Battle School Osu

Final Report

CS 467 Capstone Project

17 March 2019

Charles Jadin | Christian Roccanova | Nicolas Sim

Introduction

A semester in the making, team Navi proudly presents “Battle School Osu”, a zero-gravity first person shooter based on the *Ender’s Game* science fiction universe. The idea for the game came from our own Charles Jadin and was inspired by the battle room scenes from the book and movie. The game was created using the Unity engine due to its availability as well as the vast array of assets that could be used to make the game more aesthetically pleasing, allowing us to focus on the code. Despite the help that the assets provided, we did have our fair share of challenges as none of us had used Unity or the C# programming language on a project of this magnitude before. In particular, we had several issues regarding the physics engine as well as some initial problems with the integration of the disparate parts of the project.

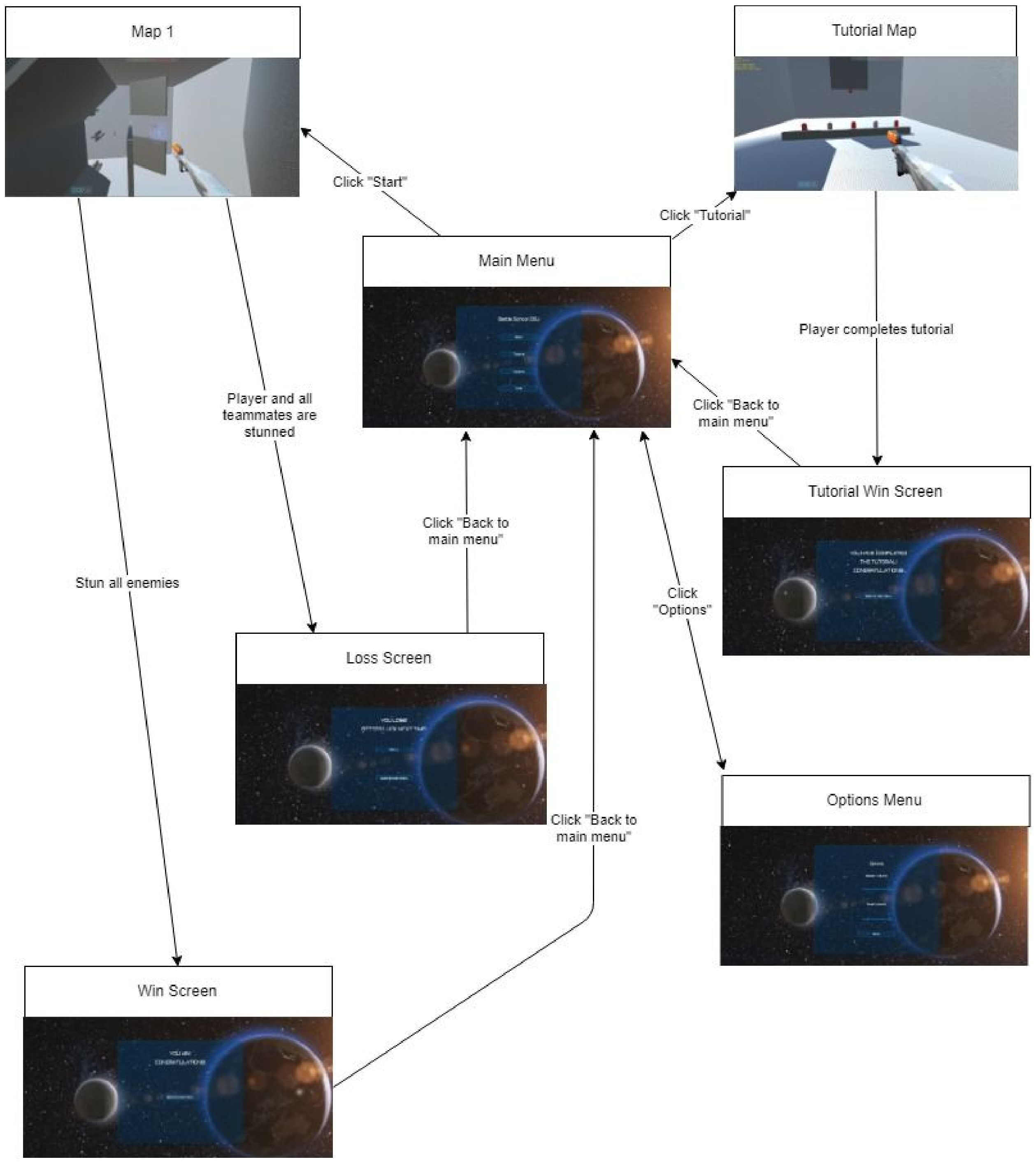
The basic premise of the game is simple enough; you are part of one team and you must find and shoot all members of the opposing team in order to win. What sets our game apart from others in the genre is the zero-gravity environment combined with the lack of any propulsion devices such as a jetpack. In order to navigate the level, players must launch themselves from one object to another and cling to these objects as they collide with them lest they bounce off and drift helplessly into the enemy’s sights.

