Storybook - A Casual Game

Emory Blackstone, Nathan Bryant, Benny Peake, Connor Porell

April 27, 2016

A Major Qualifying Project Report:
submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

Emory Blackstone
Nathan Bryant
Benny Peake
Connor Porell

Date: April 2016

Approved:

Professor David Finkel, Advisor

Professor Britton Snyder, Co-Advisor

Abstract

Acknowledgements

List of Tables and Figures

Abstract

The purpose of this MQP was to construct a small-scale game such that we would have adequate time to properly balance and polish it. The prompt of a casual game led us to design a cooperative multiplayer dungeon crawler/collectible card game hybrid, something we have never seen in a casual game before. Our goal over the course of development was to keep scale manageable while also creating a unique game that we could polish and proudly show off.

Acknowledgements

We would first and foremost like to thank our advisors, Professor David Finkel and Professor Britton Snyder. Their creative guidance was a great asset throughout the course of development.

Additionally, we would like to thank Dillon DeSimone and Francesca Carletto-Leon for their assistance with artwork and gameplay design, respectively. Though they were only available to help us for a single term, all the work they put in was greatly appreciated.

We would like to thank Louis Alexander, a student at Berklee College of Music, who composed the soundtrack to *Storybook*.

Lastly, but most certainly not least, we would like to thank everyone who playtested the game or left the slightest bit of critique on *Storybook*. You may not know it, but we took every piece of critique seriously, so each one of you helped to shape *Storybook* into what it is today.

List of Tables and Figures

1. Introduction

A. Casual Game Prompt

A casual game is typically defined as a type of video game where the "[player] does not have a long-term commitment to a game and can approach playing the game on an infrequent and spontaneous basis" (1). Most casual games are played for only a few minutes at a time before the player puts it down (2). As such, the typical casual game is more of a time-killer rather than a sit-down-and-play type of game. Many people often play casual games to pass a few spare minutes such as when waiting for a bus or train, or as a small diversion while on a lunch break. Some popular examples of casual games are *Clash of Clans* (3), *Angry Birds* (4), and *Trivia Crack* (5).

B. Inspiration

Early in the design process, we brainstormed for Genres and high-level gameplay concepts, drawing from some of our favorite games. A common theme among the team was the desire to create a dungeon crawling game, similar to *Rogue* (6) or *The Binding of Isaac* (7). While uncommon if not nonexistent among casual games, dungeon crawlers have the potential to be remade into the casual game format. By reducing the scope of the game to use smaller and fewer floors/levels, introducing the ability to save in the middle of the dungeon, and toning down the infamous difficulty common among dungeon crawlers, we drew up a plan to make a dungeon crawler game more accessible and playable in quick bursts.

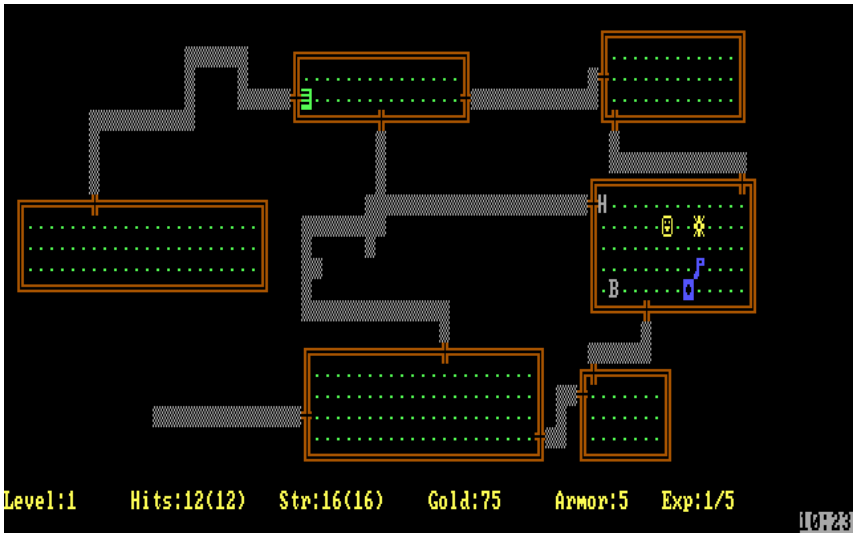


Figure 1: Screenshot of Rogue. *Dungeon crawlers such as* Rogue *were common among inspirations for* Storybook.

Another early inspiration for us was the desire to create some sort of ensemble cast uniting characters from various backgrounds, such as Nintendo's *Super Smash Bros.* (8) series. When we were thinking of characters we wanted to recreate in our game, we had wildly different tastes. The concept of an ensemble cast featuring characters of different backgrounds was a key component in choosing the setting and world of *Storybook.*

Figure 2: Ensemble casts in video games. One of the most famous examples is Super Smash Bros. (9)*, a fighting game that features a host of characters from various Nintendo franchises*.

The idea of the crossover story was another big inspiration for us. These involve characters from different stories traveling to each other's worlds so that they can interact. These kinds of stories have been around for ages, and are especially popular in things like superhero comics like Marvel's *Avengers* (10) as well as in other forms of pop fiction. Crossovers are really fun to play with from a creative standpoint because it gives you the opportunity to put characters into settings and situations they wouldn't normally be in, so your sandbox is a lot broader. This made it very appealing to base our game on this idea.

C. Art Inspiration

For the art style of the game, we wanted to aim for simplicity, not only because we felt that it would fit the casual, lighthearted nature of the game, but also because of the practical concern of only having one artist. With this goal in mind, we looked at a lot of older games from the 1990's, as many of these were forced to have simple art styles because of the technology of the time. We looked at several games that we thought achieved a really nice look despite these limitations, such as Nintendo 64 games like *Super Smash Bros.*, and thought about how we could emulate that.

As for the designs of the characters, inspiration was abundant. These characters were meant to represent heroic archetypes from broad literary Genres, so naturally we took cues from all sorts of games, books, and movies. For the comic book hero, we of course looked at characters from Marvel and DC comics, drawing a lot of inspiration from the likes of Superman (11) and Captain Marvel (12). For science fiction, we referenced games like *Halo* (13) and *Metroid* (14), as well as movies like *Star Wars* (15) to create our armored space bounty hunter. The main inspiration for the fantasy character was Robin Hood (16), because we liked the idea of a speedy roguish character rather than a knight (which may have been the more obvious choice).

Finally, the horror character takes inspiration from monster stories like *Frankenstein* (17), *Dr. Jekyll and Mr. Hyde* (18), and *The Incredible Hulk* (19).

2. Design Process

A. Tool Selection

I. Engine Selection

When starting our project one of the biggest decisions that we had to make was what engine we would use to create our game. The engine that we chose would influence nearly every aspect of the game including what we could and could not do, what the game would look like, and even how much time it would take to get tasks done. To make this decision we constantly took into account what our end goal for the project was, including what we wanted to personally gain from the experience, and looked at how individual engines would influence the path to those goals.

Our major goals for the project were to create a small, unique, casual game that provided some level of complexity in creation that would allow us to further develop our skills. To achieve these goals we needed an engine that would provide easy to use tools and be highly flexible to meet our needs.

Unity (20) met these needs perfectly. Unity focus on breadth rather than depth in its tools. Its editors are very simple and shallow. Learning these editors required no more than about an hour each meaning that the engine could be learned quickly and new skills with the engine could be picked up quickly and in parallel with development. The focus on breadth and the shallow tool set also meant that Unity was much more bare bones. Rather than Unity handling the core gameplay mechanics like walking, jumping, etc. it only handle base engine functionality like rendering, physics, and animation; the rest was up to the developer to create. This design principle of the engine meant that Unity was very open to extensibility making it very flexible. We could develop our own systems and tools where we needed to and even import plugins for

functionality that did not come with the base engine. Effectively we could mold the engine to meet our needs.

In the end the choice to use Unity greatly helped us. Over the course of the project we ended up developing many tools and resources for ourselves and gaining the experience to fully control our game. Had we picked another engine such as Unreal (21) such flexibility may have been difficulty or even impossible. The choice of Unity gave us the flexibility and creative control we needed to create such a unique experience.

II. Team Collaboration

Along with our engine we also had additional tools to help us collaborate and bring the project together. Through these tools we could share ideas, assets, design concepts, and more. These tools consisted of GitHub (22), Trello (23)/Hack N Plan (24), and Slack (25). We also played around with Unity Cloud Build (26) but were never able to get it integrated in a reliable way.

GitHub we used as our main way of bringing assets together. In addition to the Git repository that it supplied us with, GitHub also gave us a visual toolset for looking at content currently on the repository, seeing who changed what, and applying comments. In early development this was a very helpful set of tools as we could point to a specific line/section/system of code and comment about structural ideas. It also allowed us to easily share sections of the code between each other by allowing us to refer to commit numbers and URLs to such commits.

For our task system we ended up switching halfway through from Trello to Hack N Plan. Both use the same basic concept of having cards for tasks that can be assigned to users and moved around to different categories. We ended up switching to Hack N Plan (which at the time

was in beta) as it was specifically designed for game creation. We could categorize tasks based on if it was programming, art, design, etc. and for what phase of the project we wanted it done by. Hack N Plan also allowed for time estimation to be applied to each task and would calculate number of hours required to reach certain deadlines.

At the center of this we used Slack for team based communication. Slack allowed us to communicate effectively with each other and organize our chats into different channels. This allowed us to easily chat about multiple subjects with each other at once without people missing what was said or having messages get lost in the chat history. In addition to this chat functionality we could also integrate Slack with many of our other services. We were able to integrate Slack with both GitHub and Google Calendar (27) so we could get notifications of when commits were being made and when events were happening.

Using these tools together greatly helped our progress. It allowed us to be able to work more independently while still being able to bring assets together into one game. It allowed us to have constant communication as we worked on our own tasks and as we got more into the feel of how to use each one became a core part of our total tool set.

B. Workflow

Throughout every step of our game we had a clear and well defined workflow. We allowed to be subject to change as needed, but was always agreed to be followed and would not change without first consulting the team. This enabled us to very easily keep track of who was working on what, what there was left to do, and allowed us to change our way of working when we found problems in the workflow or as certain method became obsolete.

The early stages of development for us focused on building core systems and content that would be used throughout the rest of development. We needed a system that content was created

not only in an organized manner, but also in a elegant way that made sense to all people on the team. For these reasons during the early stages of development our workflow was very rigid. Tasks were first discussed in group meetings where we determined what steps forward had to be made for the next iteration, with each of these iterations being about a week long. These tasks were then assigned to a single person who seemed to fit best for that task. After the task was completed it would be put up for review before officially becoming part of the main game. This ensured that we knew who exactly was working on what and that we could shape the core of our game exactly how we wanted to.

As the core systems became complete and we continued onto the later stages of development, a more relaxed system needed to be put in place. During later development most tasks became independent pieces that did not need to be built off of. We also had many more tasks in late development as most tasks focused on individual game pieces. For this reason we changed our workflow to reflect the new needs. Rather than planning out every task we moved towards a task pool model. Tasks could be created and assigned by anyone who was free. When a task was finished we allowed it to be put immediately into the game rather than needing to go through review. As content no longer needed to be built off each other this workflow allowed for a lot more parallelization and didn't require team members to wait on other team members.

Overall our workflow allowed us to work efficiently. While there were still many bumps along the way as we learned what did and did not work, having a well defined workflow definitely did more good than it did bad and was one of the core reasons we were able to get through our project.

C. Core Mechanic - Pages

Early in the design process, we had proposed the idea of including items, gear and currency that would be separate from Pages. However, as we discussed the design, we realized that the Pages that build the rooms of the dungeon should be the core mechanic of our game, so we decided to cut all of these extras, and had Pages fill their role. Instead of having items that can be used in combat to heal health, specific Pages would do that when chosen as the move. To replace gear, which would give the player higher stats when equipped, we decided on having a deck-building aspect with the Pages, so the Deck would level up as the player went through the game. Finally, instead of having a currency for players to buy items with in the shop, we decided that players spending Pages for more powerful Pages seemed like good mechanic that put the focus on the Pages.

D. Deck-Building Combat

The deck-building style of combat came about as a result of our decision to create strategic combat while removing the stress that is common to real-time battles. Combat is entirely turn-based, allowing each player time to plan out their moves without feeling any pressure due to relentless enemy attacks. While many trading card games feature complex rules on each card, allowing for a wide variety of strategies, we chose to keep the rules of our cards relatively simple. The wide variety in the complexity of card game rules can be seen in Figure 3, which contrasts a card from *Yu-Gi-Oh!* (28) with one from *Magic: The Gathering* (29). Many early cards from the *Yu-Gi-Oh!* trading card game have fairly simple rules, featuring little more than a level, type, and attack and defense strength. In contrast, many cards from *Magic: The Gathering* feature more complex rules, allowing for complex strategies and synergies with other

cards. We chose to keep *Storybook's* rules simple, since this reduces the skill curve, making the game easier for new players to pick up.

Figure 3: Simple vs. complex cards. To make Storybook *easier to approach, we opted to have simple rules for combat, as seen in many early cards in the trading card game* Yu-Gi-Oh! *(left)* (30)*, as opposed to the more complex rules behind games such as* Magic: The Gathering *(right)* (31).

To keep Pages as the core mechanic, we chose to have Pages serve as the cards in combat. In combat, players only use Pages from an object called the Deck, which is a subset of all the Pages owned by that player. A Deck consists of a set of Pages separate from the player's inventory. Initially, the Deck is made of randomly chosen Pages, though over the course of the game players are able to swap newly acquired Pages into their Deck. There is also a limit to the number of Pages that may be placed into a Deck. This adds more depth to the deck-building aspect of the game, as the player must balance building a Deck to suit them well during combat,

while also including Pages that they wish to give up as "currency" in the shops as well as Pages with which they can build a new room.

In combat, a Page has several components that the player needs to be aware of at all times. The most prevalent is the Page's type, or Genre, indicated by the color of the Page. To help Genres feel unique, we implemented them in a manner similar to the type matchup systems found in many role-playing games, such as *Pokémon* (32) or *Fire Emblem* (33). Each Page falls under one of the four main Genres in *Storybook*: Comic/yellow, Science Fiction/blue, Fantasy/green, or Horror/red. Each Genre is unique in that it has advantages or disadvantages against the other Genres.

