Shoot'em up game "Zenith of Lead"

by

Cortez Curtis Gray

B.S., Emporia State University, 2016

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Approved by:

Major Professor
Daniel Andresen

Copyright

Abstract

This paper presents an implementation for an idea within the shoot'em up (SHMUP) genre of games. It was developed to try and present a unique take on the genre. So, this paper provides a solution which is focused on designing gameplay which is as fast paced as possible while implementing some original mechanics. To do so we evaluate other games within the genre and how they could be innovated upon. We also go over the design of the game itself and how its performance metrics were evaluated. The implementation specified uses the game engine Game Maker Studio 2 to deliver on its premise.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I want to thank Dr. Daniel Andresen who is my major professor that constantly helped throughout this master's report. I would also like to thank the Superpowers project which is a free, open source project that provided various assets used within the game [20]. Finally, I would like to thank Dr. Joshua Weese and Dr. William Hsu for being a part of my committee.

Chapter 1 - Introduction

The vision for the game was to try and blend a Twin-Stick SHMUP with a Bullet-Hell type SHMUP. Blending other genres with Bullet-Hells is something that is rarely done, so it presented a unique opportunity to test my skills. Not only was it a skills test, but also an opportunity to learn new things about game development. For example, how to create a game design document, build modular systems, and how to handle a medium-scale software project.

The rest of this paper discusses related work and genre history in Chapter 2. Then we describe our implementation and overall game design theory in Chapter 3. Chapter 4 describes the testing practices, how we evaluated our system, and presents the results of a performance test. Chapter 5 presents our conclusions and describes future work.

Chapter 2 – Related Work

So, what is a shoot-em-up? A shoot-em-up (SMHUP) is usually defined as an entity which moves around the screen shooting a large number of enemies while also avoiding enemy bullets [2]-[3]. They usually include some kind of scoring mechanic and an extra lives system. This genre of games is also one of the oldest in gaming. Setting the stage for many common game design ideas we see today.

2.1 Asteroids

Figure 1: Asteroids Start Screen [12]

Figure 1 shows off one of the first multi-directional SHMUPs, “Asteroids” (1979) [8]. This genre allows the player to freely rotate their ship to fire in certain directions and allows movement along the x and y-axis. This concept would eventually evolve into what we call Twin Stick SHMUPs. Where the player can move or face in any direction while firing in another. One of the most popular in the modern day is a game called “Geometry Wars”. This genre was one of

the main ones I studied while developing my own game. I wanted to blend a Twin-Stick and a Bullet-Hell SHMUP to create something unique. So, I learned from various games in order to learn what movement felt good, how enemies functioned, how sound should be balanced, etc.

2.2 Galaga

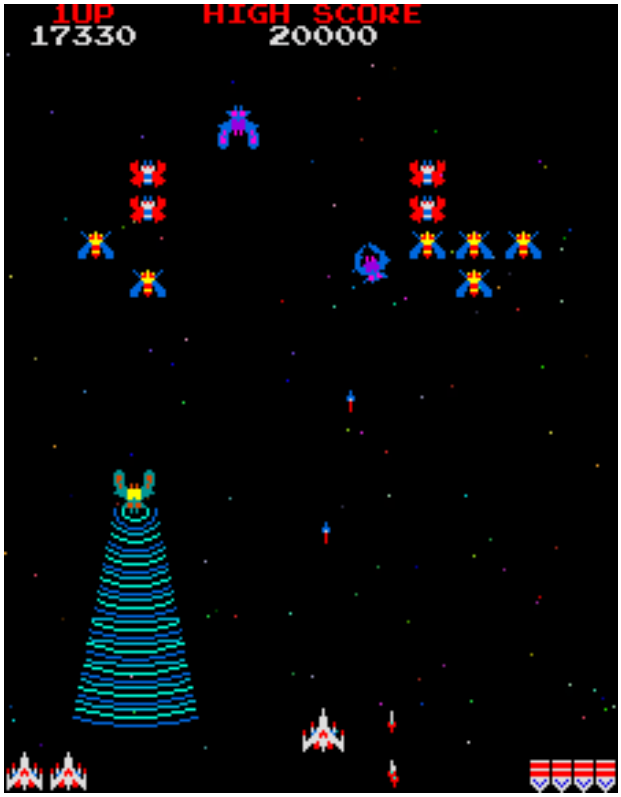

Figure 2: Galaga Gameplay Screenshot [13]

Figure 2 shows one of the most well-known SHMUPs in the genre, “Galaga” (1981) [4]. This is an example of a fixed-screen SHMUP. Where the screen does not move and instead enemies funnel into preset positions. In this type of SHMUP, the player is constrained to just moving left or right along the x-axis. Also, the player can only shoot in one direction. Along with “Space Invaders”, Galaga is one of the grandfathers of the SHMUP genre. Laying the foundation

for the genre to continuously evolve from. It is always important to study the pioneers in a field, as it helps really understand the fundamentals of said field.

2.3 Xevious

Figure 3: Xevious Gameplay Screenshot [14]

Figure 3 shows off another popular SHMUP, “Xevious” (1982) [5]. Which is a considered a vertical scrolling SHMUP and one of the first in that type of SHMUP. The perspective is presented from a top-down view and player movement is restricted to the x and y-axis. The screen itself scrolls along the y-axis. This type of SHMUP was one of the ones I

considered going with before I chose the genre I did. As this genre of SHMUPs (and the next) is one of the most popular nowadays.

2.4 R-Type

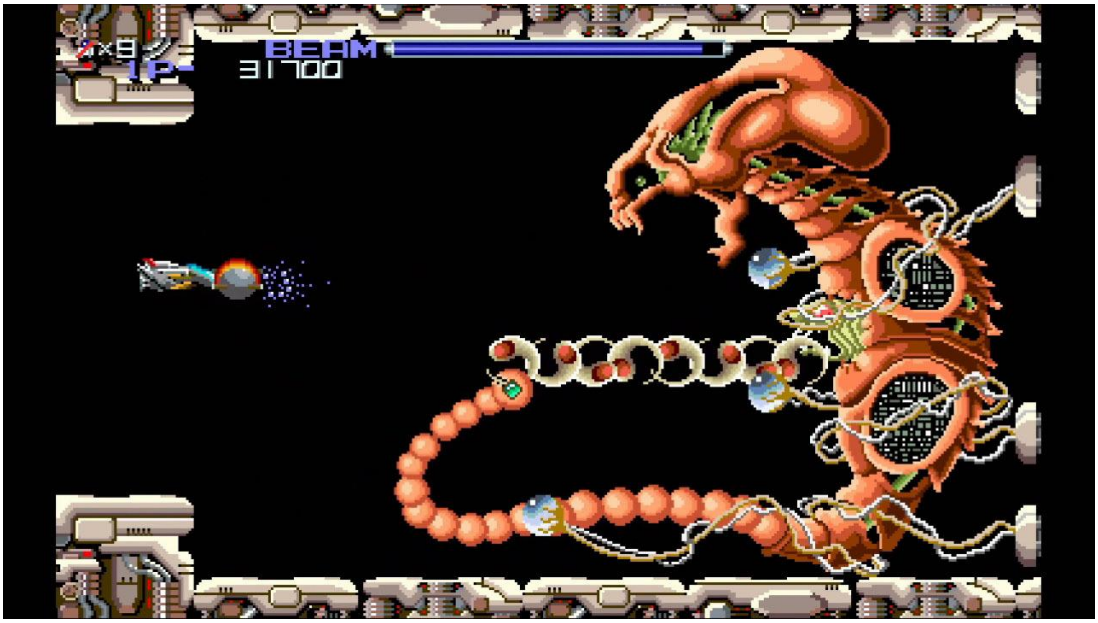

Figure 4: R-Type Gameplay Screenshot [15]

In Figure 4 one can see another well-known SHMUP, “R-Type” (1987) [6]. This is considering a side (horizontal) scrolling SHMUP. Where the screen scrolls along the x-axis and player movement is constrained along the x and y-axis. The perspective of the game is also presented from the side. These type of SHMUPs usually incorporate environmental hazards into the gameplay. Where the background itself is also a danger to the player. I decided not to go for environmental hazards like traps, since the player’s focus was already going to be stretched due to the sheer number of bullets on screen.

