working on was a particle effects system. These particle effects appear when a unit is moving around the stage, and during combat to enhance the attack and hit impacts. These particles did not require that much work, but added so much to the feel of the game during combat.

5. Art Development

5.1. Overview

The art style of the game was decided early on to be low resolution pixel art, resembling older Fire Emblem titles such as *Fire Emblem: The Blazing Blade* (2003) as well as older Pokemon games like *Pokémon Ruby / Pokémon Sapphire* (2003) and *Pokémon HeartGold / Pokémon SoulSilver* (2009) . The hope was to mimic the art style of Fire Emblem games while also being more realistic in terms of scope for only 2 artists. The characters, backgrounds, user interfaces, and menus are all to be within this art style with our own touch baked into it. The goal with this art style was to create a gritty and serious tone while also being able to lift these tones to make way for goofier and lighthearted interactions in the game. This mood is similar to the game *Enter the Gungeon* (2016) which we drew from for the more silly aspects while still maintaining the mood of battle.

5.2. Character Sprite Pipeline

5.2.1. Conceptualization

Sprite concepts and inspirations come from both traditional 13th century soldiers as well as Fire Emblem characters and weapons. This is a mix of realism with touches of fantasy with mages, staffs, tomes and silly fantasy designs like knights with capes. The inspiration is then applied to make a 32 x 32 pixel static character for the board and town HUB world and a 64 x 64 pixel version for animation sequences in the battle scene. The aim for the character designs was to reinforce the previously mentioned goal of the art style. The characters had to look gritty enough to be played off as real soldiers, but flexible enough that they wouldn't seem out of place should something more ridiculous take place amongst them.

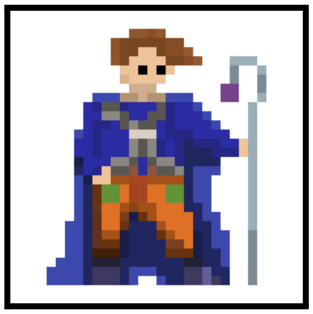
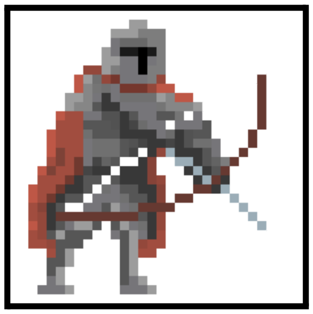



Figure 12 & 13: A 32x32 Mage sprite and 32x32 Knight sprite from *RavenGuard*.

Armor or clothing consists of only one or two layers, both because of the artist's lack of experience in pixel art, but also to ensure there isn't too much noise in the character that makes the limbs and body complicated and hard to understand with such few pixels.

5.2.2. Development Software

The software selected for the creation of the characters is Aseprite. The program is specialized towards pixel art and pixel animation. The program allows for easy creation and manipulation of pixel art which makes the creation of characters simple and easy for the two artists. Aseprite was not without its drawbacks however. The program had a finicky layer vs frame selection system, as well as a missing masking system. These couple of drawbacks caused quite an annoyance whilst using the software, but work-arounds were found, as the software was still the best of any for pixel art and animation.

5.2.3. Creation Technique Changes and Advancements

The artists for this project started out with no experience in pixel art, meaning their techniques for creating characters evolved over time. Characters started out noisy, and details were hard to discern with too many complicated shadows and curves muddying the character's individual body parts. This changed after Protofest. A solidified art style was decided on for all art assets to follow. This art style had standardized character proportions, meaning characters would all have the same shaped bodies. This art style also standardized a shading technique so the characters all seem to fit together. A relatively simplistic shading style was chosen for the characters and standard rules for shading were implemented to ensure readability of limbs and movement. Contact shadows are made on places where two different mediums meet, but depending on the values and hues of the two surfaces, a surface shadow may be withheld to ensure harsh definite lines.

5.2.4. Character Animation

Our process for animating characters for *RavenGuard* evolved as the project went on. Earlier, more limiting techniques were used to animate characters. The first iteration of characters were animated on a single layer, and each frame was individually drawn all at once. This process was eventually improved to the better and faster alternative, that being with layers. The process for animating the characters with layers starts with a 64 x 64 version of the sprite character with their selected weapon. Taking this character, the limbs, weapon, and secondary armor pieces are put on separate layers to allow for better manipulation of the limbs. After the limbs are separated, the character is animated reverse kinematically. First the weapon and its movement, then the hand(s) on the weapon, then the arm attached to the hand, etc. This process continues until satisfactory. A significant portion of the animation occurs in the arms, hands and torso of the character, as with the current system of implementing the animations into Unity, leg movement is minimized to ensure the movement that is added later in Unity is easier and maintains the visual style.