In addition to its type, each Page has a level. In combat, the Page's level determines its base power and its overall effectiveness. Page levels run from 1 to 7, with 7 being the strongest. Players start the game with nothing but level 1 Pages; these are meant to be a sort of starter Page. Over the course of the game, players will eventually fill their Deck with higher level Pages, allowing them to fight stronger enemies, win more powerful Pages, and trade up for rarer Pages at the Shop.

In combat, each Page serves one of two purposes: Attack or Boost. Attack Pages do just as they say, they deal damage based on the Page's level, as well as taking type advantage into effect. Boost Pages serve as support moves, raising a particular stat of an ally. The effect of a Boost Page is dependent on its Genre, and is clearly indicated on the Page by an icon. Fantasy Pages boost speed, Comic Pages boost offense, Science Fiction Pages boost defense, and Horror Pages restore health points. A full breakdown of a Page's structure can be seen in Figure 4.

(A) - **In Deck** - Icon indicates whether the Page is in the player's deck or if it is in their inventory

(B) - **Level** - Base strength of a Page

(C) - **Rare Icon** - A gold star means a Page is rare. Rare Pages target all players on the appropriate team instead of a single target.

(D) - **Attack/Boost Label** - Declares whether this Page is an attack (deals damage) or a boost (supports allies)

(E) - **Type Icon** - Attack Pages show their type here; boosts show the affected genre.

(F) - **Disadvantage/Advantage Icons** - Declares the genre this page is weak to, as well as the genre it is strong against.

Figure 4: Breakdown of a Page.

E. Major Goal - Simplicity

Throughout the design process of Storybook, our major goal was simplicity since it is intended to be a casual game. Initially, we considered having dungeons that the players would walk around in, solve puzzles and explore, in addition to the turn based enemy encounters. However, we felt that this would have been too detached from what we wanted to be the focus of our game; the turn based combat and Pages, so we decided to cut this altogether. Instead, we decided on having the players fight enemies in a room, and then simply choose which room they would like to visit next, where there could be another encounter or a special room, like a shop. We felt that this was better for our game because it made it simpler and easier to learn, as well as made the game more focused, since we cut everything that didn't revolve around the combat and the Pages.

However, although we were striving for simplicity, we also wanted to create a game and systems where an experienced player can also be challenged. We were inspired by the *Pokémon* series with trying to do this. In those games, a casual player with not much knowledge of the game can play through and reach the ending without much trouble. Experienced players, on the other hand, can use their knowledge in later game modes and even competitive games against other players, where they, too, can be challenged. The reason for this is that there are a lot of mechanics that are not necessary for progressing through the game, but are crucial when playing against another player of similar skill level. We planned to achieve this goal with type matchups, where certain Genres are effective against one other Genre, but are also weak to one. We also wanted to achieve this through the Deck management, since we planned on including many different types of Pages with varying effects, which meant that players had a variety of options and strategies when deciding which Pages to put in their Deck. Finally, to ensure that new players wouldn't be daunted by the game, we planned on having the more challenging modes separate from the main game.

F. Cooperative Gameplay

Playing with friends is a common feature among casual games, with some popular examples being *Clash of Clans* or *Words with Friends* (34). Most games that do have multiplayer have players going against each other in what usually is a competitive scenario. On the other hand, there are some games that have players working together for a common cause against AI opponents, such as *Minecraft* (35). This was something that we were interested in doing with our game since we thought that it would fit well with the gameplay that we had been designing. We believed that the deck-building combat would be perfectly suited for co-op gameplay since there would be collaboration between the players in terms of deck-building. For example, one player

may agree to serve a support role, taking a bunch of stat boost Pages in their Deck, while the other player may fill the role of an attacker, with a bunch of attack Pages in their Deck. We also wanted cooperative gameplay because from our experience it is a fun feature and results in engaging collaboration amongst the players.

G. Short Play Sessions

With our goal of a casual game in mind we wanted our game to be relatively easy to pick up and put down at any point. This was a significant challenge to try and merge with a multiplayer game. Unlike most other casual games we couldn't simply allow a player to quit at any point as we needed all players to be playing at one time. To overcome this issue we decided to focus on shorter play sessions. We wanted it to be possible for our game to be completed in a short amount of time and instead focus on multiple iterations. By doing this players would not have to devote a huge amount of time to a single play through.

3. Gameplay

A. Combat

In combat, the players face off against one or more enemy characters that they must defeat. Each of the characters in combat have hit points, and when these hit points reach zero, that character is defeated and removed from the combat. The players win if they manage to defeat all of the enemy characters, and lose if they are all defeated. Each character also has stats that affect the results of moves, such as the defense stat reduces the amount of damage that the character takes. The combat is turned based, and first waits for all characters, enemies and players, to select a move to use for that turn. A move can either boost a stat of the selected character for a couple of turns, or deal damage to the selected character.

Figure 5: Multiplayer Storybook Combat Screen

The enemies choose moves based on their AI, while the players use their Deck of Pages for these moves. At the start of combat, players are given five Pages randomly selected from their Deck seen at the bottom of Figure E, and draw a new Page at the beginning of each turn. Players can select any of these Pages in their hand for their move, and then must select the

targets. If the player selects a boost Page, then they must then select a character on their team to boost, and if it is an attack, they must select a character on the enemy team.

Once moves are selected, the combat enters the execution stage, where the selected moves are played out. The order that the moves are used is determined by the speed of each characters, with the ones with the higher speed stat going first. If a character is defeated before they are able to use their move, the move is not used and they can no longer select moves for the rest of the combat. Also, if a character is a target of a move and is defeated before the move is use, that move randomly selects a new target, making sure to only target enemies if it is an attack or only players if it is a boost. If all of the moves are used and both teams have at least one character remaining, the combat enters the move selection phase once again.

Figure 6: Combat move dealing damage

After each move is used, the game checks to see if either side is completely defeated. If all of the players are defeated, the game over screen is displayed and the players can return to the main menu to start a new game. If the enemy team is defeated, the players have won and are given the combat win screen, where they can select a new Page to add to their inventory. This is how the players can improve their Deck, since the Pages received from winning combat are typically more powerful than the Page that they used to make the room.

B. Character Selection

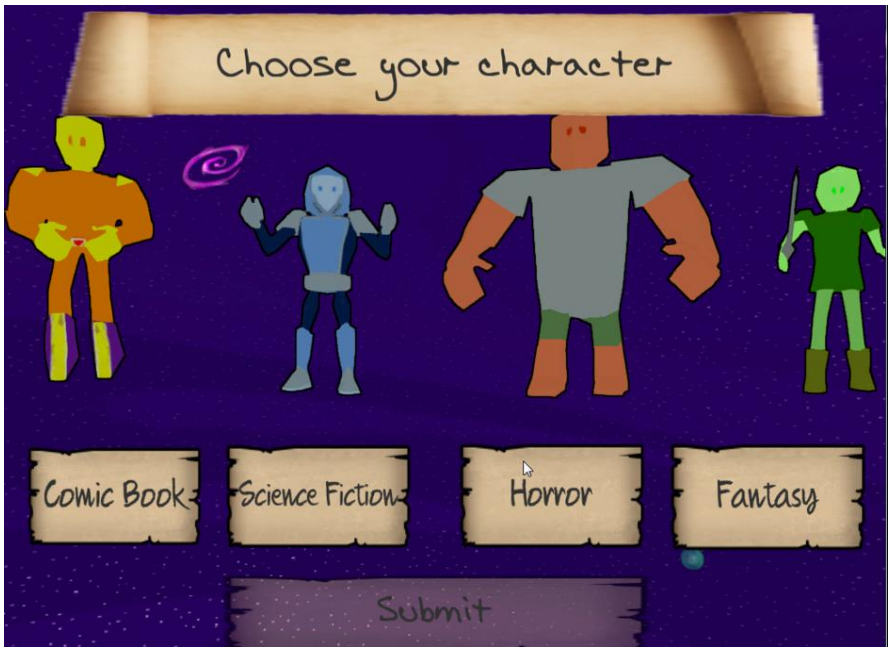


Figure 7: Character select screen

At the start of each game, players must select the character that they would wish to play. In a multiplayer game, each character can only be used by one player. A player selection is indicated by a greyed out button underneath the character, similar to how the Submit button looks in Figure 7. One of the differences between the characters is the stats. For example, since the Comic Book Genre represents attack, the comic book character has slightly higher attack than the other characters. In addition to this, each character has a Genre that they are strong against, and one that they are weak against. We will elaborate on this further in Section F: Genre Typings.

C. Room Types

When the map is generated, all of the rooms are assigned a type, each of which trigger a different event. Each room type also has a different appearance in game, with the shop room having a wooden floor, and the combat rooms having a floor based on the Genre.

I. Start Room

The start room is the room that the players are placed in at the beginning of each floor. In this room, the players can manage the Pages in their Deck, possibly placing in Pages that they received from combat wins in the previous floor. Once each player is satisfied with the contents of their Deck, the players can start to explore the floor. If the players return to the start room, the Deck management menu will appear again and they can once again select which Pages they want to have in their Deck.

II. Combat Room

When players enter a combat room, an enemy team is displayed in the center of the room, and the game transitions to combat. The Genre of the room and team is determined by the Genre of the Page used to build the room. For example, if the players use a Comic Book Page to build a room, the enemy team will be of Comic Book type and the room will be Comic Book themed. The floor of the combat room changes based on the Genre, with an example being the street that represents the Comic Book Genre. Finally, scenery objects are randomly placed within the room, that also correspond to the Genre of the room.

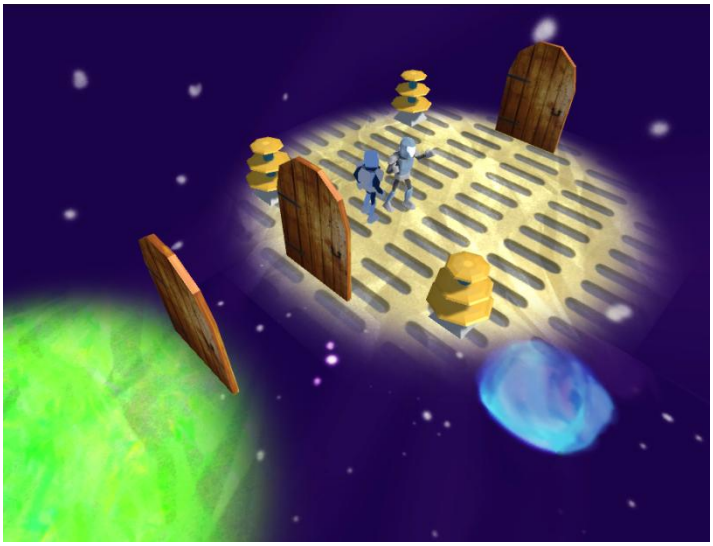


Figure 8: Entering a combat room

III. Shop Room

When players enter a shop room, Pages to trade for are generated and the shop menu is opened up. In the shop menu, players can trade their Pages for more powerful Pages, which are generated separately for each player. The levels and Genres of the generated shop Pages are based on the Genre and level of the Page used to build the room. Typically, the level is 2-3 levels higher than the Page used, and the Genre tends to match the Genre of the Page used. To trade for Pages, players must give up a maximum of three Pages whose levels must add up to or exceed the level of the Page that they are trying to trade for. Once a Page is traded for, the Pages that the player used are dropped from their inventory, and the new shop Page is added. The Pages that are generated by the shop are saved, so if one Pages is left to trade for and the player wants to get it later, by re-entering the shop that exact Page will be up for trade again. Alternatively, if a player has gotten all of the shop Pages, the shop menu will not open when they re-enter the room since there are no Pages left.

Figure 9: Shop room interface

IV. Exit Room

The exit room is the room that the players are trying to find on each floor, since it brings them to the next floor or the win screen if it is the final floor of the game. However, before being able to go the next floor, the player's must first fight a special boss character. The boss characters are more powerful than regular enemies, but the game transitions to a combat just like in the combat rooms. The combat runs as before, and once the boss is defeated, the players will have the option to move to the next floor. However, they can choose to continue exploring the floor, if, for example, they still want to find the shop of that floor. When the return to the exit room, there will be no combat since the boss is already defeated, and they will have the option to move to the next floor. If they choose to move to the next floor, the game saves their current inventory and HP, and loads a new floor of the dungeon.

D. Deck Management

As mentioned in section 2-D, deck-building is a major component of the gameplay. The Deck of Pages is used for moves in combat, and always has a set amount of fifteen Pages. At the start of the game, the player will receive fifteen Pages for their Deck, as well as additional Pages for them to possibly swap into their Deck. Also, throughout the game, players will receive new Pages from either winning combat encounters or trading for Pages in the shop. At the beginning of each floor, players will be able to manage their Deck, that is choose which Pages they want to put into their Deck and which Pages they want to keep on the side.

Figure 10: Managing the Deck user interface

E. Dungeon Traversal

As players go through the game, they will be choosing doors to open in the dungeon in their search of the exit room on each floor. After a room event occurs (combat, shop menu, etc.), the players must choose a direction to move in based on the doors available in the current room. In a multiplayer game, a leader is chosen for each room, and they are the one that have the say in which direction the group moves in.

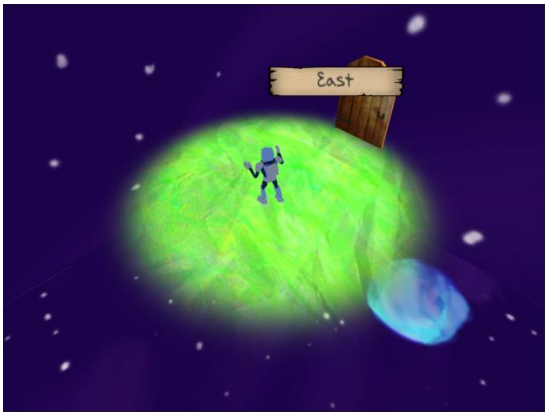


Figure 11: Choosing a direction

Once the player chooses a direction, they must choose a Page to place down to build the room. The Page chosen has various effects on the event of the room that is created. For example, if it is a combat room, then the team chosen will match the Genre of the selected Page, meaning it will mostly contain enemies matching that Genre. If it is a shop, then the Pages generated to trade for will more likely match the Genre of the chosen Page than being any of the other Genres. The level also has an effect, such as making enemies in combat rooms more powerful will better rewards, and increasing the average level of the Pages generated to trade for in the shop. When a player uses a Page for building a room, it is dropped from their inventory and is replaced with a basic level one Page.

