MQP MBJ-1704

BATTLE PATROL:

A Turn-Based Strategy Game

A Major Qualifying Project Report

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in Partial Fulfillment of

the Requirements for the Degree of

BACHELOR OF SCIENCE

in Interactive Media and Game Development

by

Thomas Lourenco and Christopher Dowding

Submitted April 27, 2017

Brian Moriarty and Ralph Sutter, Advisors

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see http://www.wpi.edu/academics/ugradstudies/project-learning.html.

Abstract

Battle Patrol is a two-player, turn-based strategy game that involves a cast of animated cartoon characters in King-of-the-Hill style combat. It was created using a suite of professional artistic and engineering software over the course of several terms. The objective was to create a compact, easy-to-play game that would serve as an art portfolio piece while demonstrating skill in JavaScript programming and collaborative project development. Though modest in scope, the project successfully fulfilled its goals.

Contents

1. Introduction **1**

2. Design **2**

2.1. Inspiration *2*

2.2. Initial technical plan *2*

2.3. Game engine *3*

2.4. Visual style *5*

3. Art production **6**

3.1. Pipeline *6*

3.2 Flash *6*

3.3. After Effects *7*

3.4. Sprite Sheet Packer *8*

3.5. Photoshop *9*

3.6. Tiled *10*

3.7. User interface *11*

4. Gameplay **12**

4.1. Character selection *13*

4.2. Movement *14*

4.3. Combat *15*

4.4. Special attacks *15*

5. Characters **16**

5.1. Character creation *16*

5.2. Rainbow *19*

5.3. Scrap *20*

5.4. Mort *21*

5.5 Slick *23*

5.6. Justice *24*

5.7. Beethoven *25*

6. Playtesting **26**

7. Post mortem **26**

7.1. Project goals *26*

7.2. What went well *27*

7.3. What needed improvement *28*

7.4. What could be added *29*

Works cited **30**

Appendix A: IRB protocols **31**

1. Introduction

Battle Patrol is a two-player, turn based-strategy game. Players take alternating turns to move characters around the map, and attack enemy characters to increase their chances of winning. The cast of characters include two members each of three classes: tank, damage dealer, and support, and are chosen via draft before each game begins. These characters engage in combat in the style of king of the hill; attacks knock enemies away from an objective area that must be held to earn the points needed to win. This style of combat and cast of characters, expressed through the art style of the game, demonstrate a not-so-serious tone that makes the game fun to watch and enjoyable to play.

The goals of this project were to create a capstone art portfolio piece, develop competency in JavaScript, and demonstrate the ability to collaborate in the design and creation of an original game. As an art portfolio piece, the art assets for this game were to be built using professional caliber software, taking advantage of how different tools worked together.

2. Design

2.1. Inspiration

The main inspiration behind this game was the *Advance Wars* series, a collection of turn-based strategy games about rival armies. Beyond the idea of events happening in turns, major similarities include the square tile grid movement system and idea of character selection as a part of strategy. However, important differences include the king of the hill combat system and win condition, cartoonish visual style, and how characters are implemented as part of strategy. Artistic inspirations for this game include works such as *Rick and Morty*, *Castle Crashers* and *No Game No Life.*

2.2. Initial technical plan

The starting plan for the project was to create game that could implement a node.js server for online multiplayer. This initial concept was unfortunately too much for one programmer to handle. Instead the game was kept as hotseat multiplayer for the duration of development.

Despite the node.js plan falling through the project was still coded in JavaScript. This was due to personal preference towards scripting languages, as well as the large number of game engines available in JavaScript. By browsing through the available JavaScript engines the project could use resources that best fit its needs.

2.3. Game engine

Four major engines were considered: PixiJS, MelonJS, Construct 2, and Phaser. Each engine has its own pros and cons, but our selection was made based on the capabilities we expected our game would need, and whether or not the engine could meet those requirements. Figure 1 is a table detailing these factors.

Engine	Tile Map Support	Collision Support	Gamepad Support	Difficulty Of Use	Cost	Flexibility
PixiJS	None	No	No	High	Free	High
MelonJS	Tiled Integration	Yes	Yes	Moderate	Free	Low
Construct 2	Tilemap Object	Yes	Yes	Low	At Least $120	Low
Phaser	Tilemaps using .csv	Yes	Yes	Moderate	Free	Low

Figure 1. Comparison of JavaScript game engines.

The first engine discounted by our selection process was PixiJS. This decision was made because PixiJS is not really a game engine, but a 2D framework for accessing WebGL through an HTML canvas element. It offers many useful features, such as asset loaders and sprite sheet capabilities, but lacks much of the game-specific support that the other engines provide. PixiJS's low-level flexibility makes it an excellent basis for *building* a game engine, but the additional overhead seemed out of scope for a team with a single engineer.

The next engine removed by our selection process was Construct 2, for nearly the opposite reason. Construct 2 stands as the one of the most capable and fully-featured HTML5 game engines available. For our purposes, in fact, it is *too* capable. Construct 2 allows users to create games without any need for coding. While this would obviously reduce our workload, it would also make the project uninteresting from an engineering standpoint, and therefore useless as a portfolio piece.

The remaining two engines, Phaser and MelonJS, were hard to decide between, as they are fairly comparable in features and capabilities. The major difference between the two is their tilemap support.