5.2.5. Creation Technique Changes and Advancements

The artists for this project started out with no experience in pixel art, meaning their techniques for creating characters evolved over time. Characters started out noisy, and details were hard to discern with too many complicated shadows and curves muddying the character's individual body parts. This changed after Protofest. A solidified art style was decided on for all art assets to follow. This art style had standardized character proportions, meaning characters would all have the same shaped bodies. This art style also standardized a shading technique so the characters all seem to fit together. A relatively simplistic shading style was chosen for the characters and standard rules for shading were implemented to ensure readability of limbs and movement. Contact shadows are made on places where two different mediums meet, but depending on the values and hues of the two surfaces, a surface shadow may be withheld to ensure harsh definite lines.

5.2.6. Paragon Creation

The paragons were designed to feel more main character like and stand out compared to the normal units. These units' intention is to be unique and define the playstyle of the run, so the design of these units have to stand out significantly, both visually and design wise. The paragons exist as “main characters”, so brighter colors or more complex silhouettes were used to help them stand out.

5.3. Environmental Art Pipeline

5.3.1. Conceptualization

Environment concepts and inspirations come from a few concept art pieces made early on in development along with the mentioned Pokémon games with the forest and mountain landscapes in those games being key inspirations. The original art pieces were directed to try to show a grounded world while also trying to be subtly magical. The environmental art aims to match these pieces making the environment feel like an area where mercenaries would encounter one another in the wilderness or do battle in a castle. Each tile is 32 by 32 pixels and is to be “tileable” meaning each tile fits together with others seamlessly and looks flush in-game. For the battle scene the perspective is top down but very zoomed out as the characters are also 32 by 32 pixels. An example of this is the forest tileset where there are 8-10 trees per tile and the mountains having one or two peaks per tile. The town hub tiles are not zoomed out but are instead to scale of each unit being 32 by 32 pixels tall, for example a tree in the townhub would be several tiles tall.

5.3.2. Development Software

Similarly to the characters, Aseprite was used to create the terrain. It is the main digital art software we used due to the price point and useful tools it has for pixel art. During our time working on this project Aseprite had an update on November 27th to include a more in-depth tool for the creation of terrain tilesets. Sadly this tool was not as helpful as intended but the software itself already has tools that allow for easy creation of tileable tilesets such as the grid and marquee tool which is what was primarily used in the creation of these tilesets.

5.3.3. Battle Scene Tilesets

Initially tilesets were designed with procedural generation in mind. Every possible combination of corners and connections were made in an attempt to create a randomly generated but seamless playing field every time the game started up. This was done for every type of tile type, some examples being grass that is traversable or water that is situationally traversable, or a bridge. Each of these types of tiles have many possible connections with other tiles, meaning a large amount of them is needed. This was done until the end of B Term where the team switched away from procedural generation and into creating the maps by hand. This made the creation of any following battle scene tilesets much easier as they one required a 3 by 3 grid of tiles and allowed for more variations of each type of tile to make each level feel more alive.

5.3.4. Town Hub Tile Sets

The town hub was designed to give a homey feel where the player starts the game and picks which of the paragon units they would like to play as, as well as provide the player places to upgrade their skills and buy more units for their army at the blacksmith and tavern respectively. We used a similar method of creating several versions of the grass and fence surrounding the town to give variation to the environment. The buildings are all scaled based off of the player so there is more detail per tile due to the scale. A base interior tileset was used as the template so all of the interiors looked similar sizes wise. The pathways also use the same method of creation with a 3 by 3 grid of tiles for the edges and corners, the reverse of that so the tile can ‘bleed’ over into another tile, T intersections and the ends of the path in each direction. Thes also have variations with a more ruined look to allow for variation on the pathways.

5.4. UI Art Pipeline

5.4.1. Conceptualization

The UI was designed as needed and based off of other TRPG games like “Fire Emblem: The Blazing Blade” (2003). We utilized 32 by 32 pixel spaces in the UI as locations for where the changing sprites would be like for the skill icons, weapon types and character portraits. The design on the UI is loosely based off of mandalas but with a more magical, fantasy feel.

5.4.2. Development Software

The software selected for the creation of the UI is Aseprite. The program is specialized towards pixel art and pixel animation. The program allows for easy creation and manipulation of pixel art which makes the creation of characters simple and easy for the two artists. Similar to the environmental tilesets, Aseprite’s grid and marquee tools were primarily used in the creation of the UI. The font was initially created in Aseprite as well but used the website Calligraphr to take the PNGs of the fonts created and make them into the two primary types of font files both a .otf and a .ttf which are an open type font file and a true type font file respectively. The software FontForge was then used to edit the .otf and .ttf to ensure the font looked as intended as Calligraphr did not have this functionality.

5.4.3. Initial Creation, Tilesets and Their Pitfalls

The UI was initially created without much inspiration or guidelines as cursors and a basic unit information sheet was created. Creating a font was irritating but once the software was understood there were no issues. To make the rest of the UI creation more uniform and straight forwards a UI tileset was created to give the UI so consistency. This was useful to make a uniform design for the UI but was not perfect due to the variation in sizes of the different menus and UI needed for this project. This required changes to the tileset on a menu by menu basis so each piece would be the size needed while looking similar.

