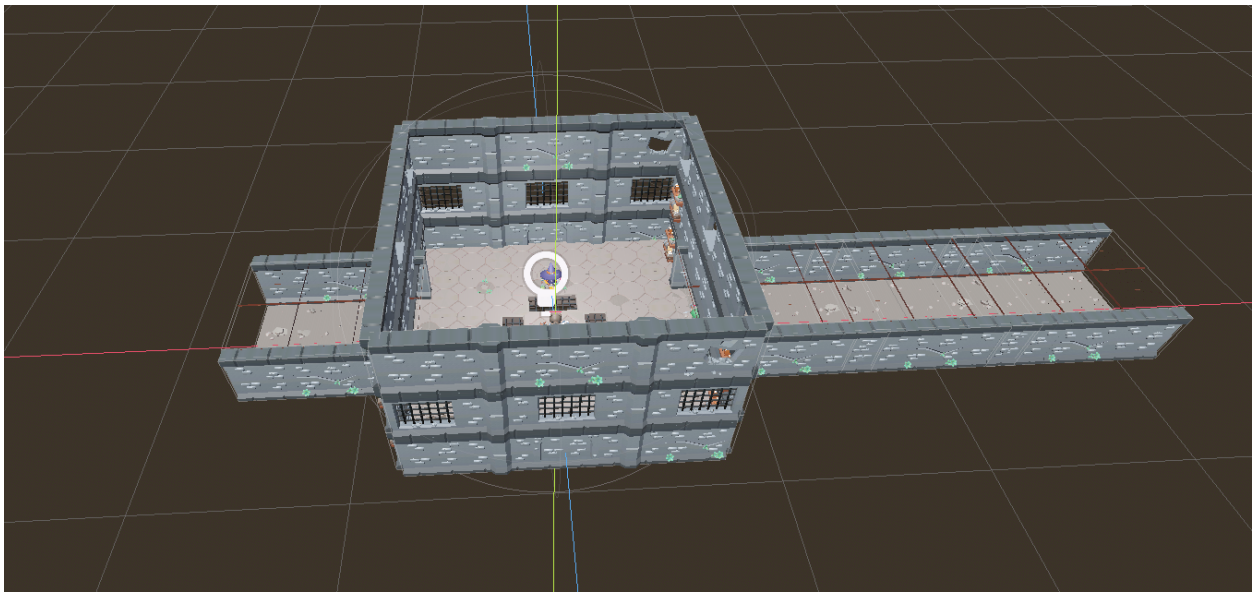
CS354R Alpha Report

For this project, we are creating a Wii Sports Resort Sword Fighting - esque game, where the player will be able to attack and block attacks from enemies, proceeding through different parts of the map until the player eventually loses all its health. We plan on implementing an AI for the enemies that learn from the user previous moves to choose their upcoming moves.

Current Implementation

We currently have an infinite map that the player can traverse through. The map was created using textures from KayKit's Dungeon Pack and looks like the following:



The ends of each individual arena are made of the same hallway scene. In the game manager, we check the player's location relative to the current arena's location to determine when to spawn a new arena:

```
if(p.get_location().x < a.global_position.x-200) :
    var arena = ArenaScene.instantiate()
    arena.position = Vector3(a.global_position.x-600,a.global_position.y,a.global_position.z);
    arena.add_to_group("arena2")
    arena.get_node("Enemy").add_to_group("enemy2")
    a2.remove_from_group("arena2")
    a2.add_to_group("arena1")
    e2.remove_from_group("enemy2")
    e2.add_to_group("enemy1")
    e2.connect("enemy_chop", enemy_chop);
    e2.connect("enemy_slice", enemy_slice);
    e2.connect("enemy_stab", enemy_stab);
    e2.connect("enemy_death", enemy_death);
    $Arena.add_child(arena,true)
    a.queue_free();
```

We also connect signals for the enemy in the new arena and despawn the current arena. This ensures that, at any given time, we only have two arenas, which are distinguished by the groups “arena1” and “arena2”.

We also have player actions (attacks/defenses) implemented. A player can currently attack in three ways: chop, slice, and stab. The player can also block in three ways: dodge, jump, and block. These attacks and blocks are the same for the enemy. Our logic for assigning blocks to attacks is the following:

Chop: Dodging a chop negates all damage. Blocking a chop deals less damage
Slice: Jumping over a slice negates all damage. Blocking a slice deals less damage
Stab: Blocking a stab negates all damage and causes knockback for the enemy. This also means that a successful stab hit deals extra damage.