Figure 12: Selecting a Page for a new room

Once the players reach the exit room and clear the boss, they will have the option of moving onto the next floor. The game does not automatically move them to the next floor since the players may want to continue exploring the current floor, with one possible reason being to

find the shop for the floor. Once players choose to move to a new floor, the game starts a new level with more powerful enemies or ends the game if they were at the final floor of the game.

F. Genre Typings

Many role-playing games feature some sort of type matchup system to allow different classes to feel unique and to add more depth to the gameplay. Perhaps the most well-known example of this is the *Pokémon* series, which features a rock-paper-scissors style of type advantages and disadvantages across each of its 18 types of characters. To give a sense of identity to each of the characters in *Storybook*, we implemented a simple type advantage system to give each Genre (the "classes" of *Storybook*) its own strengths and weaknesses against other Genres. In short: Fantasy beats Science Fiction, Science Fiction beats Comic, Comic beats Horror, and Horror beats Fantasy. Figure 13 shows how the types match up against each other.

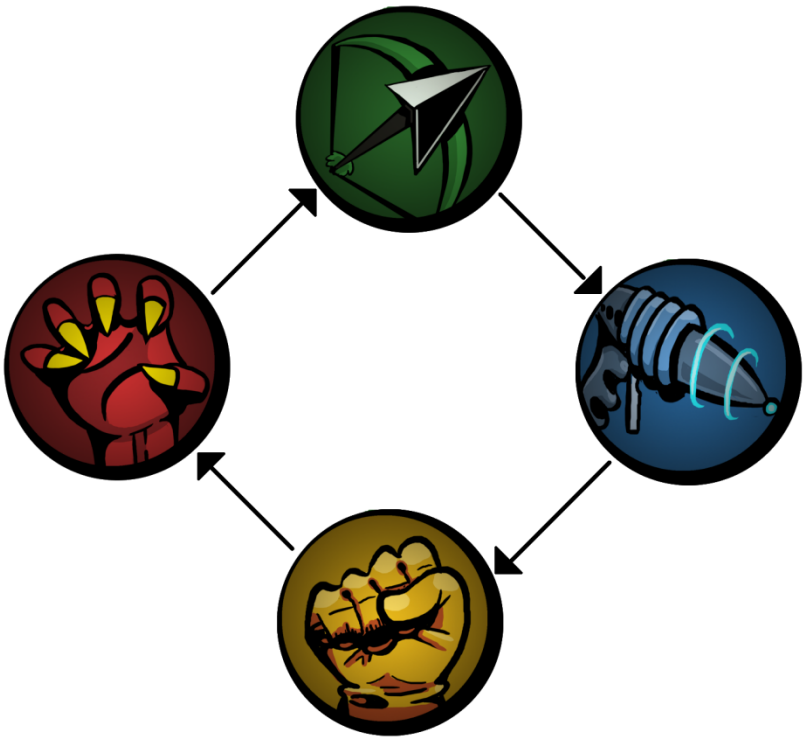


Figure 13: The type advantage system in Storybook. *Arrows point in the direction of a positive advantage. For example, a Fantasy attack is more effective on Science Fiction characters than on a Horror or Comic character.*

Because *Storybook* features relatively few classes compared to other role-playing games, we decided it was best to keep the type matchup system simple. This also makes the type advantages less daunting to learn for a new player, as they only have to remember a small handful of interactions, as opposed to the hundreds of potential type interactions in a game such as *Pokémon*. Each Genre has an advantage against a single other Genre; a disadvantage against the Genre it is weak to; and no advantage or disadvantage against the remaining two Genres. Additionally, when a character uses a Page that is of their own Genre, they receive a small boost to their attack power. However, when using Boost Pages, the type advantage system is even simpler; the effects of Boost Pages are amplified when a Boost Page is used on a character of the same Genre, otherwise there is no change in their effectiveness.

In keeping with the general theme of simplicity, the damage calculation formula in *Storybook* is rather simple as well. Though it takes in numerous factors including the attacker's strength; the defender's defense; the types of the attacker, defender, and the Page being used; and the Page's level, the damage calculation formula does not feature any complex calculations. We wanted a simple, concise formula so that it is both easier for us to balance *Storybook's* combat, as well as so players can easily grasp how effective their attacks will be. Figure 14 displays the full damage calculation formula.

$$Damage = (PageLevel + AttackerStrength + 1) * (SameTypeBonus * TypeAdvantageBonus) - DefenderDefense$$

Figure 14: Damage calculation in Storybook. *Though taking in a number of factors, the formula is relatively simple, making it easy to balance as well as giving players a relatively easy way to gauge their strength.*

G. Tutorial

To teach the players the game, we decided to implement a tutorial game mode that would cover all of the basics. In a complete run of the tutorial, players will select a character, manage their Deck, enter a combat room, enter a shop, and enter an exit room. At the beginning of each new event, a window will appear with some text explaining what the event is and how it works. For example, when the first enter a combat, the game explains that they need to use Pages from their hand as combat moves. We decided to implement the tutorial in this way because it allows the players to learn the game one bit as a time, instead of being overwhelmed from the game explaining everything to them at once. Also, it allows the player to play the game in between each tutorial message, so the player does not become bored from not being able to play for extended period of time.

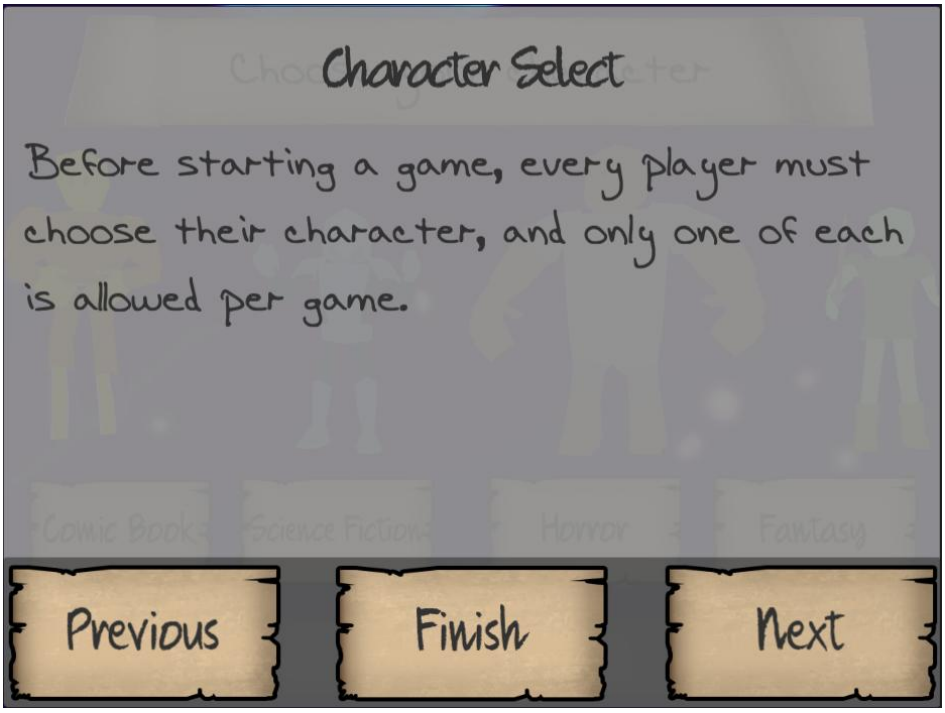


Figure 15: Example of a tutorial prompt

4. Art

A. Design Goals

In developing the game's visual style, we decided early on to strive for simplicity. We felt that a less realistic, more cartoon-ish style would fit the light, casual nature of the game, and that because we only had one artist on the team that it would be better to avoid more labor-intensive styles. We started thinking about how we could visualize our game using mainly simple shapes and colors. We looked to old Nintendo 64 games for inspiration, as that consoles limitations restricted all of its art assets to few polygons and simple textures. *Kirby 64: The Crystal Shards* (36) and the original *Super Smash Bros.* were two of the games we looked at most.

Figure 16: Using art style to overcome technical limits. Despite technical limitations, these games both achieve nice looking visuals with a lot of character.

The first major decision we had to make was whether to use 3D or 2D art for the game. We decided that either could work for our target art style, but ultimately went with 3D. The decision mainly came down to the fact that at the time, our artist had more experience making game assets in 3D and felt that he would be more comfortable with that workflow.

B. Initial Concepts

One of the first concepts we came up with was the idea of the combat sequences taking place on a giant book. At some point we did consider creating distinct battlefields for each Genre, but we wanted to keep the scope limited. We liked the book idea because it fit well with the game's themes, and the idea of fighting on top of a book seemed pretty unusual. With that in mind, we created some quick concept art.

Figure 17: First concept art

Even though nothing about this picture really made it into the final game except the book, it still demonstrates the main idea of the game: different characters fighting for control of the story. Everything after sort of grew from this basic concept.

As mentioned in earlier sections, the game's four player characters were meant to be archetypes of the four literary genres, fantasy, sci-fi, horror, and comic book. When designing these characters, I started by simply sketching out characters that seemed lifted from each genre,

a roguish “Robin Hood” type character for fantasy, a space bounty hunter for sci-fi, a Mr. Hyde-esque monster for horror, and a superhero for comic book.

Figure 18: Early character sketches

As we continued to develop the game and began to make final decisions about mechanics, we decided to go in a slightly different direction with the character designs. The game’s strategy mostly revolves around knowing how effective each Genre type is against each other type, and to make it easier for the player to keep up with, each type is coded to a color (Horror = red, Fantasy = green, etc.). We thought that the character designs should also serve to help the player keep track of the type system using the color code. In other words, the Horror character should very clearly be “the red guy” and so forth. To accomplish this, we made new designs that were less human and more abstract. Our thinking was that these new characters were not so much people plucked from different worlds, but rather representations of each genre’s “essence”. We thought of them as sort of a pantheon of “Storybook Gods”.

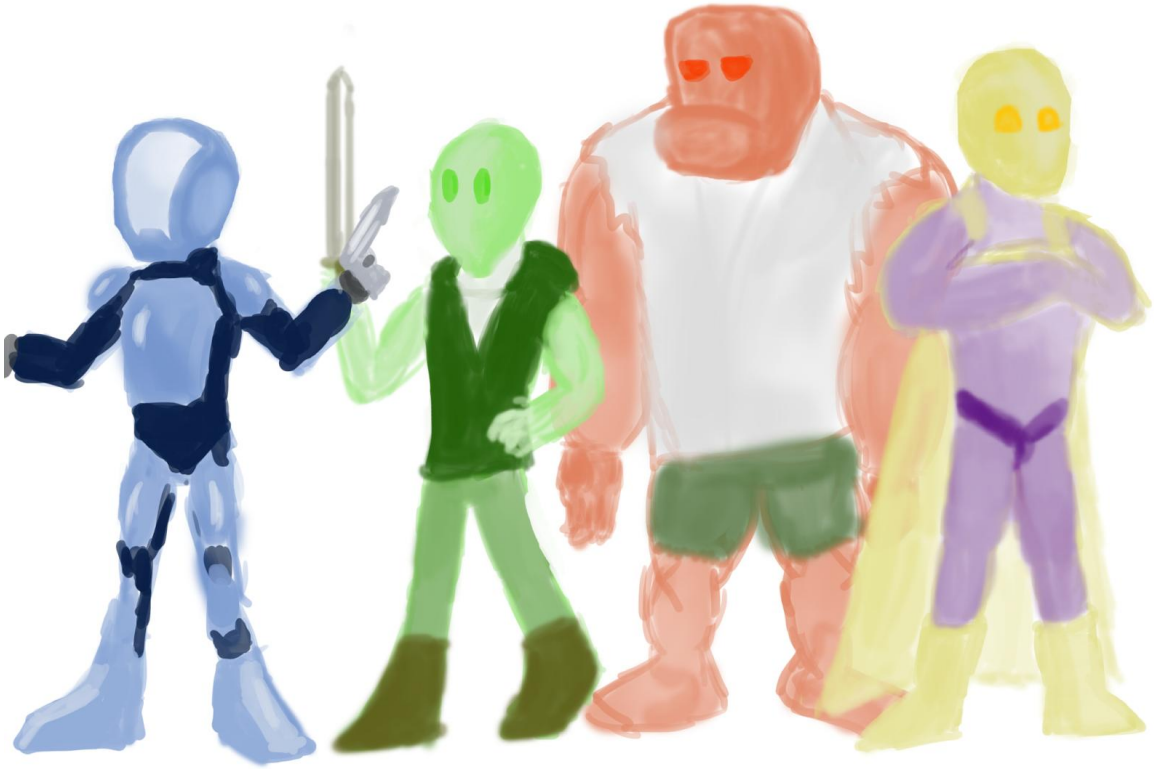


Figure 19: Character redesigns. These designs are very close to what ended up in the final game.

C. Implementation

Once we had settled on a general art direction and nailed down the character designs, the actual production pipeline was fairly straightforward. We decided it would be best to use Autodesk Maya (37) for all of the modeling and animation, as it best supports a low-poly style and workflow. After that, it was just a matter of getting each asset modeled, rigged, textured, and animated.

As mentioned earlier, one of the bigger limiting factors was that we only had one artist working on our team, so prioritization of art assets was key. It was decided early on that the four player characters should be given the most attention, as they are the most essential part of gameplay, and the only assets are seen by the player throughout play. These characters were

modeled, rigged, and animated in Maya and used textures hand-painted in Photoshop (38). The environment assets consist of Photoshop hand-painted floor textures and simple "clutter" objects made in Maya.

One issue we faced while creating the environment was what to do about the skybox. We wanted to give the sense that the "rooms" in the game were drifting in a sort of cosmic nether-space between worlds, so we thought it would be cool if the skybox was animated to reflect that. This was proving difficult to achieve with a traditional 6-sided cube mapped skybox, because Unity required each side to be an individual texture. Instead, we just put a large inverted sphere over the whole environment and textured the inside. We made the texture with several layers and transparency maps and animated them individually to achieve the hazy, floaty look that we wanted.