6. Audio

6.1. Initial Issues

Towards the beginning of development, Rose Sand had been tasked as our lead audio designer. However, after A-term Rose had left the project.

With no designated audio role on our team, audio went on the back burner for the majority of the project. We had so many other things to work on that we unfortunately had almost forgotten about it entirely. As the project moved along, we realized that audio was still a necessary part of our game, but that nobody could helm that task to a significant degree. In order to add audio to our game, we would have to rely on outside sources beyond our internal team. We reached out to some potential audio ISP's but to no avail. We did eventually settle on two solutions: stock sound effects and audio commissions from Ben Slattery.

6.2. Music Commissions

Ben Slattery is a WPI alum who helped create all the music present in the game. Using our project funding, we commissioned him to work on the music towards the end of the project. The music he created breathes life into the game that was otherwise not present.

6.3. Sound Effects

While Ben could have also made sound effects, our limited funding was instead used to create a wide variety of music. This meant we would have to go down another route for sound effects. Our final decision was to use free sound stock assets and to edit them to more closely fit our vision for the game. While we did not like to have to use assets that we did not create, none of the main members are striving for an audio portfolio piece. So, it was ultimately the correct choice to at least include some audio in our game.

7. Testing

7.1. Testing Methodology

The testing methodology used throughout this project was adjusted over time as we better figured out what we wanted to see from our testing. Due to the information density accompanying such a game, one thing we determined from the very beginning was keeping an eye out for those familiar with similar games of the genres *RavenGuard* was a part of. Overall, the process evolved from the use of simple surveys to gather opinions, to eventually involving keeping a closer eye on the tester as they made their way through the game and gathering their thoughts mid-playthrough. The following sections further break down the methodologies for testing used, and the results we gathered from them.

7.2. Testing Results

7.2.1. Protofest

The very first time we took *RavenGuard* out for playtesting was Protofest, during A term. At this point, the game was still very early on, so our testing focus was to make sure that our starting work was going in the direction we wanted it to go.

In order to test this, we built two separate gameplay prototypes, and had an array of art assets for testers to review. The first prototype we tested was a movement prototype, which focused on the player moving units around the map. This prototype lacked other systems, instead focusing solely on the beginning of our movement system. We wanted to test if the movement felt deliberate to the player, that they were able to feel in control of their choices. The other prototype we tested was a combat scene, this scene featured no movement, and only combat, allowing the player to simply select which unit from a predetermined batch that they wanted to see fight. This scene involved more art work than it did interaction, and we wanted to ensure that the animations we were designing were entertaining to watch, and made the combat feel impactful. Finally, we had each of our artists create five of the same assets in their own style, then let testers pick which they prefer between the two styles. The goal of this was to find what players preferred to see within the art style, and to help us determine what direction we wanted to take the final style, as it was something we were having trouble nailing down on our own.

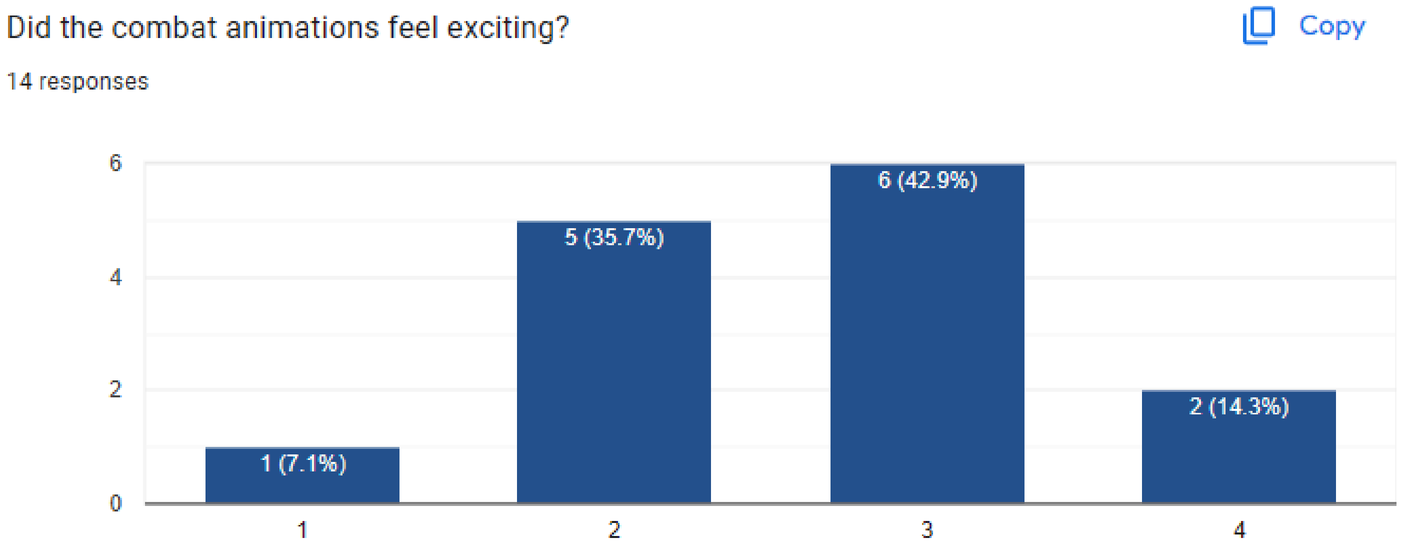
From the individuals who agreed to partake in the study, the feedback gathered was anonymous and confidential, and questions were optional. One of the biggest factors we took in breaking down the feedback we received was if people had played games from similar genres before. While not fully developed, the nature of the game was intrinsically more information heavy and strategic, meaning those with familiar experience or interests in the similar genres would have an easier time, and experience weighing in their answers. It was also important for us to consider the feedback from those with experience too, as we were working to bridge two genres, without compromising what makes each engaging to play. Being able to hear what individuals who play the corresponding genres have to say is made that much more important by that goal.