Instructions

1. Find and run the executable file.
2. From the main menu select “Options” and alter the settings (volume, mouse sensitivity, etc.) as necessary.
3. Return to the main menu and select “Tutorial”, this will teach the basic mechanics of the game.
4. The first tutorial room will simply ask the player to navigate a corridor and find a teleporter which will appear as a green portal.

5. Upon entering the portal the player will advance to the next room which is a simple shooting gallery. Shooting all eleven targets will unlock another green portal, allowing the player to advance further.
6. This third and final room is a long corridor bristling with turrets. The player must get to the exit portal at the end of the room without getting hit to complete the tutorial. Getting hit will return the player to the entrance of the room.
7. If at any point in the tutorial the player becomes stuck due to a loss of momentum pressing the “Home” key will respawn the player at the most recent checkpoint.
8. Upon returning to the main menu, select “Play”. This will begin a standard match against the AI.
9. Upon spawning, the player will be considered “latched”, they may then launch and cling to the terrain as they see fit in their hunt for the enemy.
10. To win, the player and their AI teammates must stun all members of the enemy team. Stunned AI will cease moving and shooting and will be distinguishable by a visual effect.
11. If the player is stunned, they will replace one of the still functioning AI teammates if any remain.
12. Once either team is completely stunned, a victory or loss screen will appear as appropriate. Clicking the button on screen will then return the player to the main menu.

Graph 1 - Scene Progression



<u>Using Unity</u>

- Assets -
    - Sci Fi Warrior PBR HP - This asset features the animators, materials, prefabs, and textures used for our ally AI and enemy AI. A futuristic warrior with space-like armor equipped with a gun.
        - https://assetstore.unity.com/packages/3d/characters/humanoids/sci-fi-warrior-pbr-hp-106154
    - Let's Try Shooter - This asset features our first prototype first person shooter which allowed us to modify our shooting and hit detection for our game.
        - https://assetstore.unity.com/packages/essentials/tutorial-projects/let-s-try-assets-66207
    - 10 Texture Sets 'Industrial 02' - This set of textures was used to make the hexagonal patterns on the walls and terrain objects.
        - https://assetstore.unity.com/packages/2d/textures-materials/10-texture-sets-industrial-02-17624
    - Absolute Space & Sci-Fi Music - These audio tracks make up a large portion of the background music.
        - https://assetstore.unity.com/packages/audio/music/absolute-space-sci-fi-music-free-sample-103274
    - FREE Casual Game SFX Pack - These audio tracks were used for contextual audio cues such as the “thud” that occurs upon collision between the player and a terrain feature.
        - https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116

<u>Game Concepts</u>

- Controls
    - **Mouse** camera control

  - **[space]** launch (move)
  - **[f]** latch onto a surface
  - **[left mouse button]** fire stun gun
  - **[Home]** resets player at spawn point if they get stuck

- Artificial Intelligence (AI) - The actions of our computer-controlled characters, or “bots”, were determined largely by the BotFiring and BotMovement scripts, which often share information with one another regarding the acquisition and state of the bots’ targets. A bot’s movement begins with a check against a string value, which gives direction for the bot to assume a stationary post or patrol several points throughout the map. A patrolling bot may stay on its team’s side of the map or stray into the other side, based on a boolean for patrol aggressiveness. The movement script also uses raycasts to detect whether an enemy is in range, and linecasts to determine whether the enemy is visible (not blocked by other objects). The firing script, then, takes this information on the number of targets the bot has acquired, loops through them to determine proximity, and singles out the closest active target that is within the line of sight. The script then performs a calculation of the target’s velocity to identify where it will be when the projectile arrives, and the bot fires on that location (with a slight offset to make the bot’s aim imperfect).