In the case of Phaser, tilemaps are implemented using a proprietary tilemap object. Maps defined in .csv (comma separated values) format can also be imported. Melon, on the other hand, offers deep integration with Tiled, a tilemap editing application available separately. Maps created with Tiled can be imported directly into a MelonJS project. In order to differentiate between the two options more, an investigation into Tiled was made.

Tiled is a well-established and highly capable editor that allows users to create tilemaps visually, instead of the script-based methods used by other editors. It provides for multiple layers and the placement of game entities, and also supports a variety of viewing formats, including hex grids and isometric perspective. The expressive power of Tiled, combined with its attractive price (free), was the deciding factor in our final choice of MelonJS.

2.4. Visual style

We wanted the visual style of our project to convey and reinforce the fun, not-too-serious attitude conveyed by the gameplay. Our thought process was that if the art style was as humorous and surprising as the events happening in-game, it would create a unified experience that would make the player more inclined to willingly suspend their disbelief.

Figure 2. Branding art for *Rick and Morty* (source URL) and *Castle Crashers* (source URL).

Our main inspirations for this art style were the television show *Rick and Morty*, and the game *Castle Crashers* (Figure 2). These sources were chosen for their ability to engage audiences with their silly characters and illogical narratives, and for our general admiration of their content. On a more detailed level, the character designs for *Rick and Morty* and *Castle Crashers* employ unique outline colors, which serve both to distinguish them and make them stand out. This visual strategy is also used for the animated show *No Game No Life.*

3. Art production

3.1. Pipeline

Our art production pipeline was largely dictated by the choice of MelonJS as our game engine, and Tiled as our map editor. This combination is well-suited for the production of tile-based games, particularly square-tile games like *Advance Wars*, the primary inspiration for our project.

All static images were imported into MelonJS directly via Tiled, regardless of what other program was used to create them. Animated images, on the other hand, need to be imported into MelonJS as sprite sheets. This process of producing these spite sheets proved to be rather time-consuming in practice, because it required the mastery and use of three separate production tools.

3.2 Flash

Although our team's familiarity with Adobe Photoshop made it a tempting option, Adobe Flash was eventually used to create all of the animated sprites for the game (Figure 3). There were several reasons for this choice. First, animations created with Flash are composed of vector images, which can be scaled to any size without loss of quality. This gives Flash a flexibility edge over raster-based 2D drawing programs, such as Adobe Photoshop, which creates images which degrade in quality when scaled up. The time investment required to learn Flash proved to be a wise investment.

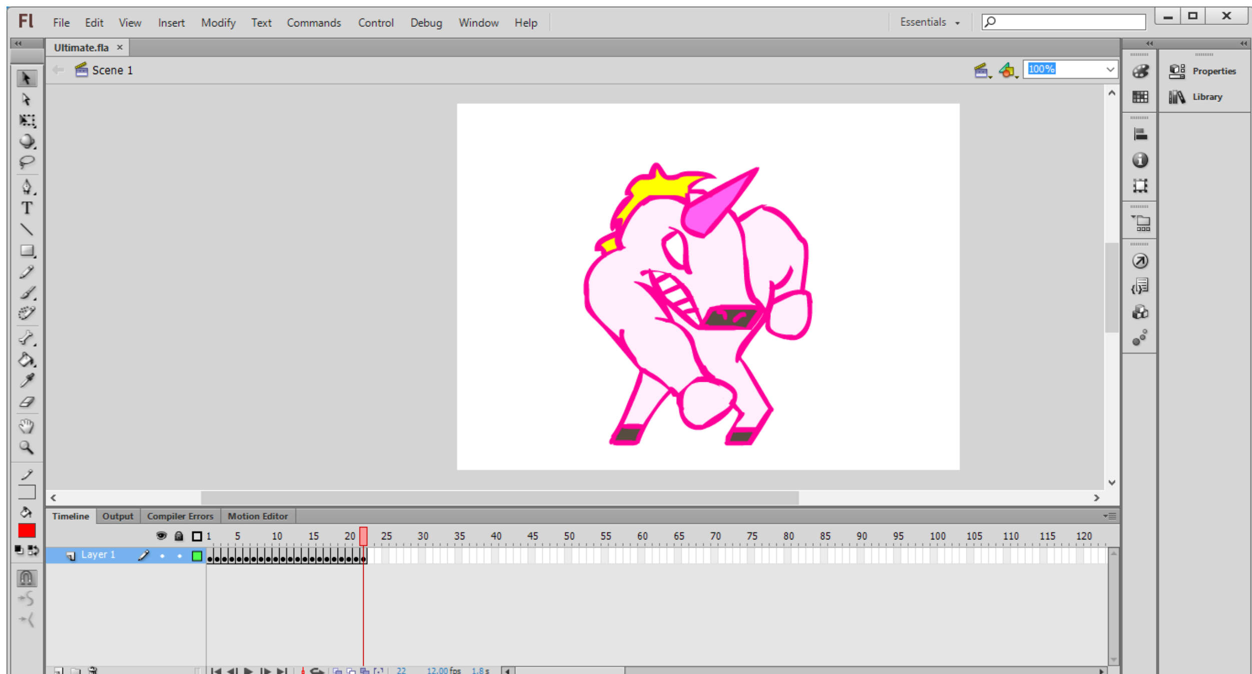

Figure 2. Animating a character with Adobe Flash. Screen capture by Chris Dowding.