The movement prototype was overall well received. Our focus here was ensuring the groundwork felt controllable to the player and intuitive, as well as encouraging the deliberate and strategic feel we wanted it to help create. Outside of the Likert scales testers could answer, there was also an open response section for other thoughts they might have. These thoughts fell into two categories. The first was talking about small quality of life things we had either missed, or not enough time to refine. At this point, a unit's movement path when being moved simply followed the cursor, meaning the player could essentially draw shapes instead of being shown how the unit will move. Another, was because of how the map was made up from the technical side, the movement cursor could only be moved over spaces units could also move on. These quality of life comments were changes we focused on amending for Alphafest. The other type of open feedback we got was players wanting a more aggressive, tactical AI. For protofest, the enemy simply moved randomly, just to give the player to move at all. This feedback was more of an effort to implement than the quality of life changes, but it fell right in line with our plans to work on AI soon anyways.

Outside of the movement demo, we had the combat demo and general art feedback. These two focused much more on art. Also, unlike the movement demo, the combat demo had a bit more mixed feedback. Some areas, such as unit stats that were less intuitive and more unique to RavenGuard were an area some people got confused on, and it was what we expected from such a limited experience. The mixed feedback we thought was more important to focus on was regarding the combat animations.

Figure 14: Survey results from one section of the Protofest survey.

Some people agreed with this, some did not. Overall, most people fell into the middle, not being particularly disinterested but also not entirely entertained. Because a lot of the game



will involve the player watching these combat scenes, the goal for them is to be engaging and fun to watch, so the player finds them less of a pain to witness in the long run. When working to alter our art style based on feedback gathered from the art style comparison, we also decided to begin putting extra focus on getting the combat animations to be more entertaining than what we had produced for protofest.

It was also more than just details about RavenGuard we learned from this first round of playtesting. We also learned some flaws in our initial approach to playtesting. Particularly, because a lot of the questions were multiple choice questions or likert scales, they were good for figuring out how people felt, but if it was something we needed to alter based on that feedback, we often ran into wondering why when working to take our next steps. This we ran into particularly a lot via the art preference survey. Some people we had talked to, and they were willing to give us their thoughts in detail, which allowed us to make it through this conundrum we found ourselves in. It was found that between the two artists, certain aspects were preferred from each while also having their own drawbacks. From Michael's characters, the shading was an appreciated part, but it was seen as “noisy” to most people. From Griffin’s art, the eye-catching coloring schemes stood out, however the proportions were not seen favorably. Taking these opinions, the two artists decided on making a mixed art style combining more simplified shading from Michael’s pieces, and the popping colors from Griffin’s. However, for most of the answers to the survey, we had to take guesses on what we could change to get the reaction we wanted. This became particularly troublesome with questions that had answers that were pretty split in where the testers leaned.

