Summary

Recoil is a short action-platformer game using shotgun recoil physics to traverse the levels and kill enemies. The player character is based off of a country type individual who mistrusts machinery, which is why the main enemies are robots. The player moves their mouse along the screen, shooting using the left click and being blown in the opposite direction of where the mouse is pointing. There are 3 levels, and the game is (lore-wise) set in a large, science fiction tower. The player can win the game by beating all three levels, but can lose health or die to kill blocks that will take him back to the last checkpoint. There are three different weapons as well; the default shotgun, a grenade launcher, and a pump action (that works like the default shotgun but has more ammo). There is also a tutorial level that quickly illustrates what the gameplay is about.

Recoil turned out almost exactly as created in the document! The only main updates and differences and the ways that the shotgun is used as simply climbing can become a bit boring and lame if that is the only objective. So to compensate for this the game sort of morphed into a slightly puzzle like platformer asking the player to find clever solutions to platforming situations with the systems provided to them.

Tickets

- #1 Project report 1
    - Justin Labrie and Jakob Kirby
    - Assigned in 1st iteration
    - Completed
- #2 Shotgun 1 Mechanics
    - Jakob Kirby
    - Assigned in 1st iteration
    - Completed by Jakob Kirby
- #3 Main Menu
    - Justin Labrie
    - Assigned in 1st iteration
    - Completed by Justin Labrie
- #4 Tutorial Level
    - Justin Labrie and Jakob Kirby
    - Assigned in 1st iteration
    - Completed By Justin Labrie
- #5 Create Basic Enemy
    - Justin Labrie and Jakob Kirby
    - Assigned in 1st iteration
    - Completed by Justin Labrie
- #6 Create Shotgun Pickups
    - Jakob Kirby
    - Assigned 1st iteration

    - Completed by Jakob Kirby
- #7 Create different walls and platforms types
    - Jakob Kirby and Justin Labrie
    - Assigned 2nd iteration
    - Completed by Jakob Kirby
- #8 Make some sprites and animations
    - Jakob Kirby and Justin Labrie
    - Assigned 3rd iteration
    - Completed by Jakeb Kirby & Justin Labrie
- #9 Make checkpoint and killblocks
    - Jakob Kirby
    - Assigned 4th iteration
    - Completed by Jakob Kirby
- #10 Complete Enemy attack script
    - Justin Labrie
    - Assigned 4th iteration
    - Completed by Justin Labrie
- #11 Complete player health script
    - Justin Labrie
    - Assigned 4th iteration
    - Completed by Justin Labrie
- #12 Enemy health script
    - Justin Labrie
    - Assigned 4th iteration
    - Completed by Justin Labrie
- #13 Make Level 1 & 3
    - Jakob Kirby
    - Assigned 4th iteration
    - Completed by Jakob Kirby

Github

https://github.com/jel2658/IGP_Project

Videos

Recoil video demonstration:
https://drive.google.com/file/d/1-gxUxsD5awbnILpa0-QYrYgCXYCzzk6y/view?usp=sharing

What We Learned

Main Menu
The UI design takes a little bit to get used to. Luckily with the Unity assignments we went over a few things that helped in manipulating the buttons that I created for the main menu. Scene

transitioning in Unity is very easy to script and very smooth, only needing to use a few lines in the scripts. However, I had to create a separate game object to hold the necessary scripts, as Unity would not allow me to directly attach transition functions to the buttons through script components.

Tutorial Level
The main issue was deciding what exactly the tutorial level should include. I elected for a very short one, just telling the player that they would need to shoot themselves into the enemy to kill it. I had a script attached to the text that appeared on the screen, which would check for when the player had shot their gun and update when they did, then would check for if the enemy was dead or not, and update one last time if it was. TextMeshPro editing in script was a little finicky, but I managed to fix the minor issues that occurred when I wanted to update the string that the text showed.

Basic Enemy
This was probably the most difficult of my tasks I needed to finish. Walking, health, damage, I decided to put in different scripts so that I wouldn't clutter one script with all of the code. Therefore it was later split into multiple other tasks down the line. This issue was considered finished after creating the animations and the walking script for the enemy.

Most of this was me figuring out a way for the enemy to find the player's position and move towards it without rising up into the air. What I managed to find was publicly assigning the player in Unity to each enemy and finding its transform, then have it move towards that position. I also figured out that I could use the enemy's own y position on its transform in order to keep it from going after the player when he was in the air, and instead just continue moving towards his x position.