Our art advisor, Prof Sutter, pointed out that creating our animations as 3D models (using a program such as Maya) and rendering them as 2D sprites would provide even more flexibility. We decided not to pursue this option because we believed our goal of achieving a fun, cartoonish 2D art style would be more easily realized using actual 2D cartoons.

3.3. After Effects

Adobe After Effects allowed us to bridge the gap between the vector animations produced by Flash and the raster animations required by MelonJS. Among its many capabilities, After Effects allows users to directly import a Flash movie (.swf file) and convert it into a sequence of .png image files (Figure 3). Flash cannot perform this important transformation itself.

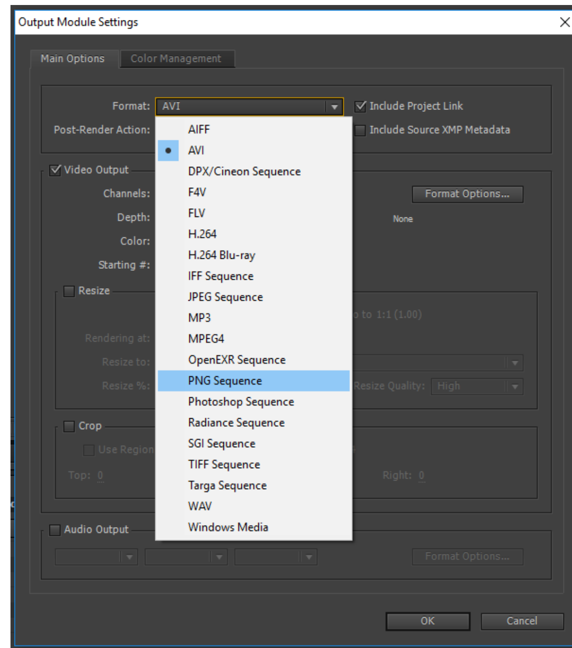

Figure 3. Creating a .png image sequence with After Effects.
Screen capture by Chris Dowding.

Fortunately, both programs are produced by the same company, and are designed to work well together. Taking advantage of this professional-level integration made After Effects an easy and obvious addition to our art pipeline. Furthermore, the visual processing tools in After Effects, particularly its motion keyframing and perspective shift, helped us improve the quality and definition of our animations beyond what would have been possible using Flash alone.

3.4. Sprite Sheet Packer

Sprite Sheet Packer is the simple utility that completed our animation workflow. It was used to concatenate the .png image sequences created by After Effects into a single .png image, or *sprite sheet* (Figure 4). The resulting file could then be imported directly into MelonJS for game integration.

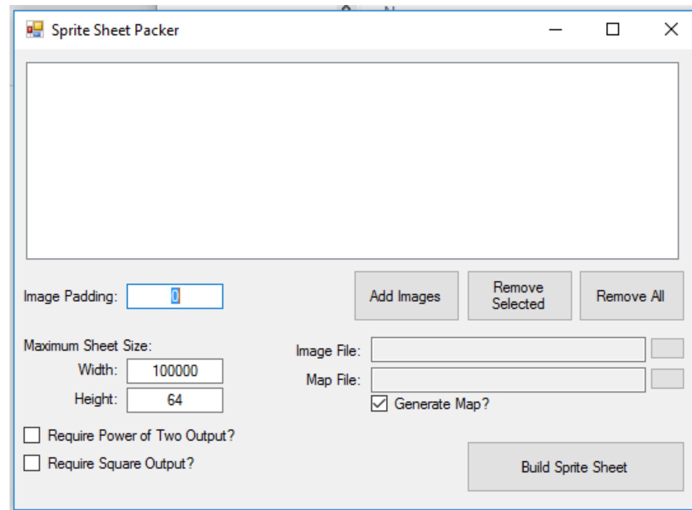

Figure 4. A sprite sheet created by Sprite Sheet Packer.
Screen capture and sprite sheet by Chris Dowding.

3.5. Photoshop

Adobe Photoshop was used to create all static (non-animated) art for our game, including maps, title screens, and user interface elements (Figure 5). This industry-standard 2D paint program readily creates assets in a format compatible with Tiled, which (as noted previously) works seamlessly with MelonJS.

The decision to use Photoshop, a program that creates raster images, in conjunction with Flash, which creates vector images, might appear inconsistent. However, all images must be rasterized eventually for display on a monitor, and our toolchain optimized the quality of rasterization for each type of image.