2.5 Rez

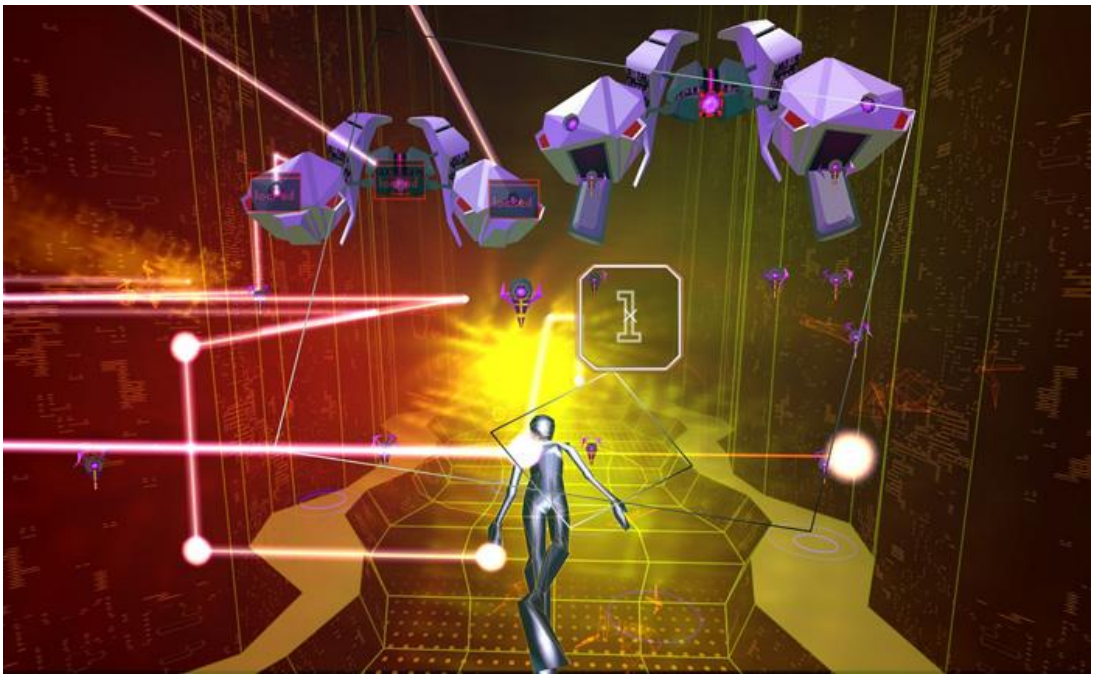

Figure 5: Rez Gameplay Screenshot [16]

The last main perspective can be found in Rail SHMUPs. Which offer a viewpoint akin to looking into a screen. Figure 5 is one such Rail SHMUP, called “Rez” (2001) [7]. Player aiming is constrained to the x and y-axis while the camera moves along the z-axis. Most games in this genre also allow the player to move in the x and y directions, which is commonly used to dodge enemies and/or their bullets. These types of games were good to study because they really showcase how you can guide a player’s eyesight around the screen.

2.6 Mushihimesama

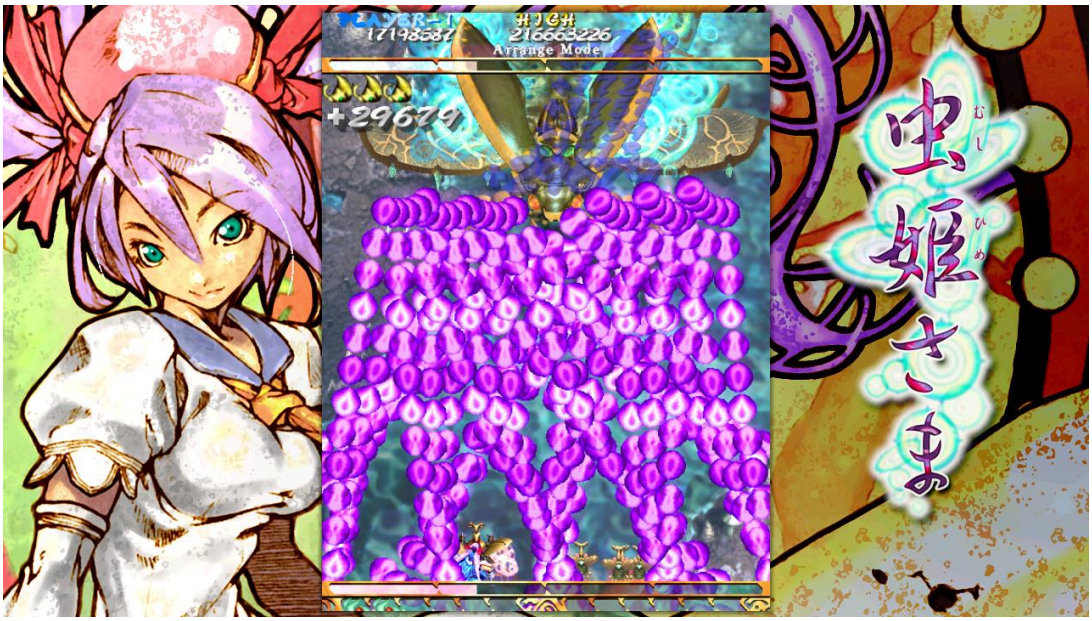

Figure 6: Mushihimesama Gameplay Screenshot [17]

Figure 6 showcases “Mushihimesama” (2004) [10], which belongs to the most extreme take on the SHMUP genre commonly called “Bullet-Hells”. These are characterized by the screen being filled to the brim with bullets, requiring the player’s utmost focus to survive. Another common characteristic with these types is that they have a scoring system which promote playing out of a player’s comfort zone. This can include either having the player move from the bottom of the screen for some reason (as this is usually the safest place to be in these games) or by having certain conditions be met to earn more score.

In the case of Mushihimesama, they use the latter method for people who are into high score chasing. Score bonuses are given for never missing shots, killing a big enemy (mid-stage bosses or end-stage bosses) with many bullets on the screen, or never dying. There is also a counter system which promotes keeping a boss alive as long as possible to gain as much counter from it as possible.

2.7 Ketsui Deathtiny

Figure 7: Ketsui Deathtiny Gameplay Screenshot [18]

As an example of a Bullet-Hell using the method to get a player to move from the bottom of the screen. We have “Ketsui Deathtiny” (2003), which is shown in Figure 7 [11]. It succeeds in moving the player around by awarding significantly more score when the player destroys an enemy at close range. Players are awarded with “chips” whose values range from 1-5 with 5 being the most desirable. The closer you are to an enemy when they are destroyed, the higher number the chip will be. This encourages the player even more to memorize enemy spawn patterns, so they can be at the closet range possible when the enemy appears.

I closely studied this genre of SHMUPs when designing my own game. I wanted some of my enemies to still be able to flood with the screen with bullets but have more of a margin of error than those game are known for. I aimed to accomplish this by having multiple ways to dodge or destroy bullets. I also studied their risk/reward approach to scoring and applied it uniquely to my own design. Where you are awarded score if you are close to enemy bullets when destroying an enemy. Another thing I studied were some of the patterns enemies used when firing their bullets.