The animations were fairly easy, I just had to grab the several sprites for the enemy, put them into a clip, and put together an animation controller, which was all covered in class. This ties into the issue about making sprites and animations, though I had already bought the spritesheet.

Complete Enemy Attack Script
This one was only mildly difficult because the player health script had to interact with it. I put a collider on the enemy, so that when it was touched by the player it would run the player's "take damage" function. What I did learn was that it was vitally necessary to have that take damage function as well as separate values for starting and current health of the player (as taught in class). This allowed the enemy attack script to fully work as intended.

Player health and Enemy health
Like I said about the enemy attack script, the enemies and player had to be able to interact for all of these components to work. Enemy health was ultimately decided to be a boolean value, and the player only needed to have a certain velocity when colliding with them to kill them. The player, on the other hand, had a health value starting at 100. I created the damage script but realized that this would effectively run immediately over and over again (as I also created a

damage function for when the player continued to collide with the enemy). I had to create a timer of “invincibility” for the player, so that they could move and get away from the enemy and not immediately die to them.

Level 2

A ticket was not created for level 2, though I should have created one when we decided that I would be working on it. I started the design by drawing a possibly level on paper, making a few adjustments with pencil and eraser until I thought it was good. I utilized sticky platforms created by Jake, but otherwise most of it was individually created by myself. I also put together shotgun shells to reload the player’s gun, for a semi-difficult portion where they had to traverse up a long upwards hallway. I realized that the enemies would immediately start going after the player, so I elected to create ranges for each enemy and require that the player had gotten into their range before they started moving toward them. This method I implemented very lazily- I created empty game objects to mark the x-coordinates of the ranges of each enemy, and publicly added them to the enemy’s new ‘range’ script in the Unity editor. Once the player’s x position was between the enemy and the range object, their walking scripts would activate (their default being inactive).

Shotgun Mechanics

The main shotgun was what I am now dubbing my trial by fire. It was my first “real” game programming task. It was really the first game or programming objective I had no reference for other than having to chart out what I wanted to happen as well as break it down. It basically led me to finding an actual work flow in programming that I was able to transfer to the other tags and programming tasks.

For this one thing 1 tag I have pages of different theories and math that might work, that all just culminated in me finally realizing that I am not just using code but Unity as a whole and it required a clever solution.

After that was the main form of reloading which was quite the pain. The little pop that the player does was something that I was dead set on and had to have in the game as I felt it not only had a gameplay role of forcing a stop in momentum but allowed for some expression with the main character and game that the little pop flip and reload had. The player physics and the eventual rabbit hole of small but important unity things such as learning quaternions and euler angles, to all the different parts of the rigid body and how they actually impact player movement. This was mainly evident in the early renditions of the shotgun and how the player would slide as I had NO CLUE what the factors governing the inertia and spin of the player was. The early iterations of Recoil also didn’t even have right left movement, it was just straight up the shotgun. I had to create all the conditions for when you can walk and when I believe it is acceptable, AND EVEN THEN it's still a little jank.

Now looking back at the shotgun code the main struggles that I had was simply me NEEDING to get the exact idea I had in my head to be on the screen. Would my life have been easier if the player would just launch to its location? Yes, but the fact that he spins based on the location the

player is pointing at, as well as his current spin all being added in as factors making it feel like the shotgun is real and has power makes me feel really happy.

Shotgun Pickups

These were fairly easy for the pickup side, them being just triggers but what I wasn't ready for was the intro to the cascading programming that I was in for. With the addition of these shotgun pickups I had to go back through and reanalyze the shotgun code. Like- "Oh god I not only have to go through every situation that a pickup might be affected, but what features are going to be affected by the pickup existing….. Cool." This was the main trouble encountered with the pickups. The reload conditions were changed, an entirely different firing script was created just for the grenade launcher being shot as the normal shooting code could not account for the DRAMATIC increase of power a grenade provides.

Platform types

This is now just part two of dear god I have to redo everything I ever thought and believed. The different platforms present in the game were not an idea at the very beginning of Recoil but just an eventual expansion of the idea of a shotgun platformer. The sticky platform was probably the hardest to create as stopping all momentum of the player and making it so that the player can't just get soft locked for wanting the hopes of sticking to a wall was a little funny in bug fixing, but a bit annoying in the code side. I learned that sometimes I tunnel so hard on a solution of forget that, stupid, you already created the solution in the bounce platform, but I was so against the idea of using a hitbox as a trigger instead of struggling with OnContact that it took a good half hour of garbage code to just do the obviously more functional and slightly ugly solution.