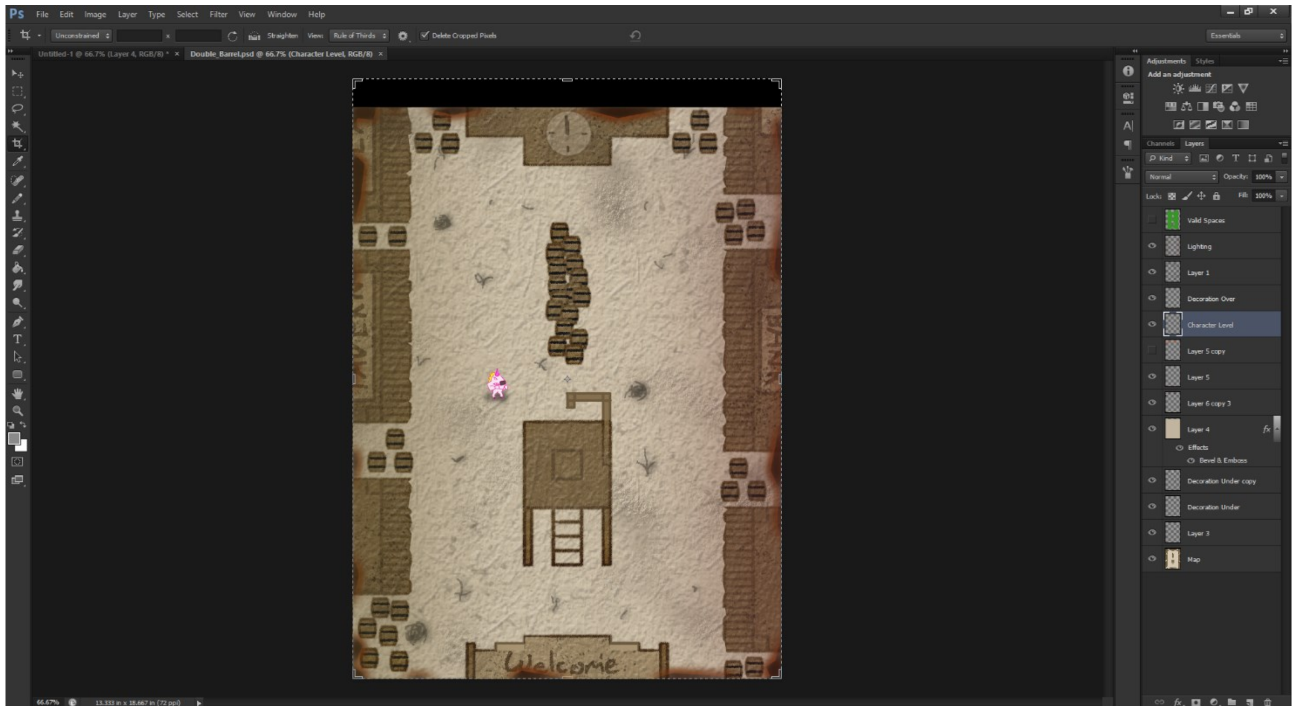

Figure 5. Drawing a raster map in Photoshop. Screen capture by Chris Dowding.

3.6. Tiled

Tiled is a program that converts 2D raster images into a structured tileset that can then be used as building blocks to create game maps (Figure 6). Assets created in Photoshop are easily be imported into Tiled, organized into layers and entities, and then exported into MelonJS for game integration. Our only hurdle using Tiled was our initial lack of experience with it, but this was quickly overcome, as the tool is not particularly complex, and there were only a handful of operations that needed to be performed with it.

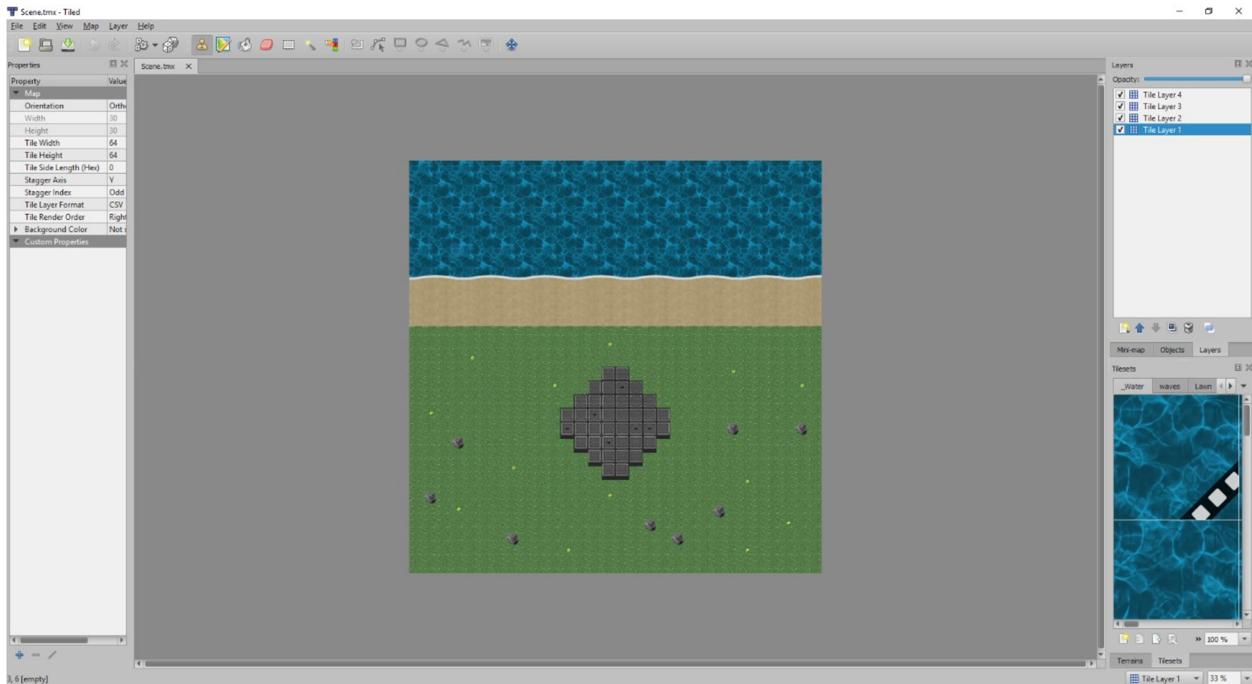

Figure 6. Defining a game map with Tiled. Screen capture by Chris Dowding.