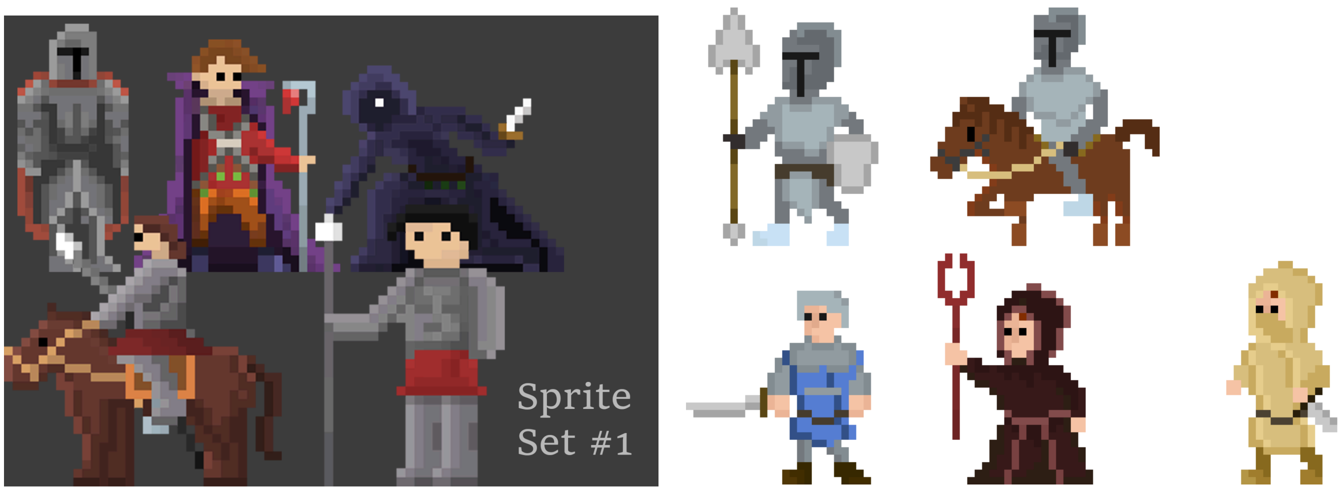


Figure 15: Asset compilations used for the Protofest art survey. Michael's assets on the left with Griffin's on the right.

7.2.2. Alphafest

Protofest marked the end of A term, so our goal immediately after was preparing for Alphafest later in B term. Our major goals for Alphafest were to combine the two major scenes we had for protofest together, create a more challenging and intelligent AI for the enemy controlled characters, and have a basic implementation of the skills system. We initially wanted to allow players to create their own builds with the skills, in order to test if the process of manipulating the skills was fun. However, combining the scenes and getting the skills themselves working took more time than we anticipated and thus our build for Alphafest was scaled back.

Our final result was a build that had the movement and combat animations from the Protofest builds, but also had six new unit types, ranged units, skills for units, and a more developed AI. The very first thing we wanted to ensure was that the experience we wanted out of the parts coming from Protofest were retained, so the first part of the survey repeated many of the same questions. The result from this section was overall the same except for a few parts. Understanding of how the combat played out, what stats meant, and understanding of the controls all went down a bit, particularly the questions regarding understanding combat. This is all likely due to the now increased complexity of these systems, as the now implemented skills not only made combat much less consistent in how it played out, but also added some extra controls that complicated the experience a bit, especially considering there was a few quality of life features lacking from the Alphafest build regarding controls. Some of this confusion is also likely due to the type of game we are creating, as it being a bit more in-depth meant that anyone sitting down for a quick single session had a lot to figure out in order to confidently say that they were able to understand it.

For new features, we were really focused on the skills. They work as a key component to many systems within the game, so we wanted to test them as soon as possible. We wanted the skills to feel impactful, allow for problem solving, and to feel engaging to the player to work with. Generally, the results of these questions were mixed. People agreed that the skill icons designed by art were easily recognizable, and that the skills felt impactful, but were otherwise split evenly between agreeing and disagreeing with the strategizing around skills. Interestingly, the vast majority felt that skills were easy to understand, and it was the engagement and strategy factor that felt lack-luster. Part of this is likely due to the fact that the skills unfortunately had to be made pre-determined for the demo, eliminating any player interaction with determining what they wanted to use. While this mixed feedback is something we plan to keep a close eye on going forward, it is something that is likely partially due to the lack of some quality of life changes we didn’t have time to implement and many restrictions regarding how the player can interact with the skill system that would be removed in future iterations of the game.

Outside of answers to specific questions, the general open feedback we received focused on various bugs plaguing the demo. Post-Alphafest, we focused much of our time amending these issues, leaving some simply because we planned on reworking the systems that caused them as a whole.

Compared to Protofest, Alphafest was much earlier in the term so we had a lot less turnaround time to produce a testable build for Alphafest in four weeks compared to the six we had prior to Protofest. The artists also had to produce significantly more assets in a short time frame so this was a learning curve figuring out how to optimize the workflow. Through repetition we were able to gain more experience animating pixel art and creating new pixel art assets like the unit UI. This did take some time to acclimate to this speed of producing assets however this is necessary as the upcoming terms will require an even greater rate of production compared to A and B terms.

On the playtesting side we have learned we need to differentiate between our play testers as not every playtester provides the same value. Playtesters that have played other TRPGs are much more valuable to our team than anyone else as they are familiar with the systems we are using and can provide valid insight on how we can improve our game. We can make this differentiation by having a survey that offers different questions based on how the playtester answers some initial questions such as, have you played a TRPG or X game before? Then we can offer different questions asking about the combat and movement systems if they have prior experience with similar games and ask about the art style, UI and how easy it was to pick up and understand for playtesters without prior experience. We can also prompt those who do have experience with similar games if they would like to join an email group/chain to notify them when new builds come out so we can retain playtesters and so they can ideally spread it to other players of the TRPG genre.

