**CA 2 – Design Document**

***Luke Courtney***

**Part 1 – Overview**

As part of CA 2 in Gameplay Programming, we were tasked with taking a pre-existing game and improving it. The game we were given is a top-down shooter where the player is required to fight off hordes of zombies, while unlocking upgrades between each wave.

I made three main changes to the game. I added a new explosive zombie type, a new way to collect ammunition, and a grenade weapon. These three changes allow the player to be more aggressive in their gameplay, as both the grenade and explosive zombie can eliminate several enemies at once, and the ability to collect more ammunition when needed allows the player to be less conservative with their usage.

**Part 2 – Coding and Implementation**

The first new feature that I added was the ability to trade health for ammunition whenever needed. This allows the player to take a risk, and lose health, for five bullets in return. This feature was inspired by Bloodborne, a game released in 2015 by FromSoftware. In the game, players can also sacrifice health in exchange for bullets, but in Bloodborne, bullets are used for parrying enemy attacks, and are more defensive as a result, in contrast to Zombie, where bullets are the players primary means of offense. Trading some health for ammunition in Zombie is as simple as pressing the F key. When this happens, the player loses a portion of their health, and is awarded five bullets. This action can be performed as long as the player has enough health remaining. If they do not have enough health left, nothing will happen. This is to prevent the player from accidentally dying as a result of trying to gain ammunition without having enough health.

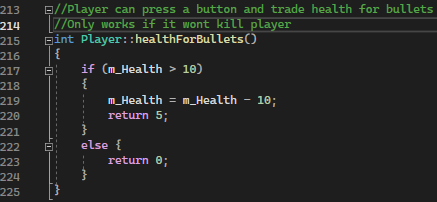


Figure 1 - Code for Health-For-Ammo action

The function *“healthForBullets”* in Player.cpp shown above subtracts ten health from the player, only if they have more than ten health remaining *(Lines 217 to 221),* Otherwise, no health is taken. The function then returns the an integer value *(Lines 220 & 223)* based on whether health has been taken. If health was taken successfully, a value of 5 is returned, otherwise a 0 is returned. This value is taken in ZombieArena.cpp and is added to the amount of ammunition the player has. In this case, when a value of 5 is returned, the player gains 5 bullets.

 A picture containing light

Description automatically generated

Figure 2 - Exploder Sprite Figure 3 – Explosion Sprite

The feature that I added was a new zombie type. This zombie, shown above, is an explosive enemy that will explode and damage nearby zombies if killed. Similarly, if the player gets too close, the zombie will explode and deal a significant amount of damage. When the zombie explodes, it is also killed, and cannot deal any more damage as a result. When this zombie dies, it leaves an explosion sprite, as opposed to the blood splatter sprite left by regular zombies.

Text

Description automatically generated

Figure 4 - Zombie explosion code

Whenever a zombie is hit, its health is checked to see if it was killed. When this happens we check if the zombie was an exploder *(Line 655)* . If not, we just play the regular splat sound effect that plays whenever a zombie dies *(Line 678).* If it *is* an exploder, we loop through all of the zombies on the map, and if they are close enough to the exploder, we hit them, and play the appropriate sound effect *(Lines 657 – 673)*. A similar sequence of events occurs if the player gets too close to an exploder. If they get hit by the exploder, the zombie detonates, damaging nearby zombies and dealing a large amount of damage to the player.

The next feature I added was the largest in terms of amount of code required. It give the player the ability to throw grenades by right-clicking. Grenades are obtained through ammo pickups, just like bullets, but are much fewer in quantity. Each pickup only grants the player one grenade. The grenade system works similarly to the bullets, with modifications to speed and max range. The grenades travel much slower, and have a much shorter range. When a grenade is thrown and reaches its max range, it explodes damaging any nearby zombies.



Figure 5 - Grenade states

To control whether the grenade is intact, should explode, or has already exploded, I used an ENUM called state. When the grenade is first thrown it is considered *“Intact”*. When it reaches max range and should explode, it is set to *“Exploding”*, and finally when it has exploded, it is set to *“Exploded”*. This system stops the grenades from exploding more than once and dealing too much damage as a result. Grenade.cpp contains several functions related to this, such as “*getState*()”, which returns an integer value according to the current state of the grenade, and “*exploded*()”, which sets the state to exploded once the grenade has detonated.

Text

Description automatically generated

Figure 3 - Grenade code

The code for the grenade is similar to that of the exploder zombie. We first update the grenades position by looping through all active grenades and calling the update function *(Lines 600 – 606).* Next we check if the grenade should explode, by checking what state it is in. If the grenade has reached its max range, it will have been set to the *“Exploding”* state, and the code will continue *(Line 609)*. From here, we loop through all the zombies on the map, and damage those that are close to the grenade *(Lines 611 – 628)*. We then set the grenades state to *“Exploded”* so that it cannot be detonated again *(Line 631)*.

**Conclusion**

As part of this CA I modified and improved the Zombie game by adding three new features; A new zombie type, a new way to obtain ammunition, and a new weapon. These features all allow the player to be more aggressive with their gameplay and take more risks.