2.8 Nuclear Throne

Figure 8: Nuclear Throne Gameplay Screenshot [19]

"Nuclear Throne" (2015) is shown in Figure 8 [9]. This is a top-down Twin Stick SHMUP with roguelike elements; roguelikes are games built around randomization, dying, and retrying. They are meant to be extremely difficult and force the player to learn the ins and outs of a game to succeed. More often than not they will obfuscate their own game systems to make the game even more difficult to learn, leading to multiple deaths until the player learns said systems.

Roguelikes also randomize their enemy and level layouts, which gives the games a much greater level of replayability.

For my game I wanted to focus on presenting the player set levels and enemy patterns. As the intent was to have the player focus on mastering the levels and learning the best way to maximize their score on each level. So, when studying Nuclear Throne, I was more interested in how it handled player movement, shooting, hit feedback, sound, etc. than its overall game design and goals with that design.

Chapter 3 - Implementation

3.1 Overall Design

The structure of the game is built around speed and keeping a steady rhythm. The score system is the main driver for this. It pushes the player to complete combat areas as fast as possible while keeping their chain multiplier high. Keeping the multiplier high is essential for getting the highest scores possible and taking too long or getting hit lowers your chain. To aid the player I came up with a cover system which blocks bullets, ways to destroy bullets, and ways to go under bullets. This is where the rhythm comes into the play, where the player dances around the combat area whilst timing enemy kills to upkeep their chain multiplier.

For development of the game, I implemented an Iterative style of game development. This allowed me to easily adjust and iterate upon multiple aspects of the game. It also had the benefit of increasing my understanding and familiarity with a popular method of software/game development. Iterative was chosen because it was far less restrictive compared to other Software Development Life Cycle (SDLC) models; although Agile could be switched to if I had a consistent group of play testers. For the game itself, I chose to implement it in Game Maker Studio 2 (GMS2). The reason I chose this game engine was because it excels in creating 2-dimensional (2D) games, which is the type of game I planned on creating. Plus, since it is a fairly old engine, it has the benefit of being well documented and studied.

The rest of this section explains more about the game engine it runs on. Some of the design patterns I used when programming the game. The various game actions possible by the player. What each game system is, what they do, and why they were included. How the game handles combat areas, dialogue, and the tutorial. Then the overall story and how it fueled the rest of the game.

3.2 Engine Explanation

The game was created in Game Maker Studio 2 (GMS2). Which is a game development application and engine; the IDE of the application was created in C# and the engine in C++ . Its main focus is the creation of 2D games but has a few capabilities which allow for certain 3D games. The engine itself is largely single threaded with only audio getting to run on separate threads; it is also only uses a single core. It has many different licenses, and each allows ports to different platforms; Windows, Ubuntu, MacOS, Android/iOS, HTML5, and PS4/Xbox One/Switch. The license I used had the capability to port to Windows, MacOS, and Ubuntu [1].

The programming language used in GMS2 is called GameMaker Language (GML). It is a dynamically typed language and is often likened to be a mix of JavaScript and C-style languages. It also has a Drag-and-Drop (DnD) interface which is aimed towards people who do not have much programming experience [1]. This interface is very visual based and allows someone to build a game without programming. I chose to use GML for this game due to my programming background.

3.3 Finite State Machine

The structure chosen when designing the backend for the game was a Finite State Machine (FSM) system. This structure allows entities within the game to function asynchronously without the need for a director-type entity. The system also utilizes parent/child inheritance to increase workflow by decreasing the amount of debugging required and it greatly improves modularity. The rest of the game is controlled through triggers which may be activated by the player, certain game conditions being met, or automatically based on where we are in the

game. These design choices allow the game to run smoothly and make it far easier to add additional enemies/guns/levels/etc. as the game grows in size and complexity.

3.4 Player Systems

Figure 9: Game Controls Menu

In the game there are multiple actions available to the player, with some actions being restricted depending on which state the player is in. Various actions include the player being able to shoot their currently selected gun; switch between two different guns; use their graze meter to fire a special bullet, which is different depending on the currently selected gun; move slower whilst showing their hitbox; slide to go under enemy bullets; melee to destroy enemy bullets or damage an enemy; throw a grenade which damages all enemies in its radius and destroys all bullets it touches; shoot a grappling hook which can grab dizzied enemies; throw a grabbed enemy; create cover with a grabbed or dizzied enemy; and activate a powered up form which

makes guns shoot more powerful bullets with different effects and makes grenades become more powerful. The game's controls are shown off in Figure 9.

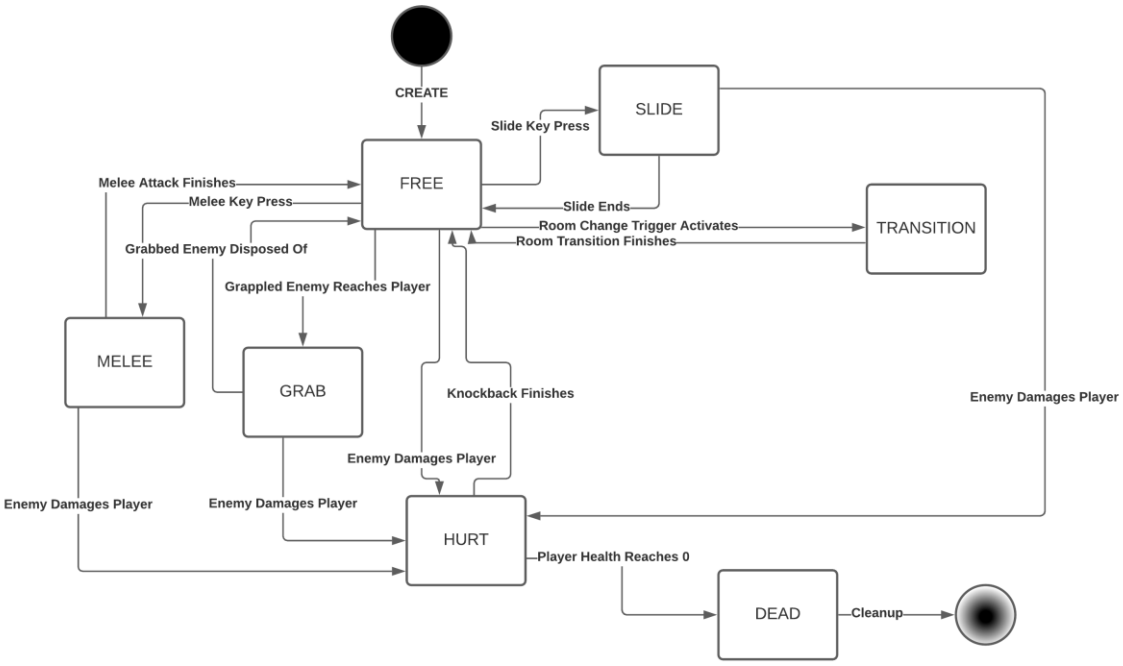

Figure 10: Player State Diagram

Figure 10 displays each state that the player can be in and all the triggers which cause state transitions. There are a couple actions which can be executed in any state (except DEAD) like powering up your guns or activating slow mode. FREE is the default state, and it allows for any action to be taken within it. Sliding, melee attacks, grappling, and throwing grenades are all restricted to the FREE state. Graze shots are restricted to the FREE and GRAB states. Movement is restricted to the FREE, MELEE, and GRAB states. Throwing is restricted to the MELEE and GRAB states. Creating cover is restricted to the SLIDE and GRAB states.