Along with the characters, the other essential art assets were the Pages used in the game. As described in earlier sections, the Pages are the core of the gameplay, used by players to create Decks for combat and to generate the dungeons. Because they are so central to the game, it was very important that the Pages looked nice and were very readable to the player. We were able to get a second artist, Dillon DeSimone, to help out for a few weeks of the project, and in that time he did a lot of graphic design work that greatly contributed to the look and layout of the Pages.

Figure 20: Genre type icons made by our "guest artist"

Because the Pages need to quickly convey several pieces of information to the player, their design relies heavily on icons and symbols like these. The Pages were revised multiple times during development in order to make them as useful as possible to the player.

Figure 21: Earlier design (left) versus revised (right), putting less emphasis on text and more on the icons

Once the assets were all in place, the last phase was tweaking settings in the Unity engine to achieve the look we wanted. With low-poly assets like ours, choosing the right shader and import settings can go a long way in nailing down our visuals. Softening the normals on the models and applying a "toon" shader (which adds black outlines to each objects among other

visual tweaks) were two of the most effective techniques we used, as they deemphasized the individual polygons on the assets and made them appear more as unified simple shapes.

Figure 22: Final art results with all "visual tweaks" in place

5. Sound

A. Music

We wanted our game to have some original music to stand out, so we decided to contact Tangent Music LLC (39), and they agreed to make a couple original tracks for *Storybook*. The three tracks that we requested were a menu theme for starting up the game and joining/creating a game, a dungeon theme, and a combat theme. To get started, we had a discussion with the members of Tangent Music LLC to describe the gameplay mechanics and art direction of our game. From there, we sent them some music sample from games that we enjoyed and believed fit how we wanted to the tracks in *Storybook* to sound. During the development of our game, we stayed in contact with Tangent Music LLC, giving them updates on our game, as well as receiving the status and latest versions of the tracks. Once we received a track for the game, we imported it into Unity and had it play at the appropriate time during the game.

B. Sound Effects

We did not have much to work with in terms of sound effects, so we decided that we would use royalty-free sounds that we would find on the internet or self-recordings for some of the simpler sounds. Because of this, we also decided that we would only use sound effects at key moments throughout the game. One of the major uses of sound effects in our game is when a selection is made in a UI menu, since giving feedback to the player is important with UIs. Also, we used sound effects in combat for attacks and boosts, since we believed that it made the combat more satisfying since each move has an auditory impact.

6. Technical Implementation

A. Photon

I. Why Photon?

From very early on in the games design we knew we would want to provide some form of networked multiplayer. To fit our design and constraints we needed a networking solution that would be both easy to use and quick to setup while still allowing for flexibility in our code. We also needed a solution that would allow for matchmaking so users did not have to remember their friends IP address. Finding such a solution proved difficulty and we ended up going through a number of different networking solutions each with their own ups and downs

We finally decided on a third party tool called Photon (40). Photon provided us with most of the features we desired. The company itself provided matchmaking and relay servers for use by developers eliminating our need to figure out how players would connect to each other. Although these servers cost money a free plan was provided to allow for easy testing.

Photon also came with an open-source Unity API. This API integrated communication with photon servers into the engine itself. Objects could be marked to receive network information including when players connected and disconnected and when new rooms were created. The only thing the API did not provide was a flexible way of sending data. The best the API could do was provide a way for game objects to write data into a stream that would be sent to other players, as well as triggering remote code on other players. Due to its open-source nature however we found we were able to write our own object networking solutions to provide us with the flexibility that we needed.

II. How Base Photon Works

Photon can be divided up into two sections. There is the architecture section, which deals with how data is physically transported from one player to another, and there is the game world section which deals with how players' worlds are represented through network data.

As mentioned previously, Photon provided servers for developers to use. These servers had two responsibilities, match-making and data relay. The match-making was relatively simple to understand. Players would connect to the match-making server where they could then ping the server about room information (Photons name for matches). Players could then either connect to an existing room or create their own for other players to connect to. Once connected to a room Photon would use its relay servers to allow communication between clients. These relay server only purpose was to relay data between clients, not be authoritative server on what's happening in the game. To still allow for an authoritative server design Photon would mark one client as a "master client" which was designated to have authority over the game.

Photon provided a Unity library which was composed of both a low level and high level API. The low level API was focused mostly on allowing communication between clients through photon. It provided serialization, packet delivery, and room management methods that could be used so that raw socket communication was not needed. The high level API provided the actual integration with Unity. The main construct of the high level API was the PhotonView. This was a component that could be added onto objects in order to make them networkable. This network view defined which client owned the object and provided functionality for serializing other components on the same object.

III. Photon Modification

Early on in the development of our game we decided that the game should be as close to full server authoritative as possible. This would mean that the server (or master client) would be in charge of everything and clients would simply replicate it on their end. Our game being entirely turn based also meant that there was no need for any client side prediction, so our network code simply had to send inputs from players to the server, and then have the server send back the world state and events. This flow of data was desirable for us as it meant our net code would be fairly easy to follow and create. At no time would we have to worry about resolving client conflicts.

To help make development easier later on we decided to dedicate time to modifying Photons high level API to be more authoritative server friendly, and to allow cleaner and more flexible net code. The first major modification we did was the addition of sync properties. This allowed us to write object serialization code not as a block of code serializing to a stream, but rather as attributes on object properties. This gave us an easy view of how an object's state would be serialized and allowed for quick changes in an object's state serialization.

When state serialization was not a viable solution however we relied on Photons RPC system (remote procedure call) which allowed us to remotely call a piece of code on another client. As we were using a central server method we did not want to allow these to be called from any client to help ensure that the game remained secure. Thus we limited the calling pipe to now be only from server to client. When clients needed to talk back to a server about input we allowed the server to mark special objects as being controlled by a client and through these objects clients could send messages back to the server.

The open nature of Unity and Photon allowed us to have a great amount of control over our networked environment. We ended up having the power to not only include these changes but several others that helped us accomplish things not possible with the out of the box solution. This flexibility allowed us the power to write complex net code quickly and spend more time on building the actual game than just writing net code.

B. Networked Combat

The major component of our game is the combat, which needed to be synced over the network, so a lot of thought went into designing this core system. There were many iterations during the implementation of this system, and every step was carefully thought out and tested once completed to ensure that everything was working.

I. Initial Implementation

To start with this system, the primary goal was to get a turn-based combat working locally, with networking being pushed back to a later iteration. The end result was a basic implementation where the user would press space bar and the player pawn would attack the enemy pawn, and then the enemy pawn would attack the player pawn. The key components of this implementation were the combat state machine, the Combat Manager, and the Combat Pawn classes.

The combat state machine handles the transitions of the combat between its four possible states; waiting for input, executing the moves, players win and players lose. The game waits in the input state until the player hit the spacebar, and then executes the moves displaying simple animations. After each move is completed, the game checks to see if either side was defeated,

and transitions to the corresponding win or lose state if necessary. If all the moves were executed and neither side was fully defeated, it returns to the waiting for input state.

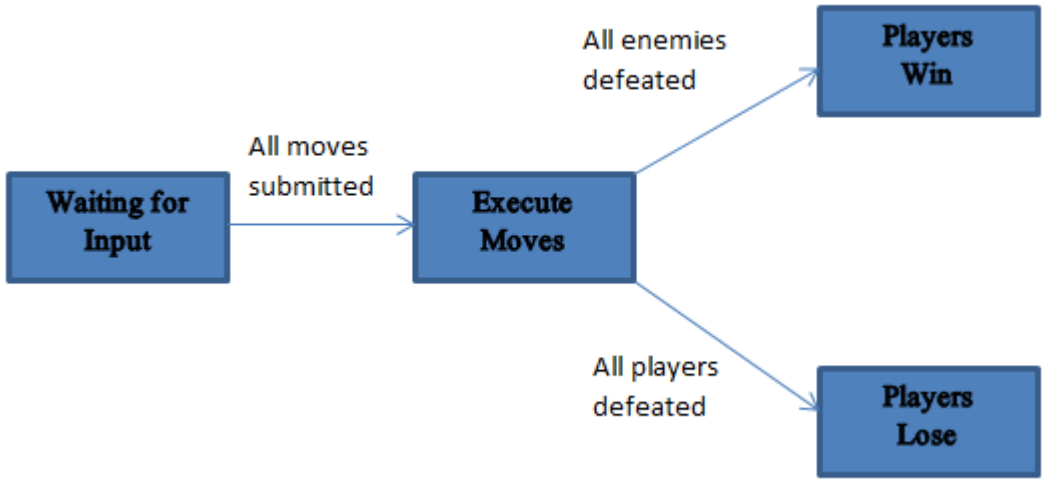


Figure 23: Combat state machine with transitions

The Combat Manager is in charge of various aspects of the combat, primarily starting the combat by transitioning from the overworld. The Combat Manager spawns the necessary pawns for the combat, as well as sets up the state machine and sets in in the waiting for input state. The final role of the Combat Manager in this version is keeping track of the moves that each pawn has selected, although in this version there was only one move that could be used.

The final component of the combat is the Combat Pawns, which store the stats of the characters, as well as handle move selection. When the player hits spacebar, the Combat Pawn sends the selected move to the Combat Manager. By the end of this iteration, we had a working a thoroughly tested version of combat for a single player, and the next step was to get it working with Photon networked multiplayer.

II. Networked Implementation

Designing how the combat would function over the network was a challenge because we needed to make sure that the implementation would never result in players being de-

synchronized. The first aspect that we considered was figuring out what we needed to send over the network to the other players. What we learned is not that much needed to be sent, basically just the move that each player selected, as well as the targets of these moves. As a result, we decided that these values would be the only things sent over the network during combat, so everything else, such as executing the moves, would be done locally so as to avoid potential points for a de-synchronization to occur between players. To handle this, there were two key additions made, being the Combat Manager sending out player move selections to all players, as well as a networked state machine.

The Combat Manager object was instantiated on all clients, meaning that RPCs could be sent out to all of the players. Whenever a move was received by the Combat Manager, it sent out the move data to all of the players via an RPC call. Because enemies choose moves based on their AI, it would be possible that if the AI was left to select a move on all clients that the selected moves would be different. As a result, we had only the enemies on the master client select a move, and then the Combat Manager sent out this selection to the other players.

The networked state machine was also something that was managed by the master client. The state machine that was in the initial version was made to only run on the master client. Because of this, we created networked objects for each of the states that would be created on all players when the master client entered a new state. So once all the moves received and the master client state machine transitioned to the execute moves state, the master client would create an Network Execute state on all clients which would run all of the selected moves on each client separately. Once all players executed the moves locally, the master client would be notified and would transition back to the waiting for selection state.

Implementing the combat in this manner proved to be effective, as there were only a few values/objects being synchronized, and thus, only a few points where the game could de-sync for the players. This made it easy for debugging when something went wrong since there were only a few spots in the code where the problem could be occurring. The next big step for us from here was allowing the user to select a particular move and its targets, as well as tying in the Pages and the Page moves.

III. User Input & Tying to Other Systems

Our first step in handling user input was designing the UI for combat, and implementing this design into the game. Once we planned out the UI, we built it in Unity using the UI building tools, and then went to work on giving this UI functionality in combat.

One of the primary additions that we made in this stage was the Combat Deck, which was associated with each player and the source of all the Pages that the player would be able to choose from for their move. The way the Deck worked was at the beginning of every combat, it would grab all of the Pages from the player's inventory that were marked as being a part of the Deck and shuffling it. At the start of the combat, five Pages would be drawn from the Deck and placed in the combat UI for them to select, with a new Page being drawn each turn after that. Once a Page was selected, the player would then choose the targets for the Page. After submitting the Page, the move was sent over the network via the method described in the previous section.

Another component of our game that needed to be integrated into the combat was the Player Entity, so that the damage that players took during each combat could be carried over to the next enemy encounters. To do this, the stats from the Player Entity object were loaded into the Combat Pawns at the beginning of each combat. At the end, the values would be updated in

the Player Entity based on the ending values in the Combat Pawn. At this point, we had a functioning combat system that could be played with multiple players over the network, and was fully integrated the necessary features of our game.

Breaking up the combat system into these steps during its creation proved effective, because the end result is a stable turn-based system that stays synchronized over the network. With many different actions and combinations that can occur during combat, having a stable system was necessary for the creation of our game. It also allowed us to add new content to combat, such as new moves or enemies, and be confident that the underlying system was not the problem if a bug arose.

C. Enemy AI

I. Properties

When designing the enemy AI, the major goal was to make it easy to create many different enemies that have variety of different behaviors. To do this, we separated out the major attributes of enemy AI, being stats, move selection, and target selection, so that it would be easy to mix and match these attributes to create different enemies. Separating the attributes made it possible to create new enemies with different behaviors and models on the fly without having to create new code, which proved helpful during the balancing stage of development.

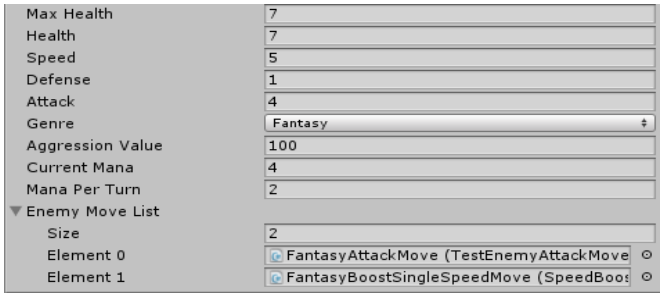


Figure 24: Enemy editor values

II. Move Selection

Every enemy has a variety of stats that can be set in the editor, including general combat stats, such as hit points and attack power, as well as some AI values. These AI values include an aggression value, starting mana value, and mana per turn. Each enemy also has a list of combat moves that it can choose from. To give weights to the moves that an enemy has, we decided to assign mana values to each move. This means that an enemy can only selects moves that it has enough mana to use. All enemies have a starting mana value and an amount of mana that they gain per turn, all of which can be set in the Unity editor. The other AI value is the aggression value, which is between 0 and 1, and affects whether the enemy prioritizes support moves that boost stats of its teammates or attack moves. The higher the aggression value, the more likely the enemy is to choose an attack move, and the lower it is, the more likely a support move will be chosen.