For preparing for the event we had a good setup with two computers running the build and one computer with the survey displayed so the playtester can give immediate feedback. We also had a sign with the logo and title of the game as well as all of our names on it, a printed out picture of the controller with arrows pointing to the buttons labeling the controls and QR codes for the Itch.io page and the survey if the playtesters would like to take it on their phones. We all have a basic speech about the game to inform the playtesters what exactly they were about to experience. We were well prepared with all of these items as we were able to explain the build effectively, inform those interested what it was, what work we have done and what we plan to do in the future.

7.2.3. D-Term Testing

Later into *RavenGuard*'s development, we decided to have more rounds of playtesting. However, these sessions were approached differently than previous playtests. While the prior playtesting sessions were hosted at IMGD events, theres were hosted on their own. Where prior sessions tested a wide variety of people, these sessions called on people we had identified as being part of our target audience, and those who had experience with other games in the genres we had been attempting to merge. While less players were tested overall, the feedback proved much more valuable since the players were already interested in our style of game. Rather than playing for a short time, these playtests focused more on constant play. Testees would play the game, and asked, if willing, to express their thoughts while playing through the playtesting build. The experiences, and concerns brought up during play were noted by testers, after which the collection of information was parsed by the team into which feedback we felt important, and achievable to amend by the conclusion of the project. There was also an extra optional short survey coupled with this playtesting session, available to those who may have wished to keep their mid-playthrough thoughts private.

At the time of these playtests, the build offered four levels, a similar selection of skills to the build taken to Alphafest, and the ability for the player to customize their army composition and what skills units had equipped. Due to the limited time left, and the similarity this build shared with what we expected to be our final product, the feedback we ultimately focused on centered around fine-tuning the game's overall balance, as well as quality of life improvements. One consistent source of feedback from players was regarding game balance. The majority of players found themselves running into issues when it came to the difficulty scaling of the game. Oftentimes, there was very little reason to avoid fully investing into one unit, leaving the others to intentionally, or unintentionally, be defeated and left behind. Then sometimes, enemy units with skills could be too overwhelming to deal with properly as many found themselves lacking enough skills to keep their units up to par. In further game adjustments, this feedback was taken into consideration when adding a few more skills, increasing the availability of skills, and altering enemy unit stats. The stats of Luck and Attunement also received adjustments. At the time, attunement would be used as the attack stat by magic units, and luck did nothing. Because of this any given unit would either have no use for attack or attunement, because of this, players ultimately concluded each unit had two stats that were worthless to invest in. The fix to this came in adjusting the function of each stat. For luck, simply adding skills that take it into consideration wouldn't solve the problem for units without said skills, so critical hits were added to the game, being a feature that scaled off of a unit's luck stat. Attunement received a major overhaul as well. Magic units would now use the attack stat when calculating damage, and attunement instead became the way units would determine how much damage a critical hit would actually deal. These changes gave these stats a consistent purpose to any unit, regardless of whatever skills might have been equipped.

One major concern of the *RavenGuard* team going into D-Term playtesting was the map size. There were two maps that were a bit larger that the typical map created prior, with one map forcing players on a notably longer journey to its end than prior stages. It was a more experimental choice, and as such was uncertain if it would work for the style of this game. Surprisingly, many players found themselves enjoying the longer experience, with it feeling more rewarding and tactically challenging as the level went on. A few players mentioned that the decision to either spread the effort among units, or keep a few units in the backlines and fresh for the later parts of the level was a tactical decision they felt was intriguing to ponder over. This feedback would inspire more levels created in the future to cause longer play, and more of these attrition style decisions.

The final area of feedback that was focused on was quality of life and the understanding of information. Due to the information, and mechanical density of the game, it was difficult to truly make every piece of information as accessible to players as we had hoped it would be. Certain mechanics, such as being able to get stat and skill descriptions directly from the Unit Stats Menu proved too complicated and unwieldy for us to successfully achieve in the time the project we had left. Even if we believe these features would have been a significant step forward in increasing the ease of interpretation for players, many things had to be cut to keep within our remaining time. However, we still attempted to make whatever adjustments we possibly could to aid in increasing readability for the game. Skill descriptions were overhauled to be easier to read and written more consistently. A "How To Play" section of the menu was added to the pause menu, so the player could access it at any time, and regardless of how intuitive, would at the least provide a consistent way for a player to learn whatever niche information about the game they might want to find. Then, finally, there were adjustments made to the use of ranged attacks and active skills to better communicate what style of effect they have, and make targeting for players easier.

Due to the time of this playtest, not every piece of feedback that the team wanted to act on was able to be fully acted on, but the feedback received by being able to perform a more robust playtest involving a more refined group of testers was able to provide feedback we feel was able to elevate *RavenGuard* to a final level that would have been impossible to do without.

8. Discussion

8.1. Experimental Design Challenges

When playtesting *RavenGuard*, the team ran into a few challenges due to the more niche genre, and thus target audience, the game finds itself with. In the earlier playtests, *RavenGuard* had very few of its roguelike mechanics implemented, instead focusing more on the core gameplay loop which is that of a tactical RPG. The complications this presented came in the playerbase of testees. Whereas many individuals had played a roguelike, a fewer selection had experience with TRPGs, which was an understanding the team found crucial to being able to obtain effective information from a playtest. Later on for the final playtesting session, the team went through the effort of manually establishing a group of playtesters both interested in the game, and with the background relevant to the development of *RavenGuard*.