- Animations - The animations were derived from the Sci Fi Warrior PBR HP asset package. In the asset folder, there were 10 animations that were pre-built in (Figure AN.1), but because of the nature of our game, we didn’t need that many. The characters are in a zero-gravity room, where you can either be alive or stunned, so we really only needed two animations - floating and shooting. We were given a preset animator which loops all animations as a showcase (Figure AN.2). I made an animator where it starts and loops in the Idle_Shoot animation and activates a boolean on a left click. (Figure AN.3 & Figure AN.4)

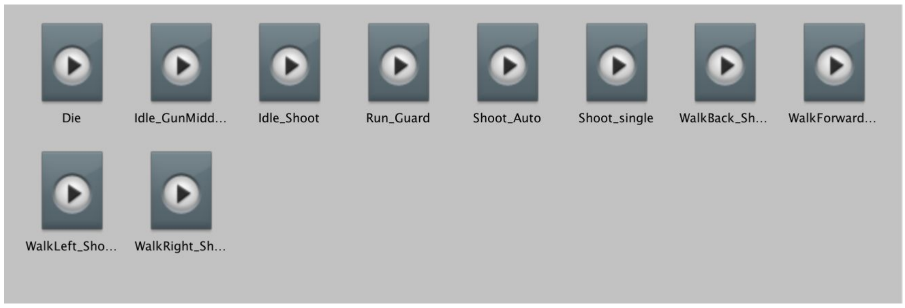
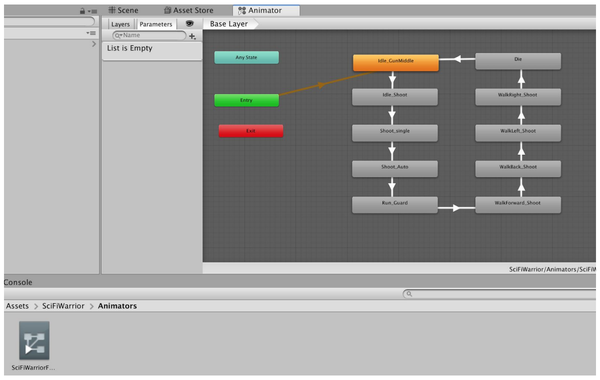


Figure AN.1



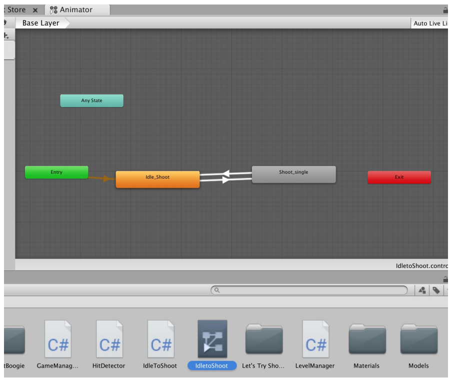
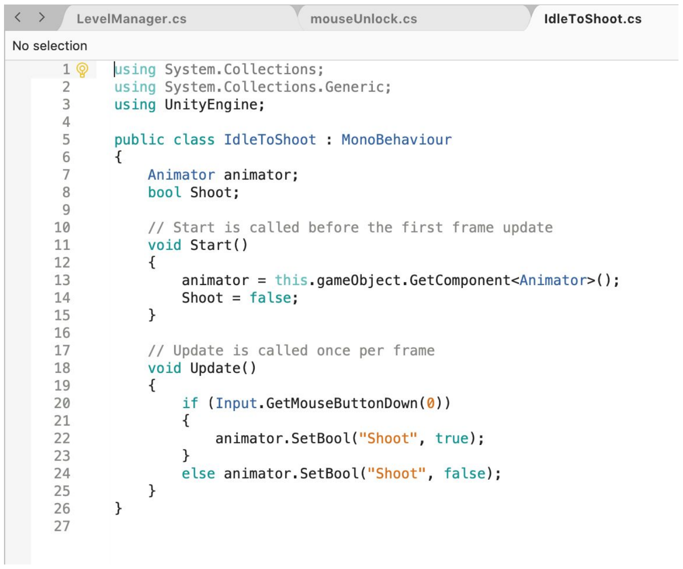