3.7. User interface

The user interface of our project is reflected in the design of the game's logo (Figure 7), which is in turn inspired by the game's six characters. Each character is assigned a unique outline color, indicated by the six colored circles surrounding the logo. The "six" theme continues with the use of two triangles, one solid and one in outline, that combine to form a six-pointed star. Finally, the slashes through the logo are there to reinforce the "battle" motif signified by the game's title. Similar elements appear elsewhere in the user interface, such as in the actions menu, and its six buttons.

Figure 7. Logo of *Battle Patrol*. Image by Chris Dowding.

4. Gameplay

The gameplay of *Battle Patrol* is based on the mechanics of *Advance Wars*, the primary source of inspiration for our project. Both games depend on the strategic selection and movement of units across a square tile grid, and the tactical use of each unit's particular abilities. However, the implementation details vary significantly.

Units within the same class in *Advance Wars* are all identical, and share the same basic statistics. These statistics can only be modified by the abilities of individual characters selected to interact with these units, and any changes apply to all units.

This interdependent system was considerably simplified for *Battle Patrol*. In our design, selected characters serve as self-contained units, each with unique statistics and abilities.

The win condition of our game is also changed from that of its inspiration. In *Advance Wars*, a player wins if the opposition no longer has any units. This goal is enabled and encouraged by the combat system, which allows players to destroy enemy units.

In *Battle Patrol*, the goal is not to destroy the enemy, but to maintain control of a specified objective area, similar to King of the Hill. This mechanic is facilitated by a combat system which does not destroy enemies, but rather knocks them aside, hopefully away from the objective area.

Several other elements of *Advance Wars* were also removed to further simplify the gameplay, including property capture, fund collection, terrain variations, and the deployment of additional units.

4.1. Character selection

Before a game of Battle Patrol begins, players must select the characters that will comprise their team. Six characters must be evenly divided between the two teams. No duplicates are allowed across teams, so all six characters are deployed in every game.

In order to keep character selection fair, players use a "snake draft" to choose their team members. This means that player one chooses first, player two chooses second and third, player one chooses fourth and fifth, and player two chooses sixth. This is the closest to even the draft can get, although it is not a perfectly neutral system. However, the characters and maps are balanced such that any advantage gained in this early stage of play will be negligible.

4.2. Movement

Characters move around the screen by selecting a tile within a certain radius and traversing a path to reach that location. This movement is broken down into three steps.

First, a method is run which allows a selected character to see all tiles accessible to it within a specific radius from their current location. This method uses an array of offset positions to determine whether or not a tile is within a character's walk limit. This array of offsets is a constant value, so affiliating a character's offset with locations in the array is relatively easy. Then an A* pathfinding algorithm is run to calculate which tiles a character can legally reach. All locations passing this check are highlighted for the player to inspect.

Next, the player selects one of the highlighted tiles to initiate character movement. With the desired tile selected, the A* algorithm is run on the tilemap again with the character's current tile as the start position and the target tile as the end position. The resulting path is then added to the character's path variable, which contains a sequential list of all the tiles in the path a character's is intending to follow. When a character is not moving, its path variable contains only the tile on which it is standing.

When the character's path variable is finalized, its update function begins moving the character, continuously checking if it can legally reach the next tile in its path, and if so, which direction it must take to reach it. When the length of a character's path variable reaches a value of one (1), movement stops and the character's animation reverts to standing.

4.3. Combat

Basic combat executes the same way for all characters, but the effect depends on the interaction of two variables specific to each character: strength and weight. These values are used to calculate how hard an attacker's hit will be, and how much effect that hit will have on the victim. The attacker's strength is divided by the victim's weight and floored to 1.

First, however, the player must determine whether they are within attack range of any enemy characters, and if so, which one to hit. This is done by opening the radial menu associated with each character. If any enemy characters are within range, the attack buttons for those enemies becomes selectable. The player then selects which enemy character they want to attack, and the hit is executed.

Since the knockback creates a path, we set the path of the victim and set the state to knockback. The update function then executes the knockback move.

4.4. Special attacks

Each character in *Battle Patrol* has a special attack, or some other unique ability. On every game turn, each team accumulates power for their characters' special attacks. When a team has enough power to activate a specific character's special, the player can select the special attack button from the character's radial menu. Similar to the basic attack function, the special's area of effect is highlighted, and the player can select it to trigger the ability. The special abilities of each character are described below.

5. Characters

There are six characters in the game divided into three classes, with two members per class. Because characters are drafted with no duplicates allowed across teams, it was important to have two characters capable of filling the same role. However, there was still a need to keep individual characters within each class unique. This was accomplished through variations in movement and combat statistics, and also through the unique functionality of each character's special attack or ability.

The reason for offering three classes is to encourage players to fill the three slots in their team roster with one character from each class. The game's intended balance is that a diverse team will do better than a homogeneous one.

5.1. Character creation

Character creation was one of the major artistic challenges of the project. The two areas of focus were idea generation/concepting, and maintaining a cohesive style across all characters while drawing inspiration from multiple sources.