III. Target Selection

Once an enemy has chosen a move, it needs to choose the targets of that move. If the move is an attack move, the enemy will only have the opposing team characters to choose from, and support move only allows for teammates to be targets. Since target selection is specific to the move being used, the target selection algorithm is placed in the move object instead of the enemy. For example, when an enemy uses one of its attack moves, a selection might be choosing the pawn with the lowest health. The way the moves are implemented make it easy to make multiple moves that have the same effect, but a different target selection method, which allows for a variety of different behaviors. With this system, we can also make moves that are specific to certain enemies, like boss characters, to make that particular enemy seem unique since they would be the only one in the game to have that move.

The end result is a system that makes it easy to create new enemies and to balance the ones that are already in the game. One downside is that this system required a lot of different move object classes to be created, each with their own target selection process and effect, but we believe that it was worth the time because it has allowed for varied enemy behaviors.

D. Map Generation

One of the most common aspects of a rogue-like game is random map generation, so it was important that we implemented this feature in a fun and effective way in our game. The major goal of the map generation was to design a system that created varied map layouts with a lot of designer input to influence the process. To accomplish this, many of the variables that are used in the generation process are exposed to the developer. The most basic control over the map generation that the designer has is that they can set the length and width of the map. Another variable that can be set is the minimum distance from start to exit, which is the minimum number of rooms that the players would have to travel through to get from the start to the exit. Finally, since the maze generation algorithm used created linear paths, the designer can set a number of extra connections that they would like to see added to the maze.

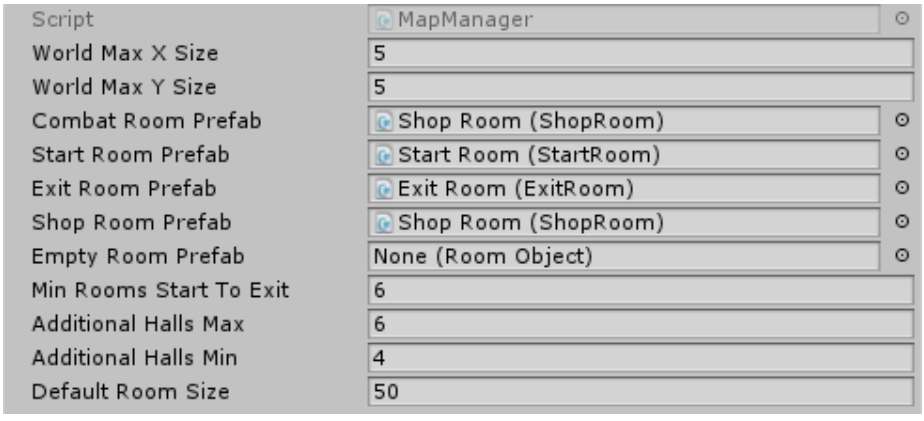


Figure 25: Map Manager editor values

To generate the door placement for each room, a depth-first search maze generation algorithm was used. First, the start and exit positions are randomly placed on the map. From there, the algorithm starts at the start position and checks the surrounding position in a random order. Once it has a random surrounding position, it will connect it to the room it is currently looking at as long as it is a valid position on the map, the room has not already been visited/connected, and as long as making the connection does not violate the minimum distance from start to exit set by the designer. Because it never makes connections with rooms that have already been visited, the algorithm tends to make a linear maze. After the maze is generated, it creates the additional hallways/connections based on the number set by the designer as mentioned earlier. To add additional halls, it first randomly selects two adjacent rooms that are not currently connected. From there, it checks to make sure that adding this connection would not violate the minimum start to exit distance. As long as it passes this check, it will connect the rooms, and keep doing this until it adds the number of extra connections set by the designer. Finally, the map generation algorithm places the shop room in the map by grabbing a random position that is neither the start nor exit.

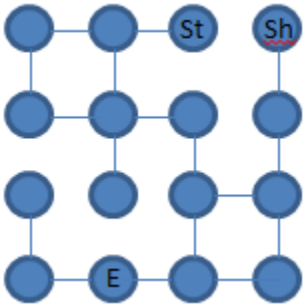


Figure 26: Sample 4x4 map generated in the game. St indicates start room, Sh is the shop, and E is the exit for the floor.

The result of this map system is that the levels in our game feel natural to progress through and as though they were hand designed. Also, with the amount of designer input that we

allowed for, it makes it so that we can change the maps based on the level that player is on, such as larger maps for the later levels.

E. Event Dispatcher

One problem we found as we got more into the core game programming was how to allow separate systems to allow objects to communicate with each other in an easy to use and flexible way. Originally we just used plain method calls from one object to another. However this quickly created a huge number of dependencies in our game and made changing code to difficult.

To fix this issue we created the event dispatcher system. The event dispatcher system was composed of three main parts, the caller, the dispatcher, and the listener. The caller was simply the object that wanted to inform the world about something happening. It would call a method on the specific event dispatcher it wanted to inform. As an example you might see some code that is demonstrated in Figure 27.

```
roomEventDispatcher.OnRoomEntered();
```

Figure 27: Using the EventDispatcher in code

This code would inform the dispatcher that a room had been entered. The dispatcher would then forward this to all listeners who had registered for this dispatcher. Listeners we implemented as interfaces which allowed us to make any type of object a listener for any dispatcher very easily. The total system ended up looking like Figure 28.

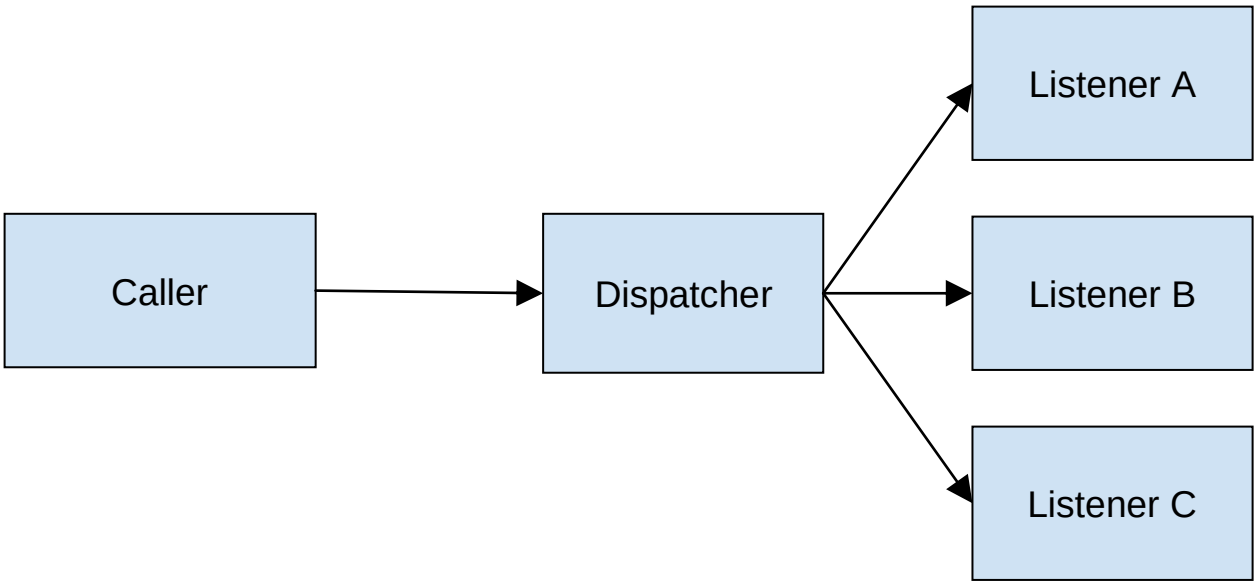


Figure 28: Event dispatcher system

Although at first glance this system may seem like an oddly complicated way of making calls between objects it was a helpful strategy for us for creating communications between two objects in entirely unrelated systems. This allowed us to make our game code somewhat flexible to change and gave us the option to move pieces around.

F. Dungeon Master

Random generation of certain aspects of the game is a major feature of rogue-like games such as this one, so we created the Dungeon Master. We wanted to centralize most of the random generation, so most of it is in this object, particularly dealing with Page generation and drops. To make it easier for balancing, we provided a variety of values that can be tweaked that would change the probabilities of certain events happening, such as the probability of obtaining a higher level Page after a combat. The Dungeon Master also handles randomly generating the shop Pages, which can also be tweaked by altering values that are exposed to the designers. The Dungeon Master object proved helpful while balancing our game because it made it easy to find

Page generation since it was all centralized this object, and made balancing a simpler task with all of the values that were exposed to affect the Page drops.

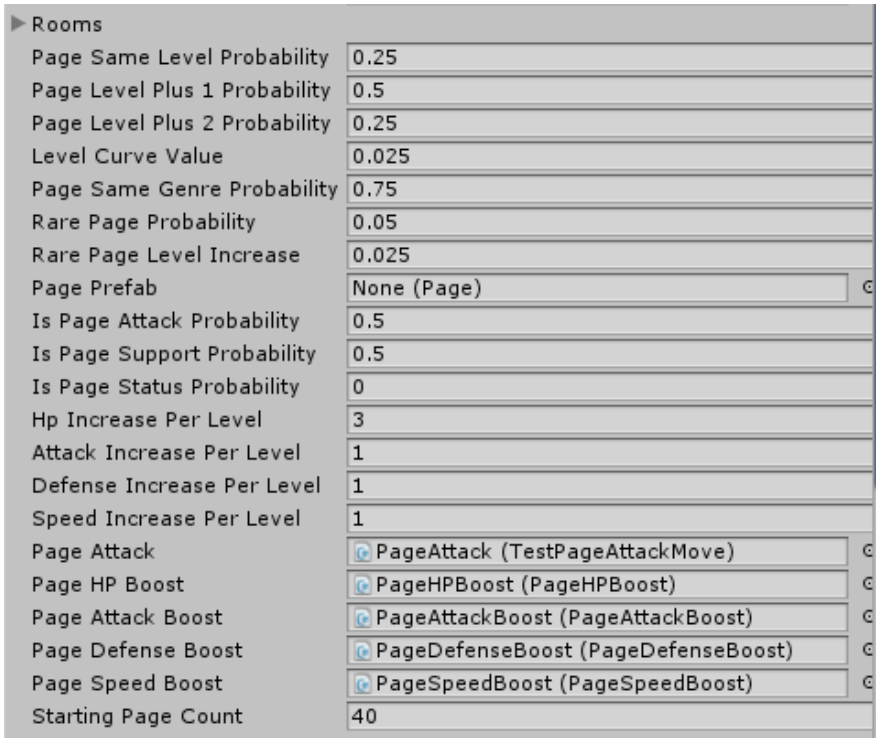


Figure 29: Dungeon master editor values

G. Inventory

One of the core features of our game was the concept of being able to hold multiple cards in an inventory. We wanted players to be able to pick up new cards, move cards around, and drop cards when the user no longer needed them. The concept of this itself wasn't too difficult; we essentially just needed a small data store that contained what cards the player had to use. The major challenge came from trying to get this to work on an authoritative server. Because the server was the only entity allowed to change things, a player could not locally make changes to their inventory, but had to inform the server of what they wanted changed. Unlike simple input, this could cause conflicts with both the server and client trying to make modifications at once. The players' state could easily become out of date causing the two clients to de-sync.

For a solution we turned our attention towards source repository systems like Git and SVN. The main appeal of these repository systems was their ability to allow multiple clients to make changes in parallel and merge them together so long as nothing overlapped. By applying this to an inventory system players would not only be able to keep their version of the inventory in sync, but would also would allow the server and player to apply changes at the same time.

In our implementation we defined a timeline of "commits" to the inventory. Each commit represented one change to the inventory system. The timeline was split up into two sections, the locked timeline and the floating timeline:

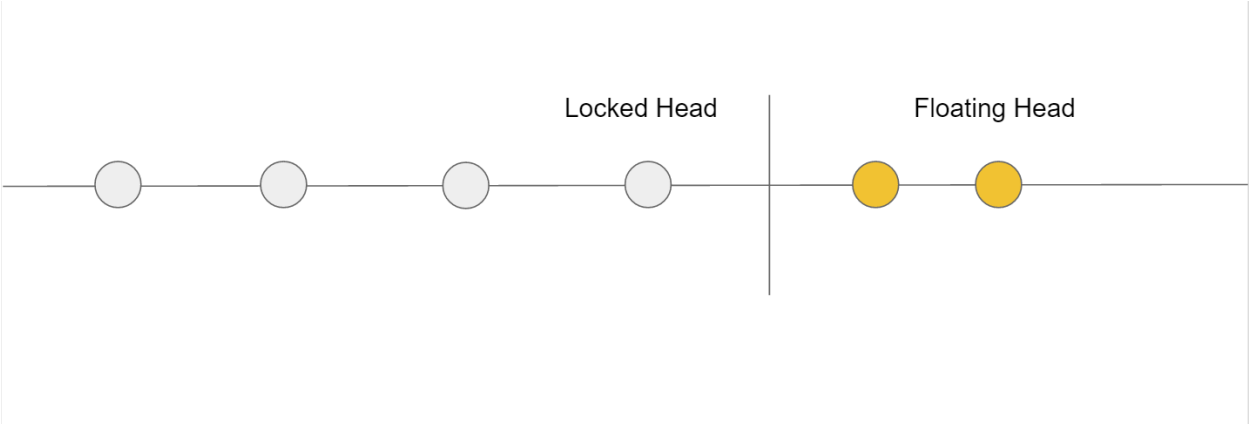


Figure 30: Inventory timeline

The locked timeline consisted of commits that the local client was certain about, and the floating timeline consisted of commits that had been applied to the inventory but had yet to be validated. When a commit is made on a client it is put on the front of the floating timeline and the changes are made locally. The commit is then sent to the authoritative client who will try to apply the commit onto their locked timeline. If the commit is successful the commit is sent to all players to be locked. If it fails, then the original sender is told to revert. This is done by having the client revert all commits on the floating head up to the invalid commit. The client will then

try to re-apply the other floating commits in front of it. This is used to check to make sure that changes to the inventory after the invalid move are also not invalid.

H. Player Entity

With our game being multiplayer it was very important for our game to have the ability to share player information between players. To give players one center location for their information we created a player entity system. The player entities were not the physical representation of the players in the world, but were rather an invisible construct used to represent the real world player in the game. These entities could store information about a player's name, what Genre they chose health, inventory, and more.