Figure AN.2

Figure AN.3



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class IdleToShoot : MonoBehaviour
{
    Animator animator;
    bool Shoot;

    // Start is called before the first frame update
    void Start()
    {
        animator = this.gameObject.GetComponent<Animator>();
        Shoot = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            animator.SetBool("Shoot", true);
        }
        else animator.SetBool("Shoot", false);
    }
}
```

Figure AN.4

- Shooting - this was handled by separate scripts for the human-controlled player and computer-controlled characters (“bots”). In the Update() function, called with every passing frame, the script checks to determine if the fire button is pressed by the player, or if the bot has a target in its line of sight. If the relevant condition is satisfied, there is another check to determine if enough time has passed since the last shot, or in other words, the “cooldown” has lapsed. This is a fixed amount of time for the player and a random number within a specified range each time for bots. The script then instantiates a pre-fabricated projectile object and begins its movement in the direction the shooter is aiming. This projectile is different for the human player, “Ally” bots, and “Enemy” bots, and each has its own script for hit detection, as explained below.

- Hit Detection - Hit detection is handled by separate scripts attached to different bullet objects based on the shooter. Regardless of type, each bullet fends out a raycast a short distance in front of it as a it travels. If these raycasts collide with an object it checks the object’s tag. Depending on the bullet type, code will then be executed based on the tag. For example, if a player’s bullet collides with an enemy, the hit detection script will access a script attached to the player and increase their score and will then access the behavior scripts of the enemy and disable them so that they cease movement and firing.

- Sprites - The Sprite that was used for our AI characters were derived from the Sci Fi Warrior PBR HP asset package. The Sprite was transformed to fit within the capsule which is linked to a capsule collider that deals with all the interactions such as wall clinging and bullet detection, but the position transform is 1:1:1 with the capsule shape.

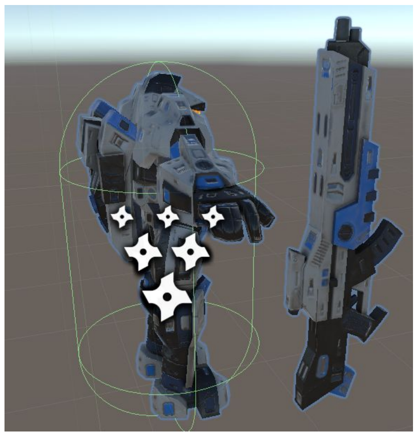
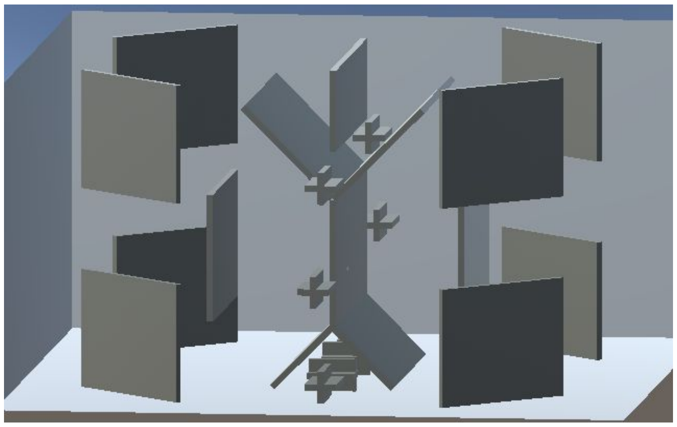


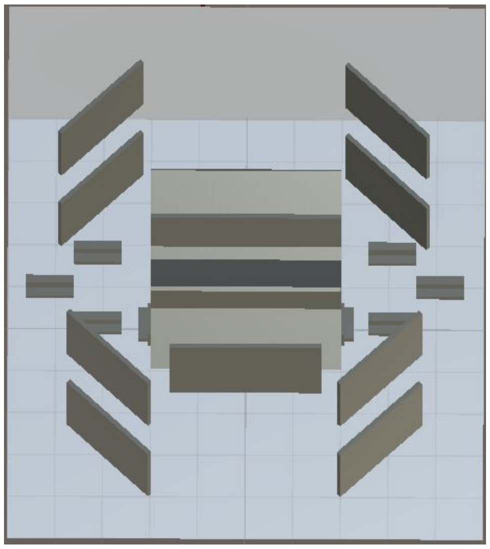
Figure SP.1

- Game Management - Game management is handled by a series of public variables spread across several scripts. There is an overall game management script that accesses these variables and uses them to control things like the scores on the HUD and when the game ends.

- Level Management - Level management is fairly straight forward and is handled by built in functions within Unity's SceneManager class. These functions load the appropriate scene based on a player's action such as clicking the button to start a match or entering the exit portal in the tutorial level.

- Map - The game's main map is an enclosed area with four walls and many large floating objects. These walls and fixtures require colliders and "edge" objects for bot players to recognize as points on their patrol routes or for defensive positioning. They also give the ability for the player to latch on to stop movement.


Side view of the main map


View from above the main map

Team Contributions

Charles Jadin

Charles was responsible for most of the AI scripting for movement and combat, as well as tweaking several environmental objects and character components in order to facilitate proper functioning of these scripts. In addition, he worked on the firing mechanics of human and computer characters and the player control shifting scheme upon becoming stunned, which required collaboration on the projectile objects and hit detection scripts to tie these mechanics together. He also designed the main menu, structured the game settings module, and created the aesthetics of the in-game UI.

Christian Roccanova