Our first method used to jumpstart idea generation was using an existing work as a reference for creating original characters. This method was used to create the character Rainbow, with *Rick and Morty* as the reference (Figure 8). The process was applied by imagining what sort of character could inhabit the existing world of the show and not look out of place. This way, the unique elements of *Rick and Morty*, such as its quirky cast and unlikely interactions, could be captured within an original character. This method was successful with Rainbow, but it proved difficult to duplicate this success.

Figure 8. Rainbow, the buff unicorn inspired by *Rick and Marty* (source URL).
Original image of Rainbow by Chris Dowding.

The second method, used to escape roadblocks created by the first, was more direct in its use of reference material. The intent was to study existing media, particularly other games, to identify character archetypes that could be adapted to our project. This method led to the creation of several characters, including Slick, Mort, and Justice. Slick and Mort were inspired by the game *Crawl*, which has players use ghosts to take over various monsters, including slimes (Figure 9).

Figure 9. Mort and Slick, both inspired by the game *Crawl* (source URL).
Original images of Mort and Slick by Chris Dowding.

Our ghost, Mort, is a highly useful character capable of moving through both walls and enemies. Similarly, Slick's perilously sticky slime trail is an attempt to reverse expectations players might have about a character commonly regarded as a low-level monster.

Figure 10. Beethoven, inspired by heavy metal music by bands such as Megadeth (source URL). Beethoven image by Chris Dowding.

The final method of character creation was used to prevent our characters from becoming too derivative of other works. It is based on the idea "write what you know," a method to overcome writer's block that involves describing objects, events or situations that are familiar due to real-world experience. Our character Beethoven was developed from Chris' participation in hard rock and heavy metal music communities, events, and his general knowledge of the genres (Figure 10). This method made rounding out the cast of characters relatively easy, and put a lot of confidence behind their designs.

5.2. Rainbow

Rainbow was the game's first character to be fully fleshed out, and the comically stark contrast of his build and demeanor versus his name and color was the inspiration for other features in the game (Figure 11). This includes how the other characters would be designed and named, as well as how the overall tone and feel of the game would develop and evolve over time.

Figure 11. Rainbow. Image by Chris Dowding.

Rainbow's intended in-game purpose is to be a tank class character. He is more resistant to attack than other characters, and is able to stun enemies for a turn within an area of effect with his special attack.

Because Rainbow's special attack is rather powerful, he was given a low movement statistic to compensate for what might otherwise be a dominant ability. This also helps to characterize Rainbow beyond his visual appearance; his size prevents him from being fleet-footed, and his hallmark ability shows off his perceived strength. This was important, because although his design

indicates that he is a strong character, he is not well-suited for direct attacks. He is a tank, not a damage dealer.

Figure 12. Scrap. Image by Chris Dowding.

5.3. Scrap

Scrap is the second tank-class character in the game (Figure 12). He is one of three non-humanoid characters in the game, and one of four non-humans. The variety of this unusual cast of characters is another manifestation of our inspiration from *Rick and Morty*.

Like Rainbow, Scrap can take a lot of damage, but cannot move very quickly. His special attack is also intended for crowd control: it pushes all targets within its area of effect away to a fixed distance from the center. However, because it does not stun those affected, Scrap's special attack is somewhat weaker than Rainbow's. This difference is made up for by giving Scrap an edge in most other statistics.

Once again, the special attack is a place for characterization. The charge up and self-destruct can only belong to something made of machinery, and was a perfect fit for the game's tone and sense of humor. This sense of humor is also present in the character's name: calling a perfectly functional robot "scrap."

Figure 13. Mort. Image by Chris Dowding.

5.4. Mort

Mort is one of the support class characters, and was the final character to be concepted (Figure 13). As an original goal, the cast of characters was supposed to be diverse, with two humans, two humanoids, and two non-humanoids. Mort was intended to fill the final missing slot as a humanoid, but this is not how his design ended up. At the time when Mort was being designed, the sense of humor and tone of the game had already taken shape, and the simple shape of a cartoon ghost fit much better than a detailed humanoid ghost. Furthermore, his large, expressive eyes and transition from docile to scary during his attack animation made Mort's addition to the game an easy

one. Finally, keeping with our humorous theme, Mort's name was derived from the abbreviation for the name Mortimer, and also the prefix mort-, meaning death.

As a support character, Mort does not have a special attack that can charge up, but rather a passive ability. Mort is able to pass freely through other characters and objects, just as a ghost should. This ability, combined with his high movement statistic, allows him to contest objectives and easily break through defenses.

Mort takes more damage than average, which counters his ability to easily get to and contest objectives, but still forces opponents to commit an attack to deal with his presence.

Figure 14. Slick. Image by Chris Dowding.

5.5 Slick

Slick was the first "non-humanoid" character to be designed (Figure 14). During character brainstorming, the idea of a slime, a common low-level enemy in role playing games, came to mind as a creature that had no obvious human or animal analog. Furthermore, slimes are not often employed as playable game characters. We saw Slick as another opportunity to use a stereotype in an unexpected way.

Slick's function as a support character is to move around as much as possible, leaving behind a trail of sticky slime that costs enemies an extra movement point to walk through. This ability is countered by the opposing team's ability to easily knock Slick away. Furthermore, Slick's movement statistic is the lesser of the two supports, and short enough to prevent entire sections of the map from being walled off behind a slime trail.