But this select group had its own fair share of problems. It wasn't until closer to the playtest this approach was decided upon, and **the team wishes that they had done more advertising, and tracking of previous individuals who may have been interested in following the further development of the game.** This would have solved one of the biggest issues with the playtest, which was a slightly smaller number of testees. Being able to match, or ideally surpass the previous playtests would have been a bigger boon to the development of the game.

Ultimately, however, it is worth noting that even with the playtesting feedback received, the team was still unable to completely act on all of it. This was the other big shortfalling that presented itself with the playtesting. Being more proactive with the testing, and ultimately better accounting for the smaller team size to reduce workload, would have been steps that could have aided in making the final playtesting more effective for *RavenGuard*.

8.2. Design Goal Evaluation

The primary design goal seeked to be achieved through the development of *RavenGuard* was to create a seamless blend between the elements of a TRPG and a roguelike. Despite some initial trouble and problem solving involved with figuring out what way to tackle this goal, the team believes they were successful in creating an enjoyable example of such a potential overlap. Feedback from the final playtesting session, and some external individuals, have shown a growing interest in the game even from individuals who are only really interested in one genre of the TRPG roguelike mash-up.

Although, even with this perceived success in combining the two genres, there are some rougher spots regarding the final outcome of the game's overall balance. A vast majority of the designed for skills were not able to make the final jump into implementation due to resource and time constraints the team ran into. The resulting lower quantity of skills creates a negative impact on the player's ability to solve problems and adapt as they progress through the game. This leaves the game feeling more shallow than initially envisioned, and theoretical tactical decisions players could make are more limited in the long run. But this shortcoming doesn't fully compromise the goals of the experience *RavenGuard* provides, merely weakening some aspects of them. Overall, *RavenGuard* is still able to show what the team believes to be a successful example of blending two genres together despite some of the shortcomings that popped up near the tail-end of the project.

8.3. Technological Challenges

One of the biggest problems we had on the tech side was that Alex had worked as a solo programmer for the majority of the project. We had tried to recruit more coders before the project started but none of those that we reached out to accepted any of our offers. For the first semester Alex was the only programmer, but for the second semester we did get Tate and Ben to join as ISP's. Because Alex did the vast majority of the coding, every system had to be designed, and built by him. With a team of even 1 or 2 more programmers, many of the systems could be offloaded to other members of the group which could have brought our game to a much farther spot code wise.

The largest technological hurdle, one that still hasn't been fully crossed, is the skills system. Skills are too abstract of an idea to be able to abstract in a way that will encompass every possible skill design we could come up with. This led to a lot of skills on the cutting room floor, as we had designed dozens but could only include around 30 or so.

AI was also a big technical challenge, especially when it came to testing. Testing and troubleshooting the AI was an arduous and tedious task. Sometimes it wasn't obvious if it had simply chosen something we didn't initially consider a good idea but was actually a smart play on further inspection, or if the code or design had any holes that needed fixing. AI also required a complete rewrite of the unit movement and navigation code. Many systems such as the active skill code however could not be altered enough for the AI to work with it, so it was cut. In the end, we believe we reached a good spot in balance between smart AI behavior while not being too challenging for the player.

8.4. Artistic Challenges

Over the course of this project despite our lack of prior pixel art experience and knowledge of Aseprite we were able to create the assets necessary. This did not come without our fair share of challenges. The primary challenge was learning Aseprite and how to animate efficiently in the program while maintaining a similar art style. This was dealt with through trial and error, communication and the art guide Michael created. One challenge that was not tackled was a uniform color palette. We had never spoken about using the same palette but this was not a huge issue as Michael created most of the animations so they are primarily in the palette he had used which is just the base Aseprite color palette while Griffin and Ian used their own palettes. Proportions of characters were another issue as we were going free hand with the proportions for a while until we got an ISP (Ian) where we made a uniform standard that we would all follow for both the 32 by 32 pixel sprites and the 64 by 64 pixel sprites. Another issue we ran into was the variation between the player units and enemy units. We decided that the player units would be blue and the enemy units would be red and that we would color them in the engine. This provided more difficulty than it was worth as it required animations with multiple sprite sheets stacked on top of one another so we switched to just doubling the animations files by making a red and blue version of each character sprite, sprite sheet and animation. Creating the tilesets also provided a challenge, more so when we had intended to utilize procedural generation of levels. We were not aware of the scale of the tilesets needed to cover every possible connection and combination created so this required several interactions to ensure all tiles necessary were created.