Player animation and shotgun sprites

Now although I do have some artistic experience I have never really done animation or drawing for games. As I said before though I am an absolute control freak and if I can't get the exact idea I have put onto screen it will keep me up at night. But what I did learn from this is what is the minimum amount of effort and production needed to get a point across. The only animation that I created was for the walk cycle and it is 4 frames not including the idle. That was all that was needed to get the point across. Now of course I obsessed over those 4 frames, as they are now my children and I know will grow up into a beautiful walking animation someday. But I tried to keep the animation to a reasonable level of work, and expectations of someone who is doing character animation for the first time ever.

Checkpoint & Killblocks

The checkpoints and killblocks really taught me how to really make clever solutions in my code. Now the checkpoints and kill blocks are very self explanatory but I needed to take what I know of checkpoints and killblocks and actually make them. I then sat down much before actually writing the code, and wrote everything that had to happen along with the relationships between the checkpoints and kill blocks. Writing out this relationship helped me come up with the solution

of having every checkpoint have an array of every checkpoint and deactivating all then activating themselves when the player entered. This would leave the kill block to have to just teleport the player to the only activated checkpoint in the level making the code much simpler to write.

Level 1

I truly believe that making the levels was probably one of the hardest parts of making the game. As a person who struggles to make a name and intro paragraph for their essay that is what level 1 felt like. How do I introduce this fairly goofy way of movement to the player, and not only do that but introduce the fact that if the shotgun has ammo, your movement dreams are possible? Not only that but how do I introduce the platforms. The first answer would be the tutorial Justin made. But i know that there are stupid people like me who jump in a game skip the tutorial and choose the hardest difficulty. How do you teach me the way to play my own game? I mainly expand upon these ideas within the video.

Level 3

Level 3 was the opposite type of learning curve. Now it was what is the difference between challenging and just straight up unfair. This I think is what led me to create the slick platforms that are only in Level 3. I had a section in the level that I envisioned since the idea of the shotgun was first introduced and I needed to have it be in the game. The problem was when I was testing it, it didn't feel like I was messing up my movement, it felt like the platform placement was unfair. So the slick platforms were created to remedy this fact. It really just appears that you are given more space to land on but with the slide you really don't have a lot of space, BUT YOU DON'T THINK THAT! Missing a tiny platform feels unfair, seeing your own lack of planning make you slide off a platform makes you feel like you just have to come up with a different and clever solution, which is what I wanted out of this level. Now there are some middle finger parts in the level such as the pump action challenges, but I wanted the final level to be a test of skill, and some fun puzzles as well.

Misc.

These are really just some tasks that didn't have tickets as they were not main gameplay objectives but more small additions I did cause I am an absolute control freak and needed them to happen.

-Bounded camera- The bounded camera mainly taught me that you sometimes don't need the perfect version of something, and that if you have a time limit good is sometimes good enough
-**AudioManager-** This was gotten from a tutorial by Unity Pope, Brackeys, but I realized that it didn't meet exactly how I wanted, so I did have to add a lot of function to the code as well as arrays, to separate the different audio and sound effects. The result may be a bit jank but I just needed them to work
-The actual SFX and Music- The beautiful sound effects were not born out of spark of genius but a desire to never go and look for royalty free sound effects ever again. So I slapped on a mic, made a fool of myself and made some magic. The music was also another of my mental

demands. After Justin presented the general story for the game being sci-fi, I NEEDED to have synthwave. I don't know why, but I felt that it would fit perfectly in with the game. So after some digging while listening, I found White Bat audio, someone who created royalty free synthwave and is now my savior.
-Weapon sprites switching and directions- The weapons sprites were a bit annoying at the beginning with me having to figure out a solution to have the sprites point in the actual direction instead of the character having to pop their shoulder out of their socket when they aim. From this i learned that unity rotation is dumb, BUT that euler angles are switched from computer numbers into degrees and data I can actually work with and understand. This allowed me to create the code that switches between a sprite that points left and one that points right depending on the direction. Other than that the only other thing is some functions that change those sprites depending on pickups.

Conclusion

Given the fun nature of this game and the massive potential it has it wouldn't be a stretch to say there might be a *Recoil2.0* or at least a polished version of this game. One with all the quirks and small bugs ironed out of it. This game was extremely fun and gratifying to work on, being able to look back at the beginning of the year and see how far we have come programing and project wise.

Setup

The build for *Recoil* is all inside of the Recoil-build folder. All needed data is in that folder.
If running inside of unity play from the main menu, otherwise you should be fine.