One of the main reasons for implementing the state system was so that I could limit player actions or even change what a button does depending on the state. For example, pressing the melee button in the FREE state causes a melee attack but pressing the melee button in the GRAB state causes the player to throw the grabbed enemy. Limiting the player's control was also useful for implementing things like knockback and sliding, which both move a player in a fixed direction for a certain distance.

3.5 User Interface Explanation

Figure 11: User Interface

Figure 11 shows off the game's User Interface (UI), which displays all relevant information to the player. All information it displays is located in the top left of the screen. Starting at the top and descending we have the player's current score and current score multiplier; health bar which displays current health points (HP) in red; graze meter which has a maximum of 4 chunks; and current number of grenades which is a maximum of 4.

Figure 12: Player Score and Score Multiplier

Figure 12 shows off the player's score (first number) and score multiplier (second number). In the game there are many ways to increase the player's score like destroying enemies; completing a combat room; completing a level; picking up a med kit; picking up a grenade pack; picking up a powerup cube; and destroying an enemy while grazing a bullet. For the medkit, grenade pack, and powerup cube bonuses the amount of score given is based on the "overflow" amount. For example, a medkit gives 25 HP (max HP is 100) and if the player picks up a medkit at 90 HP, they will be given a bonus on that 15 HP (90 - 100 + 25) overflow. The bonus is calculated by ((current HP – max HP) + medkit HP) / medkit HP * 5000. This is intended to give more score the less a player needed the medkit, so picking up the medkit at max HP will give the most amount of score. Which further incentivizes the player to not take damage for an entire level.

These flat score bonuses can be further boosted by utilizing the game's chain multiplier system. The chain multiplier takes a score bonus and multiplies it by itself then adds the result onto the current player score. The multiplier starts off at a base of 1 and increments by 1 from there; the max multiplier is 10. The main way a player can increase the multiplier is by destroying enemies. A counter keeps track of the number of enemies killed this chain, and when the number of destroyed enemies matches the current chain multiplier, the multiplier increments by 1. For example, if we have a chain multiplier of 3 and kill 3 enemies the chain multiplier increments. However, the chain multiplier has a timer of 6 seconds before it resets back to the base multiplier of 1. The timer can either be set back to 6 seconds by destroying enough enemies

to increment the multiplier, or by destroying an enemy which adds 2 seconds. The player can also increase the multiplier significantly by entering the powered-up state, which doubles the current multiplier and increases the base multiplier to 2 while powered-up.

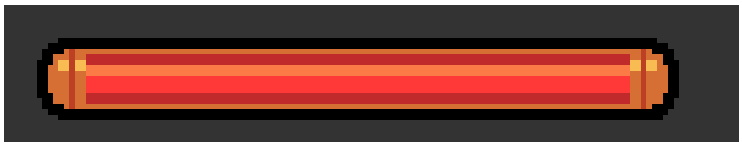

Figure 13: Player Health Bar

The player is only damaged when a bullet touches the player hitbox, all other bullets fly past the player. The player can also be damaged if they touch the enemy's body with their own. In general, when damaged the player will be knocked back, current HP decreases, and they will be unable to shoot or melee temporarily. Figure 13 shows what the health bar looks like. The red portion will always display the current HP the player has and if that bar is depleted the player dies causing the game to end.

Figure 14: Normal sprite (Left), player and graze hitbox showing (Right)

The actual hitbox for the player is only a square of 3x3 pixels located in the center of player sprite. Figure 14 shows the player sprite normally on the left and then the player sprite with the graze hitbox (grey) and player hitbox (red) showing on the right. These hitboxes are normally hidden but can be seen at any time by using the Slow Move button. Not only does it show the player's hitbox, but it also makes them move slower. Which helps them better navigate through denser packets of bullets.

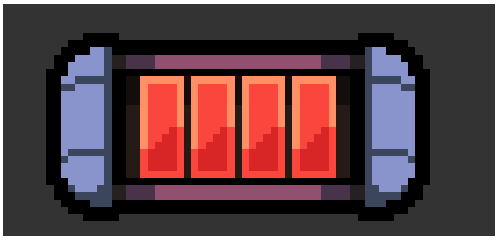

Figure 15: Player Graze Meter

If a bullet touches the graze hitbox, instead of being damaged the player will slowly gain graze meter as long as they are touching the bullet; if a bullet hits the player, they will lose some of their graze meter instead. Figure 15 shows off this graze meter. Players can choose to spend this graze meter in one of two ways; firing a special bullet out of their currently equipped gun, this will spend one chunk of the graze meter (which can build up to a maximum of 4 chunks); or enter a powered-up state where all bullets fired and grenades launched will be stronger and/or gain special effects, however this requires the graze meter to be full and it will consume all of it. There is also a score bonus given out if a player destroys an enemy while grazing a bullet.

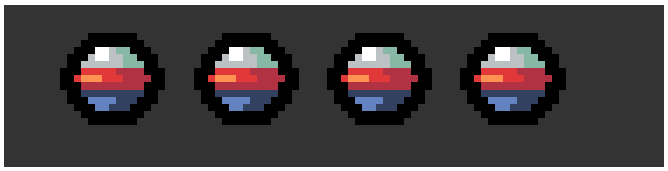

Figure 16: Grenade Counter

Looking at Figure 16 shows off the grenade counter. Grenades can be used to either damage an enemy and/or destroy enemy bullets in a small radius. Each thrown grenade consumes 1 grenade from the grenade counter; these grenades can be replenished with grenade packs that restock 2 at a time. Grenade effects can be further boosted by using them while in the powered-up form. Grenades used in this form destroy all enemy bullets for a certain amount of time and deal damage to all enemies on the screen.

Figure 17: Player Cover System

The final major player system is the cover system. Cover offers 360-degree protection from enemy bullets, and the cover itself is shown off in Figure 17. This protection does not last forever though, the cover itself has its own HP and it has a timer until it gets destroyed automatically. Cover can be created by either using the melee key while grabbing an enemy or

by sliding into a dizzied enemy. Then the cover is created where the player is standing. This offers brief reprieve from the action which allows the player to refocus.