Figure 15. Justice. Image by Chris Dowding.

5.6. Justice

Justice was one of the first characters to be designed, and was largely inspired by *BANG!,* a western-themed card game (Figure 15). The name "Justice" is a variation of the ironic naming convention that started with Rainbow. Originally, it was intended that his name would be Sheriff Justice, a comically redundant name that found inspiration from the character "Chief Over Justice" in the cartoon show *Space Patrol Luluco*. However, the name was shortened to just "Justice" as the his design evolved into more of an outlaw than a sheriff. The need to distinguish this character as *not* being a sheriff was prompted by the context of the game's setting. It makes more sense for an outlaw to be brawling than a law enforcement officer.

Justice's intended functionality is to be a team's primary offensive force. As a member of the damage-dealing class, Justice's statistics hover around average, except for his ability to knock

targets away from objectives, which is one of the highest of any character. His fearsomeness is also reflected in his special attack, which permits him to hit any single target at any range. His target is selected with a large crosshair, which seems appropriate for a sharpshooter.

Figure 16. Beethoven. Image by Chris Dowding.

5.7. Beethoven

Beethoven is the second main-offense character in the game (Figure 16). From beginning to end, the overall design of Beethoven did not change much, but he received a significant alteration during the development of his animations. Specifically, there was the problem of how to show Beethoven attacking with a guitar slung awkwardly over his back. Moving the guitar into a normal playing position more clearly communicated the intention of his design, and also allowed a more fluid attack animation, in which sonic waves are emitted by the guitar as it is played.

Beethoven's strategic role is similar to that of Justice. Both excel at doing damage, but neither receives it very well, or moves very quickly. For his special attack, Beethoven's guitar emits

sonic waves in a line capable of knocking multiple targets back a significant distance. This ability establishes Beethoven as a character who loves to play and listen to music. *Loud* music.

6. Playtesting

Unfortunately, prior to the presentation of *Battle Patrol*, only a minor amount of playtesting was completed. It wasn't until the following week that proper playtesting began to take place, and improvements could be identified and implemented.

Our hope is that playtesting will help us to balance the speed, weight, and strength of the characters so that all of them feel active and useful. Additionally, the relative strengths of each character's special or passive abilities need to be carefully calibrated.

7. Post mortem

7.1. Project goals

From an art standpoint, the goal of this project was to create a capstone portfolio piece that could be used to successfully acquire a job in the game industry. Furthermore, it was hoped that the project would provide an opportunity to develop and demonstrate skill with professional software packages, the ability to create a full game from scratch in a team environment, and the flexibility to working within limitations. Finally, this project allowed for team based cooperation and mentorship to prepare for a workplace environment.

On the technical end of the project, the goal was to create a strong portfolio piece that used a node.js server. In addition, a stronger understand of JavaScript would result with completing this

goal, as well familiarity with a strong JavaScript game engine. The final goal was to work in a team where a significant amount of art had to be implemented over time, requiring placeholders and adjustments to facilitate art incorporation.

7.2. What went well

Some aspects of the art pipeline worked very well. Specifically, the latter half of the pipeline behaved very smoothly. This includes the postproduction and compositing work done in After Effects, as well as the process of compiling sprite sheets with Sprite Sheet Packer. This can be attributed to experience and familiarity with the After Effects environment and the very simple interface of Sprite Sheet Packer. This latter half also includes the use of Tiled for bringing static assets into the game engine, and its ease of use can likewise be contributed to its clear interface.

The entire process of creating maps was indeed straightforward and had very clear motivations behind the evolution of the maps' style. This process was not only made easy by Tiled, but also by familiarity and experience with drawing images in Photoshop. The evolution of map design was also a straightforward process. This is because there was a clear need for the maps to match the art style of the characters as they evolved. Furthermore, maps needed to change in such a way that made characters easier to read off the background.

The game engine selection was very good and therefore resulted in a stronger understanding of software tools, but also JavaScript itself. At the end of the project, when cleaning up old mistakes, it became very clear that higher quality projects would not be a stretch in the future, due to what I learned during this process. This also includes the use of the MelonJS engine, which is a great tool for producing JavaScript games.

7.3. What needed improvement

The primary obstacles of the art pipeline of this project revolved around the character animations and character concepting. The difficulty with animations was primarily because of an inability to quickly devise an efficient pipeline, and a lack of familiarity and experience with the tools used to create them inside Flash. This included issues with using poor choice of tools, such as Photoshop, to create animation frames, and trying to interject 3D assets into a 2D pipeline, making the entire process more time consuming, less efficient, and less valuable. Gradually, this process became easier as experience was gained over the course of the project. The animations were also difficult to critique due to the fact that they were not often able to be seen in game with the proper maps behind them. This situation itself arose from backups in other areas of development.

Issues with character concepting largely revolved around difficulty with idea generation and finding a cohesive style across the characters. The problem with idea generation was tackled by looking to reference and inspiration for assistance with three methods: (1) Using an existing work's environment to create a character that would belong in such an environment, (2) Borrowing character archetypes from other works and adding a unique aspect or two and (3) Drawing upon personal experience to create an original character from scratch.