To allow the player entities to be flexible throughout the game we developed a game management system to handle the player entities. This system was built to allow player entities to be swapped out at any point during the game, thus allowing functionality of the player entities to change as our game went from one state to another. This system allowed us complete flexibility over the game logic will relative ease.

I. Map Movement

While our concept of map movement was simple, making it work reliably over the network proved to be more complicated. We needed to create a system that would run a state machine, allow rooms and other events to execute code, allow player objects to move with each other, and keep all players in sync across the network. We also wanted this system to be flexible and very open to change to allow for features of the game to be changed easily.

We ended up creating three major pieces for this code. The first was the room movement code itself. To make moving from room to room easy we define nodes within each room to

define where the doors and center of the room are. We then created a very simple mover that could be instructed to move to a node on the master client. The master client would relay this node to all players and all players would simply move towards the node at a constant speed. To make sure everything remained in sync with each other we also had the master client handle all events for reaching the nodes, and had the event messages relayed to the other clients.

We then had to solve the challenge of keeping players moving with each other. Originally we defined each player's pawn (the player model they controlled) as a mover on the network and had each player move to a specific node. However we quickly found that this code was very hard to manage and was to prone to visual errors. It was much simpler for us to define a single mover, and create the pawns as dummy objects that moved to offsets around the real mover. This made moving all players very simple as we now had one unified piece of code to deal with.

Finally there was the matter of applying the game logic to the player's movement. As our movement was going to go from one state to another naturally we built our player movers designed around a state machine system. We used Unity's coroutine system as the basis for our state machine. This allowed us to write code that could stop computing on one frame, and resume computing on the next. By doing this we could write easy to understand state code that would wait on conditions to be meet with a simple loop rather than requiring complicated code to return back to the same point of execution.

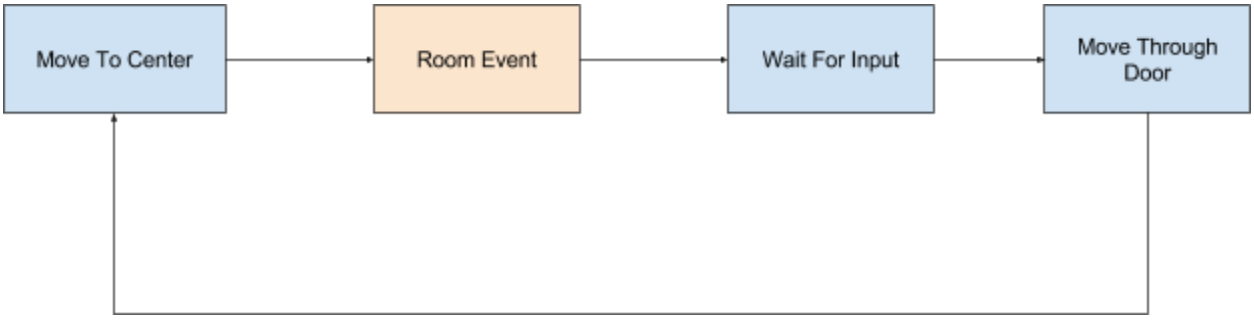


Figure 31: Map movement state machine

The *Room Event* state seen in Figure 31 ended up being a part of the room's code rather than the mover. This gave us the ability to completely change the behavior of this step in the state machine with very little effort.

To make this work over the network we decided to follow a strict authoritative server model. Rather than running the state machine on all clients and syncing the transitions, we only ran the state machine on the master client and informed the other clients of information they needed to know. Clients thus would be informed of what state the machine was on, but would have no actual logic for what to do. This allowed us to easily change the code for the state machine without worrying that clients code would break.

J. Music and Sound Managers

In order to play music in-game, we needed to program our own music system, as Unity does not have a built-in way of handling music without code provided by a developer. Over the course of *Storybook's* development, the way in which we structured the music system changed several times, resulting in a constant shift in the way in which we designed the music manager.

To handle playing music over the course of the game, we developed a Music Manager. The Music Manager's primary job is to handle storing all music tracks in the game, as well as switching between them. We put all the music tracks into the Music Manager as individual members- i.e. one audio file per music track instead of a single list holding all of them. Because of the low music track count in *Storybook*, keeping track of individual pieces of music is relatively easy. Additionally, we added class members to the Music Manager that allowed us to revert to a previous track easily, which is particularly useful for going in and out of combat. The Music Manager primarily works by listening for various types of events that would signal a

change in music, and calling a function to fade into the appropriate track. For example, when the players enter a new room, the room will send an event stating that the players have entered a new room to the Music Manager. The room also sends its Genre with this event, so the Music Manager knows which music track to load. Upon receiving this event, the Music Manager sets the current music track based on the Genre passed in. The Music Manager behaves similarly when it receives an event stating that combat has started or ended. If combat has started, it simply changes the music track to the combat music. If combat has ended, the Music Manager will receive an event stating so, and it will play the music track stored as the last track played. This event-driven architecture for *Storybook's* Music Manager is a relatively clean and simple way to handle changing music, though it only came about through a series of redesigns.

Originally, the Music Manager was an isolated entity that had no knowledge of all music tracks in the game; the different types of Rooms stored their own music. In theory, we thought this would be an easy way to customize rooms, as each room could have had its own unique overworld and combat themes. Whenever the game required the music to change, the appropriate object would directly ask the Music Manager to fade into the desired track. Obviously, this structure is an extremely bad way to design code, as it allows any object with audio files to potentially hijack the music system, so further in the development cycle, we chose to redesign the Music Manager.

As the Alpha version of *Storybook* neared completion, we chose to restructure the Music Manager. For this redesign, we chose to store combat music and the last music track played inside a Combat Instance upon its creation. The Combat Instance then handles changing the music to and from combat. However, we felt that this implementation was too complex and convoluted to use, so we planned to redesign it once more into the state it is in now.

To handle playing the title theme, we created an alternative Music Manager, called the Persistent Music Manager. We used a different object to play the title theme because we felt the base Music Manager and the Persistent Music Manager served different purposes. The base Music Manager is designed to handle music changes in-game, while the Persistent Music Manager is a simpler object that exists to play a single music track across several scenes. It lacks any sort of fading functionality, and it only plays the title theme until the game starts. From that point, the Music Manager will take over the job of playing music for the rest of the game.

In order to play sound effects, we developed the Sound Effects Manager. It is used for one-shot sound effects, such as the clicking sound used in the user interface. Because it is only concerned with playing single-use sound effects without volume control, fading, etc, the Sound Effects Manager is much simpler in its design than the Music Manager. All it needs is a field for each type of simple sound effect as well as a function to play them once.

7. User Testing

A. Testing Process

For playtesting, we decided that we wanted to test both the tutorial game mode, as well as a multiplayer game with two players together. As a result, our process was have players go through the tutorial, and then join a multiplayer game with another player involved in the testing. To recruit playtesters, we sent out an email to CS and IMGD majors at WPI, with the incentive being a gift card. We had them sign up for time slots so that the process was organized, and so users could sign up with a friend to play the game together. If only one person was signed up for a time slot, a member of our team played the game with the tester.

While players were playing the game, we took notes on the actions that they were taking and any questions that they might have had. We waited for players to complete the tutorial, and then once they were both done, they started up a multiplayer game. In the multiplayer game, we had they go through a few rooms together, typically four or five, and then had them stop so they had time to fill out the survey. The survey contained a variety of questions, covering the UIs, difficulty, tutorial and gameplay.

B. Results

I. Tutorial

The results from the tutorial section was that players felt decently prepared to start a game after playing the tutorial, with the average score being three out of five, with five being the best. From the responses, it appeared that the Deck management section was weakly covered in the tutorial, as players seemed to have a hard time understanding the difference between the

Deck and the other Pages in their inventory. Also, some players noted that a few of the UIs could have been explained better, such as the shop or the Deck management menu.

II. Gameplay

Since Pages are the core mechanic of our game, most of the gameplay questions asked were about the Pages. From the results, it appears the people enjoyed the idea of Pages and how they are used in game, particularly using Pages to build rooms. However, players seemed to have some trouble understanding the differences between Pages and what their effects in combat would be. For example, people were unsure which color corresponded to which Genre, with some players having to ask a couple times during the playthrough.

III. User Interface

The final section of the survey covered the User Interfaces in the game, asking about the quality of each of the different screens in our game. The results here were positive, with most people saying that the combat UI, setting up a game, building a room with a Page UI, and the shop UI were all easy to use and understand, with the average scores being around four out of five. The menu that got the least positive feedback was the Deck management UI, averaging around three out of five.

C. Changes Made Based on Feedback

After completing the playtesting, we analyzed the survey results and decided on some changes that we would make to the game based on the feedback. We noticed that the two major areas that needed changing were the Deck management UI and explanation in the tutorial, as well as the visual representation of Pages.

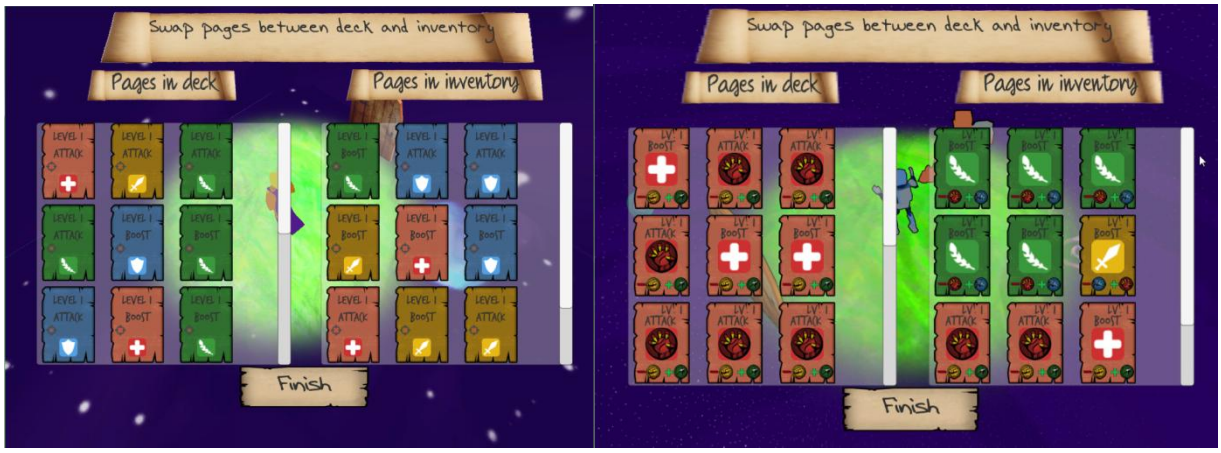


Figure 32: Deck management UI before and after feedback
In the left screenshot, the Pages are not sorted in the menu, and in the right one, they are sorted by Genre

To improve the Deck management system, we first improved the messages that were displayed in the tutorial, based on some of the difficulties people were having with it. People noted that they felt overwhelmed when they first opened up the Deck management menu due to the number of Pages in it, and one of the suggestions was sorting the Pages. As a result, we decided to sort all of the Pages in most of the menus in the game by Genre, which makes it easier to comprehend the contents of the Deck when viewing the Deck management UI. We also made the Deck size smaller, from twenty to fifteen, so that it is easier for players to remember the Pages that they have in their Deck. Finally, to make the characters feel more unique, each player's Deck contains only Pages that match the Genre of the character they have chosen, with some extra randomly generated Pages in the inventory for them to choose to place in their Deck.

As mentioned in the previous section, players were having a hard time remember what the effects of Pages would be and what each color represented. As a result, we decided to redesign how the Pages were displayed in game. One of the key changes was that in addition to the color, we added an icon for the Genre on the Page. In addition, we also display which Genre

a Page would be effective against, and which Genre it would be weak against if it is an attack Page. A comparison of the original Page design and the redesign can be found below.

Figure 33: Original Page design (left) versus redesign (right). Page design was overhauled after many playtesters felt that Pages did not convey enough information.

8. Post Mortem

A. Evolution of Design

As is expected with any game project, the design of *Storybook* changed numerous times over the course of its development. When we started high-level design discussions early in development, we originally planned to make *Storybook* more of a role-playing game with dungeon crawler elements. The initial theme was drastically different too. Early ideas for *Storybook* included crossovers of different time periods, mythologies, or cultures instead of the final decision of literary genres. Shortly afterwards, we settled on the concept of a crossover of literary genres, which we decided was a much more unique idea.

One of the biggest changes we made from early design was the focus on multiplayer. Many of our notes from early development state that the game was designed with a single-player focus, but that we would allow for multiplayer. The initial goal was to create a game that was meant to be played alone. Multiplayer would be implemented and balanced after we created a single-player experience that was fun and satisfying to playthrough. We shifted the focus to a multiplayer game early in development, as the programmers on our team felt that making a solely single-player game would not pose much of a technical challenge and may not have been as much fun to play.

Initial ideas for gameplay were drastically different than what the current game became. Early concepts for *Storybook* were much more like an exploration-based role-playing game with heavy dungeon crawler elements. Like in the final game, players would still build the world as they progress. However, the rooms were much bigger, allowed for free movement, and were far more diverse than in the final game. At this point in development, we had decided on the theme of literary genres, but we had not come up with the idea of universal Pages. Originally,

Storybook was going to feature a wide array of items, broken up into three categories: Pages, which only served to build rooms; Equipment, which worked like armor in a role-playing game; and Active Items, which were one-use items that provided some sort of bonus during combat.

Puzzles featured much more prominently in early design discussions as well. In early designs of *Storybook*, not every room featured enemies. Some rooms would require characters to use abilities unique to their Genre to complete puzzles which would unlock shortcuts through the dungeon, reveal a hidden treasure, or provide some other bonus. Because the abilities required to complete these puzzles was meant to be character-exclusive, these puzzles would only provide additional bonuses to the player; they were not intended to halt progress.

Early designs of *Storybook* allowed players to save a record of a victorious playthrough, in a way, by saving a single item that the player held in their inventory at the end of the game. The player would then have the option to keep that item as a special permanent item that would persist across playthroughs. At the start of subsequent games, players would be able to choose to start with one of their permanent items.