3.6 Enemy Systems

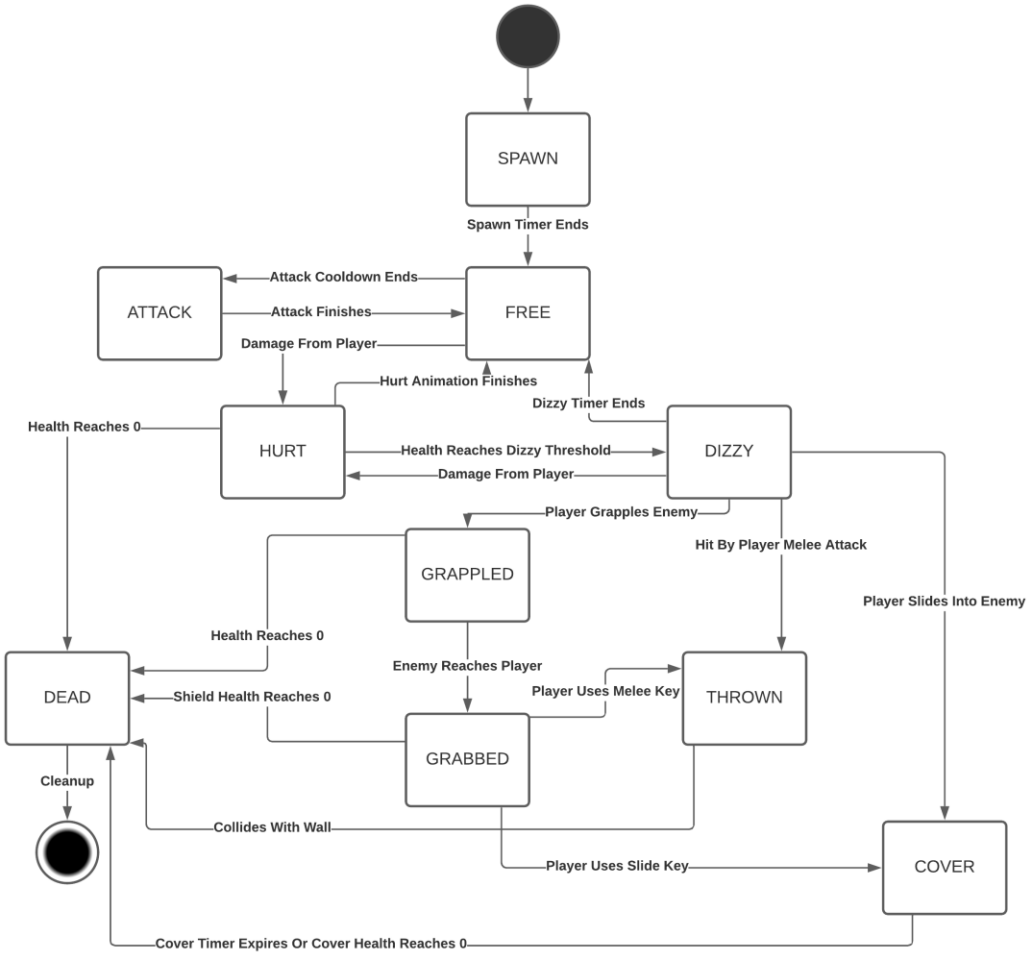

Figure 18: Enemy State Diagram

All possible enemy states and state transition triggers are displayed in Figure 18. Unlike the player state machine, enemy states share no common actions. This allows for a very fine-tuned control of the enemy Artificial Intelligence (AI). Which has the extra benefit of memorizing enemy behavior easier for the player, plus it allows for far easier debugging.

In general, the enemy AI works by first being created and then entering the SPAWN state. This state plays a small animation and makes it so the enemy can't attack, move, be damaged, or deal damage to the player by touching them. This was necessary to implement because enemies spawn in waves and their spawn positions are not known to the player beforehand. So, it was important to implement this way to make sure the player does not get hit randomly, plus it gives them time to get into position to defend themselves. After a short amount of time all enemies automatically transition out of the SPAWN state and into the FREE state.

In the FREE state each enemy has their own AI which determines what they do. In general, an enemy waits for their attack cooldown timer to reach 0 and once it does they initiate an attack; the attack launched depends on things like distance to the player or even just randomly selecting an attack. Once an attack is initiated the data for that attack is loaded and then the enemy transitions to the ATTACK state. Otherwise, if the attack cooldown time is not 0, they go into a function which determines how they should move around.

```
6  attackBite =
7  {
8      attackName : "Bite",
9      attackFrame: 1,
10     attackSprite: spr_enemy_bat_bite_attack,
11     attackDamage: 10,
12     attackDistance: 15,
13     attackSpeed: 5,
14     attackTriggerRange: 70,
15     attackBullet: noone
16 }
17 attackArray = [attackBite];
```

Figure 19: Enemy Attack Struct Example

Figure 19 showcases an example of the enemy attack struct. This struct is used for all enemy attacks in the game and each of these structs are loaded into an enemy specific array for random selection. To break down the struct, attackName says the name of the attack; attackFrame is which frame of the attack animation to launch the attack on; attackSprite is which sprite to use for the attack; attackDamage is how much the attack does when it hits the player; attackDistance is how far the attack goes in pixels; attackSpeed is how fast the attack should move; attackTriggerRange is the distance in pixels to launch the attack from; and attackBullet is which sprite to use for the bullet of the attack, if it has one. The attack is executed using this data and then bullets, sounds, or other behaviors may be created based on the current frame of animation the attack is in. Once the attack animation finishes the enemy transitions back to the FREE state.

The other major state for enemies is the DIZZY state. This state is reached anytime an enemy's HP is reduced and the HP reaches the enemy's specific dizzy threshold. Upon reaching that threshold the enemy enters the DIZZY state and turns red to signify the change to the player. From there a timer begins counting down until the enemy breaks out of the DIZZY state. While in the DIZZY state the enemy is unable to attack or move and is vulnerable to being grappled, thrown, or used to create cover; the player can continue shooting the enemy like normal though.

If a player hits the enemy with their grappling hook the enemy will be pulled towards to where the player is. If the enemy hits a wall in this process, they will explode dealing damage to all enemies around them. However, if they make it to the player, they will enter the GRABBED state. From there a player can swing the enemy body around themselves in a circle like a shield. The player can shoot through this shield whilst the shield absorbs all enemy bullets. The shield

itself is on a timer until the enemy breaks free; the shield can also be destroyed by absorbing enough enemy bullets. This is very beneficial for the player since it is a mobile piece of cover.

Once in the GRABBED state a player can choose to create a stationary, 360-degree cover with the enemy. This makes the enemy enter the COVER state, which functions similarly to the GRABBED state in that it has a timer until expiration and can only absorb a certain number of bullets before being destroyed. The key differences are that the cover cannot be moved around, and the enemy enters the DEAD state afterwards no matter what. This is useful for more hectic situations where multiple enemies fire bullets from different directions.

An enemy can also be thrown by the player while an enemy is in the GRABBED state. This causes a transition to the THROWN state which will move the enemy in whatever direction the player is aiming. An enemy can do nothing once this happens and immediately explodes upon contact with a wall, dealing damage to all enemies in an area; a player can make the explosion do more damage by shooting the enemy while it is flying towards its destination. Once this explosion occurs, the enemy immediately enters the DEAD state.

3.7 Bosses

Bosses in the game function slightly differently than regular enemies. They have an additional state called TELEPORT, which is the main way bosses move around the combat area. Some bosses instantly travel to the intended position while others will slowly travel to the intended position. A boss enters the TELEPORT state after executing a certain number of attacks, then exits the state once the teleport finishes.

Bosses also have significantly more attacks available to them and gain even more attacks after hitting a certain health threshold. Not only do they gain more attacks but the number of

attacks before teleporting can change and the time before launching another attack can decrease. Defeating the boss is also the only way to progress to the next level. Once this is done the player is given full health and a score bonus.

3.8 Inheritance

The other major programming concept implemented is inheritance. There are multiple entities in the game which make use of it, which helps speed development time up. The main target for this was enemy types. I have multiple different types of enemies with their own AI, attacks, movement, etc. but I needed to ensure that I had a baseline set of behaviors and reactions for all of them. So, each enemy in the game has a parent/child relationship with the parent class par-shootable.