8.5. Looking Forward

At this time, we do not have plans to continue working on this game together. However, this may change given our own situation after graduating. If there are desires to continue the project after this year, there is a lot more we can work and expand on. More content is an obvious choice: a wider variety of skills, units, levels, level themes, etc. A lot of skills never left the design phase, so implementing those in the game would be a big technical job for the future. Our artists could add more units with new animations and designs, as well as create new tilesets for more variety in the theming and look of levels. We can also use the existing and any future tilesets to create more level layouts. The largest working feature we could expand upon is the hub world. We can add permanent upgrades to the player, achievements, skill trees, and more paragon units to join the game. These tasks would require both our programmers and artists to commit, but is a possibility if it's deemed feasible.

References

ChriSX698, "Pokemon HG-SS Sprite Gold", DeviantArt, 25 May 2010 [Photograph].
https://www.deviantart.com/chrisx698/art/Pokemon-HG-SS-Sprite-Gold-165246738

Nintendo (2017) Fire Emblem Heroes (Mobile, Android and IOS Version) [Video Game].
Nintendo EPD.

Nintendo (2012) Fire Emblem Awakening (Nintendo 3DS Version) [Video Game]. Nintendo EPD

Nintendo (2003) Fire Emblem: The Blazing Blade (Game Boy Advance Version) [Video Game].
Nintendo EPD.

Nintendo (2009) Pokémon HeartGold / Pokémon SoulSilver (Nintendo DS Version) [Video Game].
Nintendo EPD

Luca Galate (2022) Vampire Survivors (Steam Version) [Video Game]. Poncle.

Mega Crit Games (2017) Slay the Spire (Steam Version) [Video Game]. Self-published.

Red Nexus Games (2022) Peglin (Steam Version) [Video Game]. Red Nexus Games, Indie Ark.

WAve, "FE8 Red Sun Tileset" feuniverse, March 2020 [Photograph].
https://feuniverse.us/t/fire-emblem-resource-repository-battle-animations-portraits-music-etc/3326/1420?page=15

Appendix A: Protofest Playtesting Survey

Protofest testing made use of two surveys, one for concept art and the other for the two gameplay demos. Each survey was completely optional, with testers only needing to answer the questions they wished to.

Art Survey
This survey used images to see which style of a few select unit types testers preferred.

Multiple Choice
Option 1 or Option 2
1 - Knight sprites
2 - Mage sprites
3 - Assassin sprites
4 - infantry sprites
5 - Cavalry sprites

Gameplay Survey

Experience Background

Have you played any of the following tactical games?
- Fire Emblem
- Final Fantasy Tactics
- Wargroove
- Advanced Wars

Have you played any of the following roguelike games?
- Slay The Spire
- Vampire Survivors
- Wildfrost
- Dead Cells
- Hades
- Inscryption

Movement Gameplay Prototype

Modified Likert Scale
The following questions were answers with a scale of [Disagree 1 - Agree 4]

Did moving around your units feel deliberate?
Did moving around your units make you feel strategic?
Were the controls easy to understand?
Did moving your units feel responsive?
Did moving the camera feel responsive?

Open Response

Any other comments regarding the movement demo?

Combat Prototype

Modified Likert Scale
The following questions were answers with a scale of [Disagree 1 - Agree 4]

Did the combat animations feel exciting?

Did the combat feel impactful?
Were the unit stats easily understandable?
Did the combat outcome make sense?
Did choosing which unit to use feel impactful?

Open Response
Any other comments regarding the combat demo?

ISP Interest

Short Answer
Name
Email
Discord
Area of interest

Multiple Choice
Term of Interest
- B Term
- C Term
- D Term

Appendix B: Alphafest Playtesting Survey

Alphafest testing made use of one survey. The survey was completely optional, with testers only needing to answer the questions they wished to.

Experience Background
Have you played any of the following tactical games?
- Fire Emblem
- Final Fantasy Tactics
- Wargroove
- Advanced Wars

Have you played any of the following roguelike games?
- Slay The Spire
- Vampire Survivors
- Wildfrost
- Dead Cells
- Hades
- Inscryption

ISP Interest

Short Answer
Name
Email
Discord
Area of interest

Multiple Choice

Term of Interest

- C Term
- D Term

Unit Movement

Modified Likert Scale

The following questions were answers with a scale of [Disagree 1 - Agree 4]

Did moving around your units feel deliberate?
Did moving around you units make you feel strategic?
Were the controls easy to understand?
Did moving you units feel responsive?

Open Response

Any other comments?

Combat

Modified Likert Scale

The following questions were answers with a scale of [Disagree 1 - Agree 4]

Did the combat animations feel exciting?
Did the combat feel impactful?
Were the unit stats easily understandable?
Did the combat outcome make sense?
Did choosing which unit to use feel impactful?

Open Response

Any other comments?

Skills

Modified Likert Scale

The following questions were answers with a scale of [Disagree 1 - Agree 4]

Were the skills easy to understand?
Did the skills feel impactful during gameplay?
Were you able to strategize with the skills?
Strategizing around the skills felt engaging.
Were skill icons recognizable?

Open Response

Did any skills feel particularly powerful?
Did any skills feel particularly weak?
Any other comments?