The last major feature that was cut during development was a bestiary. Bestiaries feature in many role-playing games as ways for players to view detailed information or get extra lore about foes they have encountered. We originally planned for a much wider variety of enemies in *Storybook*. Upon encountering an enemy for the first time, players would automatically record details on that monster in their bestiary. In a multiplayer game, players' bestiaries would synchronize. Content inside a bestiary entry would include the enemy's level, statistics, and strategies to defeat them. The concept of a bestiary did not last long enough for us to consider things such as completion rewards.

B. Change in Scope

The key to making a well-polished game in a short time span is to keep the scope of the game manageable. Several times over the course of *Storybook's* development, we made changes to the scope of the game in order to afford enough time for balancing, as well as simply to cut features we felt were unnecessary or did not fit right.

I. Focus of Game

One of the biggest changes in the scope of the game was the focus on making it a more condensed experience. Before we began programming the game, our advisors commented on the monumental difference between having a more freeform style of exploration - like in our original design - versus the more restrictive and condensed style of exploration present in the final game. By this point, we were set on making a multiplayer-focused game, so the technical hurdles we would have to overcome with freeform exploration were even more daunting than in a single-player game. In a multiplayer game featuring a more open world, allowing players to freely roam around, there were so many additional problems to consider. What happens if one player enters combat, but not the rest of the party: Are they all dragged in, or can they join whenever they want? What happens to players when they enter combat: How are they represented on the overworld, if at all? Is the party allowed to split up and have players in more than one room at once?

Since a mobile build was always one of our goals over the course of development, we had to take into account the amount of processing power a game of this scale would use as well. Allowing players to freely roam about the world meant that each phone now had to keep track of up to four players' worth of constantly changing positions, inventories, status, etc. For a PC-based game, this type of multiplayer gameplay is certainly feasible and manageable, but for a

mobile phone, however, we felt that the technical requirements were just too much. Dropping the open exploration style of gameplay also allowed us to design a game of a much more manageable scope, making our goal of creating a finished and well-polished game much more attainable.

In addition to dropping the open world gameplay, we decided to cut the number of items by making Pages universal. In the original design, Pages were a bit of an afterthought, as they only served the purpose of building new rooms, serving no other purpose beyond that. Equipment originally served as the passive stat boosts for players, and Active Items took a role similar to Boost Pages in the current game. As we delved more and more into the literary themes behind *Storybook*, we thought that making Pages an all-purpose item would both be an interesting gameplay mechanic as well as easier to develop and balance. By combining all the purposes of Equipment, Active Items, and the old Pages into a single item, players now have to weigh the value of each Page strategically as it serves multiple purposes.

II. Room Types and Features

Our early notes for *Storybook* listed a wide variety of rooms that could appear throughout the dungeon. While the final game only includes four types of rooms (the starting room, combat rooms, shops, and exit/boss rooms), the original plan was to implement over twice as many special types of rooms. Among the cut types of rooms were the following:

- Curse room: Contains more challenging foes than usual, but will also drop a powerful cursed item. Cursed items cannot be dropped unless the player holding it enters a Sanctuary room.
- Empty room: Contains nothing of note. Unlike the final game, where the only featureless room is the Start room, Empty rooms could be placed anywhere in the dungeon.

- Sanctuary room: Removes the “cannot drop” effect from Curse room items. Can randomly drop items of above-average quality.
- Skull room: Contains boss-level enemies. Unlike the final game, where the boss only appears at the exit, boss rooms could be placed anywhere in early *Storybook*.
- Speed room: Enemies walk faster in the overworld and have a higher Speed stat in combat.
- Multiplier room: Contains greater quantity of enemies. Enemies are more likely to pursue the player in the overworld.
- Teleport room: Acts as a shortcut between other Teleport rooms on the floor.

Since we decided to cut out much of the open world exploration and item-based gameplay, we either redesigned or cut these rooms entirely. Since the short play sessions are a key part of *Storybook’s* design, we figured that having too many room types - while adding diversity to the challenges the players encounter - would cause dungeons to become much larger than they needed to be. In a game of a much larger scope, many of these rooms could have fit it, but since *Storybook* was designed to be of a small scale, having such a large variety of rooms seemed out of place.

C. Change in Art Direction

One of the serious challenges we faced throughout the course of the project was working with a small art staff. With three programmers and a single artist, we had to make do with whatever we could get. For part of the year, we had the assistance of a few artists who contributed work to *Storybook* as part of an independent study program. However, despite the contributions of these artists, we still planned far more assets than we would ever be able to produce. Early in development, we created a master list of art assets that we wanted to create for

the game. Among these assets was a fully modeled, rigged, and animated playable character for each of the four Genres; as well as three common enemies and a boss for each Genre. That doesn't account for UI assets, skybox, textures, embellishment, and other necessary assets. We had to cut most of the characters simply because creating good-looking, fully-animated models of nearly two dozen characters is simply too much for a single artist on this sort of time scale. In the end, we decided to design enemies as palette swaps of the player characters, featuring a darker or more sinister-looking texture.

D. Final Result

The final version of *Storybook* is something incredibly different than what we set out to create. Over the course of its year of development, *Storybook* evolved from a single-player-focused action/exploration role-playing game with dungeon crawler elements to a multiplayer-focused dungeon crawler with more linear progression. We faced practically every major hurdle a game can face during development. Overambitious plans resulted in design overhauls on more than one occasion. A wild overestimation of the feasibility of creating art assets caused us to find creative solutions to implementing enemies into the game. What resulted was a valuable learning experience. Every step we took to forming the final version of *Storybook* was a teaching moment that helped us learn a bit more about the dos and don'ts of game development. Most importantly, however, we feel that we succeeded in creating what we originally set out to make: a small-scale game that has been finely polished to show not only our game development skills but also our attention to and care for the small details.

Appendix

A. List of Definitions

What follows is a list of commonly used terms that will appear in this paper, alongside their definitions.

Dungeon crawler: Dungeon crawlers are games that chiefly involve the player exploring and fighting their way through a virtual dungeon. Many dungeon crawlers are procedurally generated, meaning that the game builds the dungeon at the start of a play session instead of using a pre-made layout (41).

Role-playing game: The most common definition of a role-playing game - also called an RPG - is a game where a player assumes the role of a particular character. Common elements of RPGs include leveling, where a player has various statistics that increase over the course of play; menu-based combat; and a major central storyline that the players pursue (42).

Rogue-like: A game that features a combination of many of the following elements: procedurally-generated worlds; a singular game world where all actions take place; dungeon-crawler gameplay; turn-based combat; permanent choices and failure (43).

B. Survey and Results

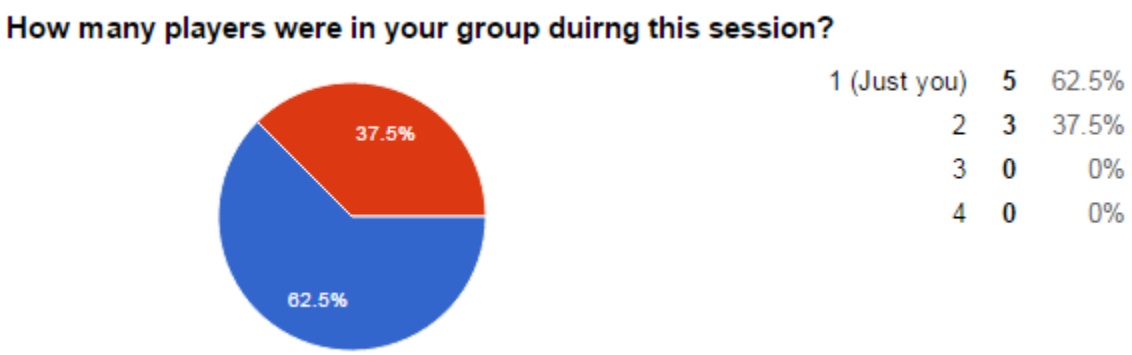
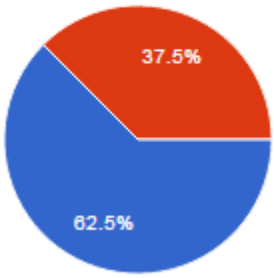


Figure A-1: Survey question 1

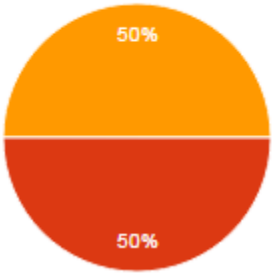
How many playthroughs did it take for you to complete the demo by winning?



1	5	62.5%
2	3	37.5%
3	0	0%
4+	0	0%

Figure A-2: Survey question 2

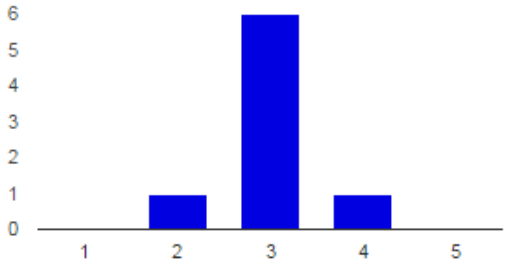
Did you prefer a solo-type game (like in the tutorial) or playing with someone else?



Solo	0	0%
Multiplayer	4	50%
Other	4	50%

Figure A-3: Survey question 3

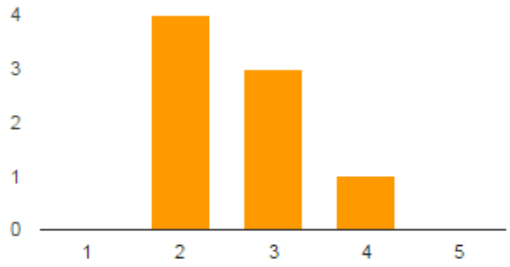
The demo is a work-in-progress of the first level of Storybook. Do you feel it is too long? Too short?



Too Short: 1	0	0%
2	1	12.5%
3	6	75%
4	1	12.5%
Too Long: 5	0	0%

Figure A-4: Survey question 4

Do you feel the game was too easy? Too hard?



Too Easy: 1	0	0%
2	4	50%
3	3	37.5%
4	1	12.5%
Too Difficult: 5	0	0%

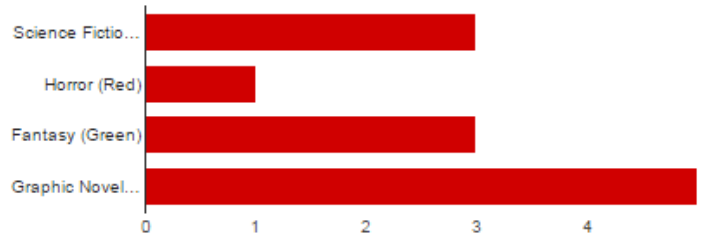
Figure A-5: Survey question 5

(OPTIONAL) Feel free to elaborate on what you felt about the difficulty.

It felt very luck based. If you had like a bunch of Comic Book Boosts and Horror Boosts, you could pretty much live anything. You just needed to add in a couple of attack cards and your deck was set.
A little on the easy side at first but I think it gets harder. didnt get that far though.
I think the health is well balanced right now so that it doesn't become immediately apparent, but your health will slowly wear down without good cards.
Randomized deck combined with hard to remember (without some way to reference it) rock-paper-scissors of types, along with having 2 enemies per room very quickly, made it feel more frustrating than it should have been.
In my first playthrough, I did not have very good luck with RNG
It's fine for a first level, but would need to ramp up a bit later.

Figure A-6: Survey question 6

Which of the characters did you play the game as? (Check all that apply)



Science Fiction (Blue)	3	37.5%
Horror (Red)	1	12.5%
Fantasy (Green)	3	37.5%
Graphic Novel (Yellow)	5	62.5%

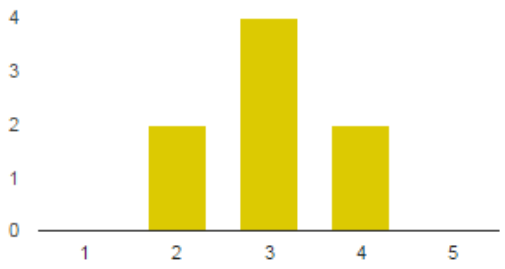
Figure A-7: Survey question 7

(OPTIONAL) If you played as more than one character, did you feel the characters played differently?

not really. they effected what rooms I picked, but mostly it felt like attacking was the best move.
Not especially, I felt like the main choices were which cards should I use to defeat the opponent, but then again I was focused on attack because of that and best fit the graphic novel character. I can see how the horror character would play a good support though.
They didn't feel too different

Figure A-8: Survey question 8

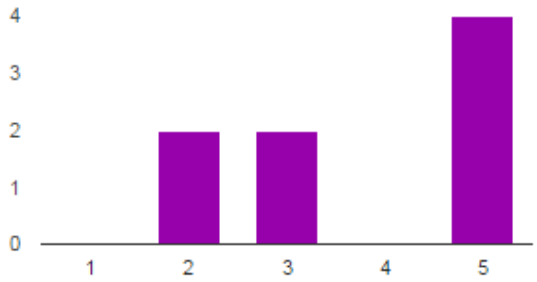
How Informative was the tutorial?



Not Very Informative: 1	0	0%
2	2	25%
3	4	50%
4	2	25%
Very Informative: 5	0	0%

Figure A-9: Survey question 9

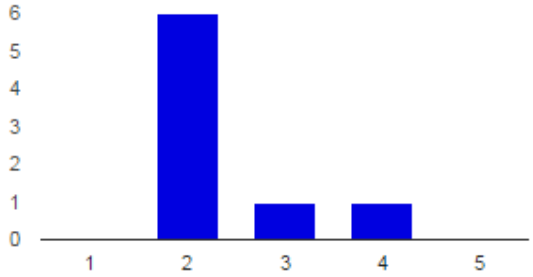
How easy was it for you to find or create a multiplayer game?



Very Difficult: 1	0	0%
2	2	25%
3	2	25%
4	0	0%
Very Easy: 5	4	50%

Figure A-10: Survey question 10

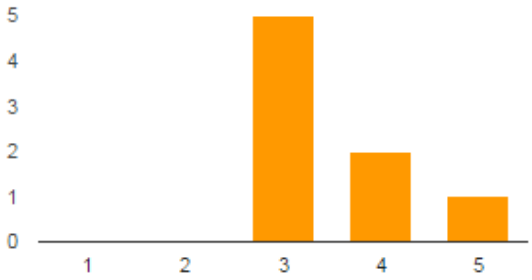
How well did you understand the concepts of your Deck and your Inventory?