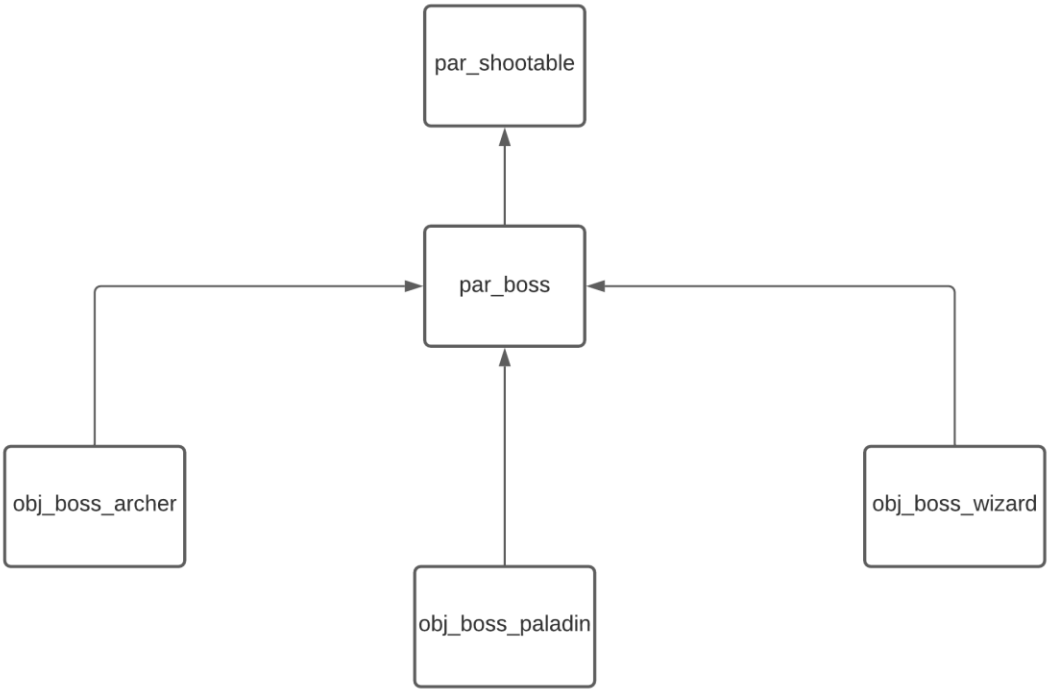

Figure 20: Parent/Child Relationships for par_boss

For example, this parent class handles things like collisions with player bullets, setting common variables, drawing itself to the screen, and more. All child classes can then react the same way to various game situations but implement their own variables or behaviors on top of the parent class. Enemy bullets, player bullets, player guns, and boss entities also take advantage of this parent/child system. Figure 20 shows off the parent/child relationship with par-shootable and par-boss; it also includes all the children for par-boss.

3.9 Menu Systems

Figure 21: Title Screen Menu

For major general game systems, we have the menu system and it functions by creating menu items based on the current context. If the player is at the title screen the start menu is created, which is shown in Figure 21. From there multiple clickable buttons are created which match the menu options. Clicking one of these either does a certain function (like starting the game) or advancing to another menu like the settings menu. The settings menu shown in Figure

22 is useful for adjusting the game's volume and for checking controls. The player can also bring up the pause menu at any time by simply pausing the game; this menu is shown in Figure 23. All movement, animations, and game logic stop while the game is paused. The pause menu is also useful for adjusting settings in-game or quitting back to the main menu.

Figure 22: Settings Menu

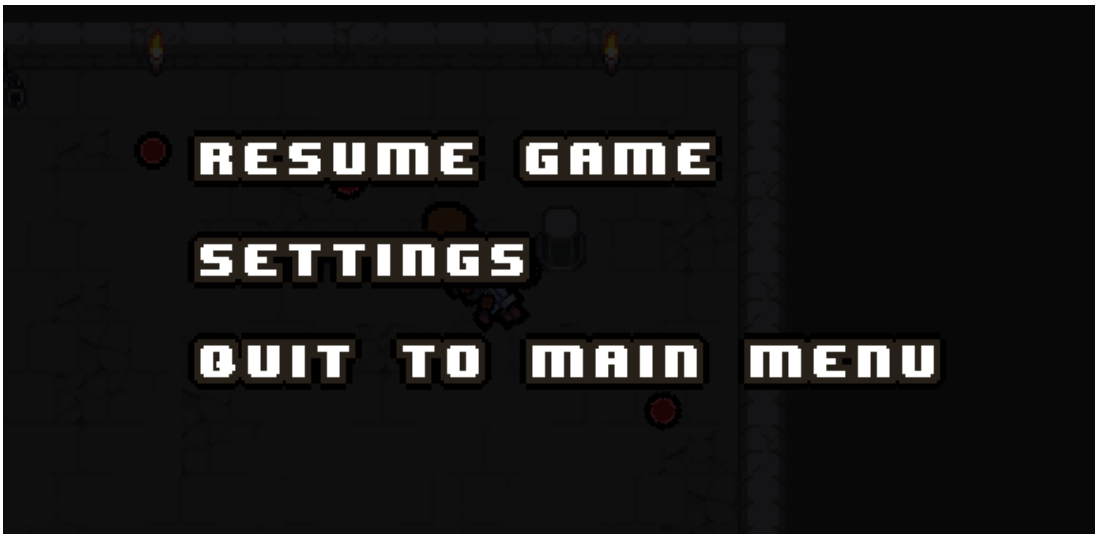

Figure 23: Pause Menu

3.10 Tutorial

An interactive tutorial is included with the game, and it can be accessed from the title screen. The tutorial itself covers all game mechanics and describes the ins and outs of each of them. It also covers all the different controls found within the game; what the current key binding is for them and what each does. This was mainly designed to let the player try out each of these mechanics for themselves in a no pressure environment. The tutorial delivers most of its information through the game's textbox system.

3.11 Dialogue System

Figure 24: Dialogue Box Example

All dialogue in the game is handled via a series of textboxes. These textboxes are created from a square-shaped sprite which has been sectioned off to 9 equal parts. These parts are used to build a textbox which can be any size; the 8 outer squares are used to build an outline and then the center square fills everything in. Once that is finished a portrait of the person talking is displayed near the top left of the textbox. Then text appears in a side scrolling fashion until the

dialogue for the current textbox is completely written, Figure 24 shows off an example of a completed textbox. While a textbox is on screen, all player actions are blocked.

These textboxes are mostly trigged by the player colliding with the object called obj-textbox-trigger. These can store a series of textboxes which are displayed one right after another. This is done via a ticket system, where all textboxes are assigned a ticket number. If the number is 0 then that textbox is displayed, and then once that textbox is destroyed all other textboxes tick down by 1. Once all textboxes are displayed the player's actions are no longer blocked.

3.12 Particle System

Figure 25: Particle System

I extended the particle system within GMS2 and made it more dynamic. So, whenever an enemy is hit by a player's bullet a small "blood" splatter occurs. This is handled by the game's dynamic particle system. It takes into account the bullet's speed, the angle it came from, the size of the bullet, and the main colors of the enemy it hits. From there the particles will fly out the other side at different sizes, speed, and color. This was added to give the player further feedback when their bullets hit an enemy. The enemy sprite also flashes white when they get hit by a bullet. This effect was done via a shader which just "paints" the sprite a transparent white which quickly fades with time. The particle and shader effect can be seen in Figure 25.

3.13 Spawn Triggers

The game itself is divided into levels and each level has multiple combat areas. These combat areas are triggered by the player colliding with an enemy spawner object called cont-spawner-enemy. This object will then lock all doors relevant to the combat area, locking the player inside until the area is completed. Each spawner has a preset number of waves, and each wave must have every enemy defeated before the next will spawn in. Once all enemies and waves are defeated, the doors unlock, and the player is given a score bonus based on fast they completed the combat area.

3.14 Story

The story in the game was inspired by the movie John Wick. Except here the villains steal your pet. A mysterious portal in your back yard opens, they take your pet, and you must go on a quest to get them back. The culprits responsible for this are representative of what a normal “heroic” role-playing game (RPG) party looks like. So, you have a knight, archer, wizard, and paladin. This was not only a fun way to flip a classic hero look on its head, but it also helped fuel the design of everything else in the game. Everything from art, enemy types, music, and sounds were chosen or made with this theme in mind.

Chapter 4 – Evaluation

4.1 Debugging

During the process of debugging and testing the game, I first started with unit testing. This involved creating a special room in which I could spawn in any enemy I wanted. Then I would test all facets of their AI like attacking, movement, etc. In this room I would also ensure all player systems were working correctly. I tested out every key binding and ensured none violated the state machine diagram I created. A small sample of unit tests I performed is shown in Table 1. GMS2's built-in debugger was great for this, as it allowed me to validate memory contents and I could set break points. I also did fairly extensive logging for error tracking.