As for creating a cohesive style, this problem was solved with the switch from Photoshop to Flash, which allowed for much cleaner lines and eased the creation of a cartoon style that would have otherwise been a struggle in Photoshop.

From a technical standpoint, the JSDoc comments could be improved significantly. Early in the development it was determined that JSDoc could be used to produce strong documentation for the final project, but an issue arose with not keeping the proper comments up to date. Due to this, in order to properly document the code, hours were spent reviewing and adding comments. Instead

of this, it would have been much better to spend at most a minute, with the creation of each new variable or object, and create the JSDoc comment.

Additionally, any time a method of execution in the code was changed, refactoring should take place immediately. On multiple occasions, code had to be reviewed just to determine that it was one of three or four different ways a task was being handled. If time is spent refactoring everything to a set standard, then this confusion would not arise, and cleaning of the code would not need to occur immediately.

7.4. What could be added

From an art standpoint, there are plenty of areas that could be improved upon. First, there could be a wider selection of maps to play on. Maps were seen as a lower priority during the course of the project, so it was not possible to implement as many as originally hoped. Also, the addition of more particle effects to go along with normal attacks, and perhaps movement would have been a welcome improvement. Finally, it would benefit the game to have more sounds that not only give meaning to an action, but also serve to characterize the locations, characters, and actions being performed.

Since the node.js server was never implemented, it would be an obvious addition to this game. The theory stands that since this is a turn-based strategy game that a server would simply need to inform one client of events that occurred on the other, without much concern for lag or latency. It would also allow features outside of gameplay, such as analytics and chat to be incorporated into the game and improve the experience.

Works cited

Art and engineering tools

Adobe Creative Suite (Flash, Photoshop, After Effects): http://www.adobe.com/

Construct 2 Game Engine: https://www.scirra.com/construct2

JSDoc: http://usejsdoc.org/

Phaser Game Engine: https://phaser.io/

PixiJS WebGL Renderer: http://www.pixijs.com/

MelonJS Game Engine: http://melonjs.org/

Sprite Sheet Packer: https://spritesheetpacker.codeplex.com/

Tiled Map Editor: http://www.mapeditor.org/

Appendix A: IRB protocols

Informed Consent Agreement for Participation in a Research Study
Investigator: Brian Moriarty
Contact Information: bmoriarty@wpi.edu
Title of Research Study: MQP MBJ-1704 Interactive Storytelling

Sponsor: N/A

Introduction (recommended)
You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: The purpose of this study is to obtain feedback on the above-named MQP project in order to facilitate design improvements and find/address operational bugs.

Procedures to be followed: You will be asked to play an entertainment game lasting 10-20 minutes. After completing the game, you will be asked to complete a brief, anonymous survey describing your subjective experience.

Risks to study participants: There are no reasonably foreseeable risks associated with this research study.

Benefits to research participants and others: You will have an opportunity to enjoy and comment on a new entertainment game under active development. Your feedback will help improve the game experience for future players, and its published MQP report will lead to an improved understanding of the game design/development process.

Record keeping and confidentiality: The survey you complete as part of this research will contain no personal information. It will be secured in a password-protected database, and deleted at the conclusion of the project.

Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators, the sponsor or its designee and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: There is no reasonably foreseeable risk of injury associated with this research study. Nevertheless, you do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact the Investigator listed at the top of this consent form. You may also contact WPI's IRB Chair (Professor Kent Rissmiller, Tel. 508-831-5019, Email: kjr@wpi.edu) and the University Compliance Officer (Jon Bartelson, Tel. 508-831-5725, Email: jonb@wpi.edu.

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

______________________________ Date: ____________________

Study Participant Signature

____

Study Participant Name (Please print)

____________________________________ Date: ____________________

Signature of Person who explained this study

Addendum for WPI IRB Application Form

Title: MBJ-1704 Interactive Storytelling

Purpose of study

To obtain playtest feedback in order to locate/address operational bugs, and to identify opportunities for design improvement.

Study protocol

Participants are provided a computer on which to play the game. Investigators observe participants during play. Afterward, participants are asked to fill out a short survey to characterize their subjective experience.

Hazardous materials/special diets

No hazardous materials or special diets are involved in this study.

Opening briefing for testers

"Hello, and thank you for volunteering to test our game. Before we begin, could you please read and sign this Informed Consent form? [Tester signs IC form.] Thank you. When your session is complete, we will ask you to complete a brief survey about your play experience. At no point during your play session, or in the survey after, will any sort of personal and/or identifying information about you be recorded. Please begin playing when you feel ready."

Questions for post-test survey

1. Does this game seem similar to any other games you have played? If so, name them.
2. What are the game's rules and objectives? How did you discover them?
3. How would you rate the level of challenge presented by the game?
    - (1-4 scale, 1 = Difficult, 4 = Easy)
4. How would you rate the game's user interface/ease of use?
    - (1-4 scale, 1 = Difficult, 2 = Easy)
5. How would you rate the visual design of the game?
    - (1-4 scale, 1 = Unattractive, 4 = Highly attractive)
6. How would you rate the sound design of the game?
    - (1-4 scale, 1 = Unappealing, 4 = Highly appealing)
7. Did anything about the game seem particularly confusing or obscure?
8. Did any aspects of the game stand out as particularly effective or ineffective?
9. How would you describe the game to someone who has never played it?
10. Any general comments or questions about the game?