Not Well At All: 1	0	0%
2	6	75%
3	1	12.5%
4	1	12.5%
Very Well: 5	0	0%

Figure A-11: Survey question 11

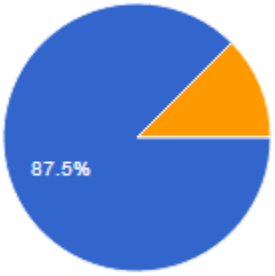
Was the concept of using Pages in-game easy to grasp?



Very Difficult:	1	**0**	0%
	2	**0**	0%
	3	**5**	62.5%
	4	**2**	25%
Very Easy:	5	**1**	12.5%

Figure A-12: Survey question 12

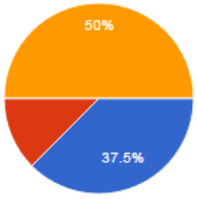
Did you like the concept of using Pages to determine the contents of Rooms?



Yes	**7**	87.5%
No	**0**	0%
Other	**1**	12.5%

Figure A-13: Survey question 13

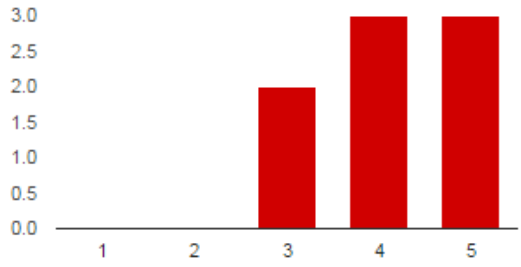
Were you able to easily tell the differences between each type of Page (Attack, various Boosts, Rare vs Common, etc)?



Yes	**3**	37.5%
No	**1**	12.5%
Other	**4**	50%

Figure A-14: Survey question 14

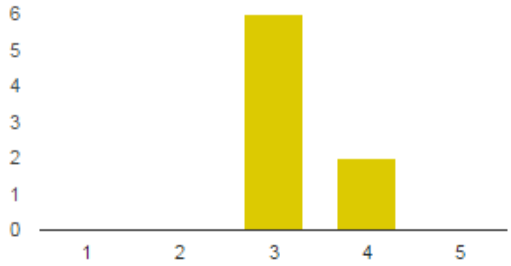
How intuitve was the UI for creating and starting a multiplayer game?



Not intuitive at all: 1	**0**	0%
2	**0**	0%
3	**2**	25%
4	**3**	37.5%
Very intuitive: 5	**3**	37.5%

Figure A-15: Survey question 15

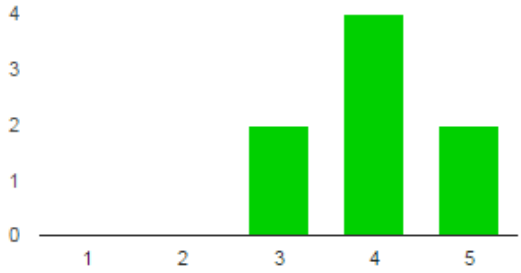
How intuitive was the Deck Management UI?



Not intuitive at all: 1	**0**	0%
2	**0**	0%
3	**6**	75%
4	**2**	25%
Very intuitive: 5	**0**	0%

Figure A-16: Survey question 16

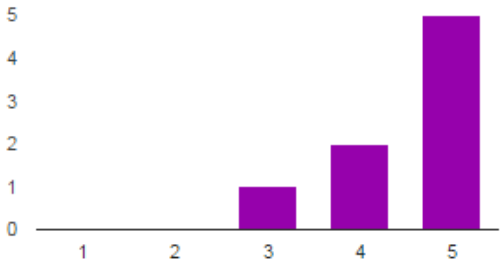
How intuitive was the "Build Room with Page" UI?



Not intuitive at all: 1	**0**	0%
2	**0**	0%
3	**2**	25%
4	**4**	50%
Very intuitive: 5	**2**	25%

Figure A-17: Survey question 17

Was the Combat UI easy to understand and use?



Not intuitive at all: 1	0	0%
2	0	0%
3	1	12.5%
4	2	25%
Very intuitive: 5	5	62.5%

Figure A-18: Survey question 18

If you encountered a Shop, did you understand how to use the UI to trade for Pages?

Yes	8	100%
No	0	0%
Never encountered a Shop	0	0%
Other	0	0%

Figure A-19: Survey question 19

(OPTIONAL) If you have any further comments about any aspect of the User Interface, please let us know!

My only complaint about the deck management UI is that I didn't realize it was click and replace, so I was trying to drag cards around, and that made me really confused. On another note, I recommend sorting the cards to make them easier to view what you have and don't have, because oftentimes I was looking for a particular card in my starting deck and couldn't see how many of them I had because they were spread all over the place.
I didn't understand that only 3 cards could be used in the shop, and though how to play cards was apparent in the combat, I didn't understand until explained to me how exactly the boosts worked with the decrementing per turn and initial expected stat boost. It would be good to know how much health for instance I might expect back from a boost, or how much damage cards will deal because the main skill of this game seems to be counting numbers and calculating number of turns and efficiency in that regard. I didn't immediately grasp that I actually got to keep the inventory at the start of a world for choosing cards for a room or trading.
I failed my first playthrough and was unable to finish the second playthrough

Figure A-20: Survey question 20

(OPTIONAL) Do you have ANY final comments for us about any part of the game?

The game looks fun and interesting, I'm wondering what sort of thing the final, polished version will be.
I think the tutorial broke. Speed seems pretty useless. Attacking generally always seems like the right choice.
It might be helpful to have a reminder left throughout the game that reinforces which colors beat which colors as it was quite easy to forget and not have a reference menu. Perhaps have an ESC pause menu with the explanation in it. I enjoyed playing this game!
Decks should probably be organized by genre and then internally by attack/boost, and then levels, etc., at least when they're first presented to the player. The randomization combined with the scattered types made looking at the deck kinda daunting, especially as it was really unclear whether the given deck was good or if I needed to swap in inventory pages.
I felt that the colors/genres could have been explained better, or through mouseover textboxes during the game.

Figure A-21: Survey question 21

Works Cited

[1] "What Is Casual Gaming?" *What Is Casual Gaming?* Computer Hope, 31 Dec. 2015. Web. 05 Apr. 2016. <http://www.computerhope.com/jargon/c/casual-gaming.htm>.

[2] "Report: Americans Serious About Casual Play." *Report: Americans Serious About Casual Play*. Nielsen, 02 Sept. 2009. Web. 30 Mar. 2016. <http://www.nielsen.com/us/en/insights/news/2009/report-americans-serious-about-casual-game-play.html>.

[3] "Clash of Clans." *Clash of Clans x Supercell*. Supercell. Web. 11 Apr. 2016. <http://supercell.com/en/games/clashofclans/>.

[4] "Angry Birds." *Games | Angry Birds*. Angry Birds. Web. 11 Apr. 2016. <http://angrybirds.com/games/>.

[5] "Trivia Crack." *Trivia Crack*. Etermax. Web. 11 Apr. 2016. <http://triviacrack.com>.

[6] "Rogue (Game) - Giant Bomb." *Rogue*. GiantBomb. Web. 11 Apr. 2016. <http://giantbomb.com/rogue/3030-22309/>.

[7] "The Binding of Isaac." *The Binding of Isaac on Steam*. Steam. Web. 11 Apr. 2016. <http:// http://store.steampowered.com/app/113200/>.

[8] "Super Smash Bros." *Official Site - Super Smash Bros. for Nintendo 3DS/WiiU*. Nintendo/Bandai Namco Entertainment/Sora Ltd. Web. 11 Apr. 2016. <http://smashbros.com/us/>.

[9] *Screenshot of character selection from* Super Smash Bros. for WiiU. *2014.* Video game screenshot. <http://img15.deviantart.net/0050/i/2015/318/6/a/smash_wii_u_css_with_cloud_by_gameonion-d9goef2.png>.

[10] "Core Team Members." *Core Team Members | Characters | Marvel.com*. Marvel. Web. 11 Apr. 2016. <http://marvel.com/characters/list/986/core_team_members?&options[offset]=0&totalcount=17>.

[11] "Man of Steel: Superman." *Superman | DC Comics*. DC Comics. Web. 27 Apr. 2016. <http://www.dccomics.com/characters/superman>.

[12] "Captain Marvel." *Captain Marvel | Characters | Marvel.com.* Marvel. Web. 27 Apr. 2016. <http://marvel.com/characters/9/captain_marvel>.

[13] "Halo." *Halo - Official Site*. 343 Industries/Microsoft Studios. Web. 27 Apr. 2016. <https://www.halowaypoint.com/en-us>.

[14] "Metroid." *Metroid Prime: Federation Force for Nintendo 3DS - Official Site*. Nintendo. Web. 27 Apr. 2016. <http://metroidprime.nintendo.com/>.

[15] "Star Wars." *Star Wars.com | The Official Star Wars Website*. Lucasfilm Ltd. Web. 27 Apr. 2016. <http://www.starwars.com/>.

[16] "Robin Hood." *Robin Hood - Wikipedia, the free encyclopedia.* Wikipedia. Web. 27 Apr. 2016. <https://en.wikipedia.org/wiki/Robin_Hood>.

[17] "Frankenstein." *Frankenstein - Wikipedia, the free encyclopedia.* Wikipedia. Web. 27 Apr 2016. <https://en.wikipedia.org/wiki/Frankenstein>.

[18] "Strange Case of Dr. Jekyll and Mr. Hyde." *Strange Case of Dr. Jekyll and Mr. Hyde - Wikipedia, the free encyclopedia*. Wikipedia. Web. 27 Apr. 2016. <https://en.wikipedia.org/wiki/Strange_Case_of_Dr_Jekyll_and_Mr_Hyde>.

[19] "Hulk." *Hulk | Characters | Marvel.com*. Marvel. Web. 27 Apr. 2016. <http://marvel.com/characters/25/hulk>.

[20] "Create and Connect with Unity." *Unity - Game Engine*. Unity. Web. 27 Apr. 2016. <https://unity3d.com/>.

[21] "Unreal Engine." *What is Unreal Engine 4*. Epic Games. Web. 27 Apr. 2016. <https://www.unrealengine.com/what-is-unreal-engine-4>.

[22] "GitHub." *GitHub*. GitHub. Web. 27 Apr. 2016. <https://github.com/>.

[23] "Trello." *Trello*. Trello. Web. 27 Apr. 2016. <https://trello.com/>.

[24] "HacknPlan." *HacknPlan | The project planning tool for game developers*. HacknPlan. Web. 27 Apr. 2016. <http://hacknplan.com/>.

[25] "Slack." *Slack: Be less busy*. Slack. Web. 27 Apr. 2016. <https://slack.com/>.

[26] "Unity Cloud Build." *Unity - Services - Cloud Build*. Unity. Web. 27 Apr. 2016. <https://unity3d.com/services/cloud-build>.

[27] "Google Calendar." *Google Calendar - Get the new app for Android and iPhone*. Google. Web. 27 Apr. 2016. <https://www.google.com/calendar/about/>.

[28] "Yu-Gi-Oh! Trading Card Game." *Yu-Gi-Oh! TRADING CARD GAME*. Konami. Web. 11 Apr. 2016. <http://www.yugioh-card.com/en/>.

[29] "Magic: The Gathering." *MAGIC: THE GATHERING*. Wizards of the Coast. Web. 11 Apr. 2016. <http://magic.wizards.com/>.

[30] Takahashi, Kazuki. *Blue-Eyes White Dragon*. 1996. Trading card artwork. <http://940ee6dce6677fa01d25-0f55c9129972ac85d6b1f4e703468e6b.r99.cf2.rackcdn.com/products/pictures/130170.jpg>.

[31] Palumbo, Anthony. *Elgaud Shieldmate*. 2012. Trading card artwork. <http://ecx.images-amazon.com/images/I/61jRdQAFb%2BL._SY355_.jpg>.

[32] "Pokémon." *The Official Pokémon Website | Pokemon.com | Explore the World of Pokémon*. The Pokémon Company. Web. 11 Apr. 2016. <http://www.pokemon.com/us/>.

[33] "Fire Emblem Fates." *Fire Emblem Fates for Nintendo 3DS - Official site*. Nintendo/Intelligent Systems. Web. 11 Apr. 2016. <http://www.fireemblemfates.nintendo.com>.

[34] "Words With Friends." *Words with Friends | The No.1 Mobile Word Game! | Zynga*. Zynga. Web. 27 Apr. 2016. <https://www.zynga.com/games/words-friends>.

[35] "Minecraft." *Minecraft*. Mojang Syngergies AB. Web. 27 Apr. 2016. <https://minecraft.net/>.

[36] "Kirby 64: The Crystal Shards." *Kirby 64: The Crystal Shards - Wikipedia, the free encyclopedia*. Wikipedia. Web. 27 Apr. 2016. <https://en.wikipedia.org/wiki/Kirby_64:_The_Crystal_Shards>.

[37] "Maya." *Maya | Computer Animation & Modeling Software | Autodesk*. Autodesk. Web. 27 Apr. 2016. <http://www.autodesk.com/products/maya/overview>.

[38] "Photoshop." *Photoshop Inspiration, Photoshop Information | Photoshop.com*. Adobe Systems Incorporated. Web. 27 Apr. 2016. <http://www.photoshop.com/>.

[39] "Tangent Music." *Tangent Music."* Tangent Music, LLC. Web. 27 Apr. 2016. <http://www.tangentmusicllc.com/>.

[40] "Photon." *Photon Unity 3D Networking Framework SDKs and Game Backend | Photon: Multiplayer Made Simple.* Photon Unity Networking. Web. 27 Apr. 2016.

[41] “Dungeon-crawler.” *Dungeon-crawler dictionary definition | dungeon-crawler defined*. YourDictionary. Web. 13 Apr. 2016. <http://www.yourdictionary.com/dungeon-crawler>.

[42] “Role-Playing Game (RPG).” *What is Role-Playing Game (RPG)? - Definition from Technopedia*. Technopeida. Web. 13 Apr. 2016. <https://www.techopedia.com/definition/27052/role-playing-game-rpg>.

[43] “What a roguelike is.” *What a roguelike is - RogueBasin*. RogueBasin. Web. 13 Apr. 2016 <http://www.roguebasin.com/index.php?title=What_a_roguelike_is>.