Table 1: Sample of Unit Tests

Test Case ID	Test Case Objective	Steps	Setup	Expected Results
01	Test chain multiplier increment	1. Set chain multiplier to base X1 2. Shoot enemy until it dies	Obj-plant enemy in test room	Chain multiplier goes from X1 to X2
02	Test player receiving score bonus	1. Set score to 0 2. Shoot enemy until it dies	Obj-plant enemy in test room	Player score displays 400
03	Test chain multiplier being applied to score	1. Set base chain multiplier to X2 2. Shoot enemy until it dies	Obj-plant enemy in test room	Player score displays 800

04	Test cover being created from dizzy state	1. Shoot enemy until dizzy 2. Slide into enemy	Obj-plant enemy in test room	Cover created
05	Test grenade destroying bullets	1. Let enemy shoot bullets 2. Throw grenade at bullets	Obj-plant enemy in test room	Bullets destroyed
06	Test graze meter decrement	1. Set graze meter to 100 2. Use a graze shot	Test room	Graze meter displays 75 (3 chunks)
07	Test HP bar decrement	1. Set HP to 100 2. Get hit by enemy bullet	Obj-plant enemy in test room	HP bar displays 95
08	Test grappling hook	1. Shoot enemy until dizzy 2. Hit enemy with grappling hook	Obj-plant enemy in test room	Enemy is pulled towards player
09	Test melee throw	1. Shoot enemy until dizzy 2. Hit enemy with a melee attack	Obj-plant enemy in test room	Enemy is launched in melee direction
10	Test player death	1. Get hit by enemy bullets	Obj-plant in test room	Player dies at 0 HP

Once all systems were confirmed to be working within design specifications. I moved onto integration testing, which entailed putting multiple different enemies throughout a level and checking to see if this broke anything in the game. From there I moved onto alpha testing which mainly included running through the entire game multiple times looking for any oddities or failures. Then finally, I ran performance testing to ensure that I was hitting my target framerate of 60 frames-per-second (FPS). This test would verify that the frame rate never drops below 60 FPS during a level.

4.2 Performance Testing

Since user testing was not available due to COVID, my next best metric was how the game ran on my personal system. It ran on a PC with an i9-9900k CPU, 3080 RTX graphics card, 1TB NVMe SSD, 32GB of DDR4 RAM, and had Windows 10 as the operating system. The performance metrics used during testing was memory used and the FPS. The test itself was ran from the launch of the game until the credits rolled. I used GameMaker Studio 2's built in debugger to capture these performance metrics [1].

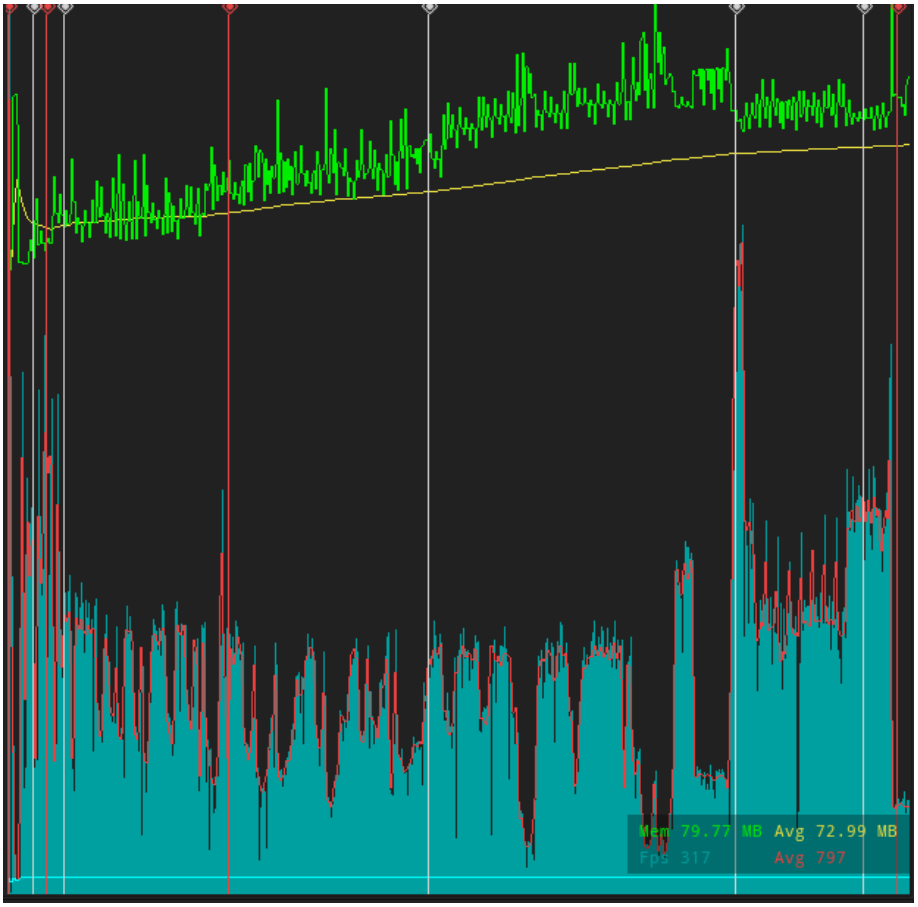

Figure 26: Performance Run-through Results

Figure 26 shows the performance graph for a full run through of the game. Average memory usage was 72.99 MB, which is fairly low compared to most modern games. For FPS the average was 797 FPS, with the lowest recorded FPS being 48 FPS which occurs when the game is started from the title screen. Otherwise, the biggest FPS drops occur in the game's most taxing situation, boss fights. These hover around 120-200 FPS. While normal enemy combat is in the 400-500 FPS range.

These are very good results as it means my solution should easily scale down to less powerful machines and the gameplay feel will never compromised in stressful situations by frame drops. Since GMS2 has neither multicore nor multithreaded capabilities, performance

should not greatly differ with weaker CPUs; in fact, older CPUs may perform better due to their higher single core clock speeds. It also needs very little RAM to run, so 1GB should be sufficient. Slightly more testing is necessary but as long as the graphics card is not an integrated type, any graphics card should do.

4.3 Project Metrics

Some of the metrics for the game is that it has 248 sprites, 99 sounds, and 12 music tracks. There are also 113 objects/entities and 58 scripts. I would estimate that in total there is well over 10,000 lines of code, as just the enemy AI alone is about 2,500. I would also estimate that I have spent well over 600 hours developing this game. Which includes everything from initially designing it, gathering art, sounds, music, and programming it

4.4 Project Difficulties

There were a couple of difficult problems I encountered while developing the game. One was a bug with the collision system and the other was getting the grab mechanic to work correctly. The bug in the collision system was sprites getting stuck in the wall if they were big enough. This was caused mostly when the sprite would collide with a wall and then flip along the y-axis to face the player. I solved this by having a separate object which would handle collisions with walls. So, each enemy in the game has a 16x16 collision box following them at the sprite's origin, and this is used for wall collisions instead of the entire sprite. This completely fixed that bug and made designing movement easier.

The last major hurdle was getting the grabbing and throwing of enemies functioning correctly. Originally, I just offset the enemy sprite's origin and rotated the sprite around the

player. This worked until I implemented throwing. Since I was just changing the sprite's angle, the actual x and y position of the sprite did not change. So, when I threw an enemy and reset the sprite's origin, it would result in a juddery movement as the x and y coordinates did match up to where the sprite angle made the sprite appear to be. To fix this I implemented a trigonometry function which would give me the x and y points on a circle when given the angle and radius. Then I would move the sprite to those coordinates, set the sprite angle accordingly, and then move it around the player in a circle. Throwing was now one smooth movement and I no longer had to worry about changing the sprite's origin back and forth.