Christian’s primary role was level design and the implementation of physics. The majority of the maps and scripts involving map objects such as spawning characters and teleportation were part of his jurisdiction. He was also responsible for the majority of the audio and lighting effects found in these levels, as well as their triggers. As far as physics was concerned Christian designed the player movement and latching system used to navigate the terrain. He also created the hit detection scripts for the various projectiles in the game. He also had a hand in writing many of the scripts that handled game management including win and loss conditions, disabling characters when they were stunned and management of elements of the heads-up display such as the match timer and score trackers.

Nicolas Sim

Nicolas was responsible for the first person character features and animations. He also worked on the beginning stages of the shooting mechanics and character movement mechanics. He assembled the sprites and animations for the ally and enemy AIs. He also created the level management system, and corresponding menu screens with button functionality. In addition, tested for bugs and did small revision in scripts.

Deviations from the Plan

Over the course of the project, we did have some deviations from the initial plan. Originally, we intended for there to be a second map option for the full game, but this was cut due to time constraints and the need to implement more important code such as that needed to run the heads up display.

Some other deviations occurred as a result of testing gameplay. One example is the number of players on each team. In our initial proposal, we considered that approximately twenty per team would be a good number, however due to the complexity of movement and shooting in the zero-gravity environment, we scaled this down to give a less hectic feel. Another change was in the conservation of momentum. We originally wanted players to move until a collision occurred and then stop, however this gave movement a stiff feeling. In the end we decided to make players bounce off of objects, losing a set amount of their velocity for each collision. This allowed for more interesting movement as well as the ability to perform more complex maneuvers involving these bounces.

The team number change resulted in slight modifications in the movement behavior plan. There was a tentative vision for small subgroups in each team moving as a unit. This would have made sense if each group could have had four to five members, but with six total on each team it became impractical.

Another AI-related deviation involved the combat state of the bots and its effects on movement. Originally the plan was to have AI-controlled combatants check for targets and interrupt their initially-defined movements (patrolling or stationary) when they had enemies in line of sight. This proved problematic and disruptive, causing anomalies like bots floating towards the corner of the map and becoming suspended in midair spontaneously. We opted for resuming patrol or stationary posts, as this ended up flowing more naturally than anticipated (and much more so than the alternative).

Conclusion

For all of our team members, this was the first time working with Unity and C#, and there was a lot of learning and conceptualization before we could start creating. In the end, we were all able to learn the basics of building a first person shooter game with the help of online tutorials and some background research on the internet.

Getting started with Unity was easy to pick up due to the wide variety of free assets from the assets store. The popularity of shooter games meant that with a quick google search, we were able to find a plethora of resources to help us make the game function as smoothly as possible. Thanks to this abundance of resources and some good old-fashioned teamwork, we were able to complete our first game and come out of it learning so much from the experience.

In the end, we covered all the basic concepts that we wanted to implement, but given more time, we would’ve liked to expand in terms of content such as making additional levels with more dynamic features. In a way, having to settle for a more modest version of our grander design was a lesson in its own and gave us a greater appreciation for the amount of time, effort and passion that goes into projects such as these. Overall, we found the experience quite enjoyable and are proud of what we made and hope that you enjoy it too.