Chapter 5 - Conclusion and Future Work

5.1 Conclusion

There are two SHMUP genres which are often not considered for combination, Twin-Stick and Bullet-Hells. Zenith of Lead was created to mash those two genres together to create a unique take on SHMUPs. With the goal being to present a player with a new concept to keep them engaged. This foundation will hopefully be used in future games (by other people or me), to further innovate upon this blend. For further improvements I would like to make myself, I did have a couple of things which were either in planning or had to be cut for time.

5.2 Future Work - Leaderboards

Although this game is complete in the sense that it has a beginning and ending. There are still several features and content I did not have time to implement. If given more time I would have wanted to implement some kind of local leaderboard system. Which would store your top 20 high scores of all time. This would also require implementing a way for the user to enter a name in an intuitive way (e.g. not just a textbox pop up).

5.3 Save and Quit

I also would like to finish implementing my save and quit feature. Which would allow the player to stop playing but still keep their progress in the game saved. While the bulk of the work is done (saving and loading a file), I still need to integrate it into the game itself. Which would take a decent amount of time just due to the nature of how buggy saving and loading can be.

5.4 Additional Level

For content, I had plans to include one more level and boss in the game. However, due to growing time constraints this level had to be cut. If I had more time, I would implement that level. I also wanted to include a few more enemies, but this would have required hiring an artist to mimic the style of the assets I am using. Budget constraints led to that not being possible at the time, but something I would investigate for the future.

5.5 Hard Mode

There were also plans to implement a “Hard Mode”, which would make all enemies and bosses in the game more difficult to defeat. This would have been done by making the bullet patterns harder to evade and/or changing up movement for an enemy. The intention would be to have this Hard Mode be unlocked after beating the game once. As this allows for the player to get acclimated with the game’s systems and how some enemies function. Then if they choose, they can replay the game on Hard Mode and enjoy a brand-new challenge. However, implementing this would have required a significant amount of time and I felt like the normal mode would be sufficient to showcase the game.

5.6 Porting

Something I had always planned to do after I finished the game was porting to other platforms. With the focus of that effort getting an HTML5 version running. Currently the game was developed and released as a Windows executable, but even with Windows being the largest operating system in the world we still limit our reach by only releasing on it. Also, not everyone wants to download and run random executables, and not everyone even has Windows to begin

with. A web version of your game allows you to reach the greatest amount of people with the least amount of commitment from their end. It is also a great way to show off your work to employers.

5.7 Playtesting

One of the biggest things I did not get a chance to do with the game was play testing. I could only do numerical adjustments (enemy health, damage, etc.) with data gathered from myself. This allowed the game to be toned down or up in sections I thought were lacking. However, I am quite familiar with the SHMUP genre. So, this data is only useful for the higher end of player skill. I need more data on medium player skill and those who are newish to the genre. The reason as to why I could not gather this data was simply because of COVID-19. I originally planned to have some in-person testing on campus, setting up a kind of booth for people to try it out. Since I was not comfortable sending out builds to people I did not know over the internet, I was stuck with myself for feedback.

5.8 Accessibility Features

The last big thing I would like to work on in the future is adding some Accessibility features. While I did design the game to avoid some accessibility pitfalls like flashing lights or low contrast areas. There are still more features I want to add that bring it up to the basic standard of accessibility. These would include things like being able to control text size, color blind filters, and customizable controls. While some work on these is already done, I would need a large chunk of time to get the rest finished. All of these improvements would greatly improve the game and lead it to being the best version of itself.

5.9 Things I Learned

I learned many things developing this game and further improved the skills I had. One of the most critical things I learned was the importance of keeping the game design document updated. There were a few situations where code rot could have been combated with an updated game design document. I also learned how to create a game design document; how to schedule small tasks which lead into large task completions; how to handle a medium-scale software project; and how to balance audio. One of the benefits I wasn’t expecting was that I improved my art and audio creation skills. So, this was a very fruitful endeavor for me.

Chapter 6 - References

[1] YoYo Games, *GameMaker Studio 2 User Manual*, Opera Software, 2017. Available: https://manual.yoyogames.com/.

[2] B. Wirtz, "All-Time Favorites: Shoot-em-up Games (Guaranteed To Give You An Adrenaline Rush!)", *gamedesigning.org*, Jun. 25, 2021. [Online]. Available: https://www.gamedesigning.org/gaming/shoot-em-up/. [Accessed Jun. 29, 2021].

[3] F. Rojas, "What is a Shmup?", *gaminghistory101.com*, Feb. 29, 2012. [Online]. Available: https://gaminghistory101.com/2012/02/29/shmup/. [Accessed Jun. 28, 2021].

[4] Galaga. [Arcade]. United States: Namco, 1981.

[5] Xevious. [Arcade]. Japan: Namco, 1982.

[6] R-Type. [Arcade]. Japan: Irem, 1987.

[7] Rez. [Dreamcast]. United States: United Game Artists, 2001.

[8] Asteroids. [Arcade]. United States: Atari, Inc, 1979.

[9] Nuclear Throne. [PC]. United States: Vlambeer, 2015.

[10] Mushihimesama. [Arcade]. Japan: Cave, 2004.

[11] Ketsui: Deathtiny. [Arcade]. Japan: Cave, 2003.

[12] M. Townley, "The iconic GPS triangle actually comes from the retro classic Asteroids", *knowtechie.com*, Feb. 11, 2011. [Online]. Available:

https://knowtechie.com/the-iconic-gps-triangle-actually-comes-from-the-retro-classic-asteroids/. [Accessed Jun. 29, 2021].

[13] Wikipedia. “Galaga”, *Wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Galaga. [Accessed Jun. 29, 2021].

[14] C. Savorelli, “Xevious”, *hardcoregaming101.net.* May. 06, 2015. [Online]. Available: http://www.hardcoregaming101.net/xevious/. [Accessed Jun. 29, 2021].

[15] R. Lambie, “R-Type: Examining the Legacy & Influence of the Space Shooter”, *denofgeek.com*, Dec. 27, 2017. [Online]. Available: https://www.denofgeek.com/games/r-type-examining-the-legacy-influence-of-the-space-shooter/. [Accessed Jun. 29, 2021].

[16] “Iconic Video Game ‘Rez’ Gets a Rerelease and Vinyl Soundtrack”, *insomniac.com*, Aug. 25, 2016. [Online]. Available: https://www.insomniac.com/news/iconic-video-game-rez-gets-a-rerelease-and-vinyl-soundtrack. [Accessed Jun. 29, 2021].

[17] “Mushihimesama”, *steampowered.com*, Nov. 12, 2015. [Online]. Available: https://store.steampowered.com/app/377860/Mushihimesama/. [Accessed Jun. 29, 2021].

[18] “Ketsui: Deathtiny”, *m2stg.com*, Nov. 11, 2020. [Online]. Available: https://m2stg.com/en/ketsui/. [Accessed Jun. 29, 2021].

[19] M. Cox, “Have You Played... Nuclear Throne?”, *rockpapershotgun.com*, Nov. 10, 2017. [Online]. Available: https://www.rockpapershotgun.com/have-you-played-nuclear-throne. [Accessed Jun. 29, 2021].

[20] “Superpowers”, *superpowers-html5.com*, Jan. 06, 2016. [Online]. Available: http://superpowers-html5.com/index.en.html. [Accessed Jan. 25, 2019].