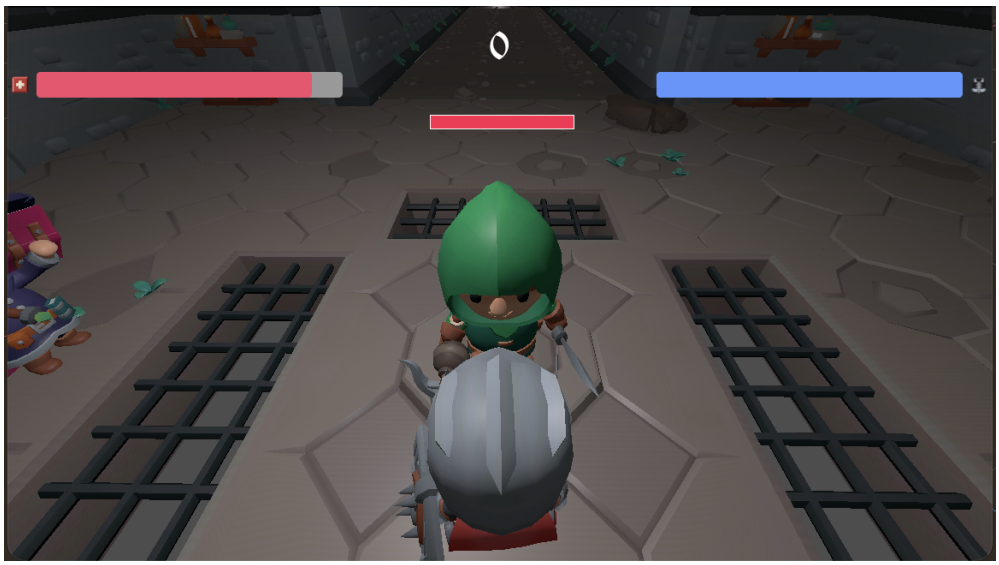
The code for the player and the enemy for attacking is relatively similar. We first are in an “idle” state, and transition to different attack states based on user input or based on a random number generator for the AI. **We plan on expanding on this AI to add weights for aggressiveness and action choice based on user past actions**.

We also have enemy interactions implemented. Each enemy can currently attack in the same 3 ways (chop, slice, stab) and block in the same 3 ways (dodge, jump, block). The interactions are the same in both directions (from the player to the enemy and from the enemy to the player).

Each implementation gets the other node’s current move and interacts with it according to how the player’s action and enemy’s action interact. Each action done by the current person checks for the ways it can interact with other actions of the opposing person. One of the interaction implementations is shown below where the enemy is chopping and changes the player’s health based on the player’s current action.

```
func enemy_chop() :
    var p = get_tree().get_first_node_in_group("player")
    var move = p.get_move()
    # dodge
    if move == 4 :
        return
    # block
    elif move == 6 :
        p.set_health(p.get_health() - 1)
        update_health(p.get_health())
    else :
        p.set_health(p.get_health() - 5)
        update_health(p.get_health())
    pass
```

We have in-game GUI elements that allow the user to see the player's health bar, the player's shield bar (goes down as block occurs and regenerates otherwise), and the enemy's health bar. This can be seen below.



When the enemy's health is depleted (shown by an empty health bar), the enemy dies and the player moves onto the next arena to face the next enemy. The player gets a point for every enemy they kill. When the player's health is depleted, the player dies and the game ends. Some of the code that manages the enemy death is shown below where it plays an animation and set's a boolean that will be used to kill the enemy so the player can continue onwards.

```
void Enemy::set_health(double p_health) {
    health = p_health;
    get_node<ProgressBar>("SubViewport/ProgressBar")->set_value(health);
    if (health <= 0) {
        dying = true;
        AnimationPlayer* animation = get_node<AnimationPlayer>(NodePath("Skin/AnimationPlayer"));
        animation->play("Death_A");
    }
}
```

Teammate Contribution

Sam: I worked on generating an infinite world by creating and destroying instances of the arena. I also worked on player actions and interactions with enemy.

Josh: I worked on setting up the initial map and lighting. I also worked on adding the physics to both players (mainly gravity) in case we wanted to add extensions to our game. I spent a majority of the time working on enemy actions and enemy interactions with the player through sending signals. Lastly, I worked on setting up an initial basic AI where the enemy AI picks actions at random.

Zach: I worked on the level design and animations. Using the textures provided by the Dungeon Pack, I built the arena and hallway, which were instanced for the infinite world generation.

Timeframe

We are currently on schedule regarding our implementation. We have a simple AI set up that attacks back but does not learn from user actions. We also have an infinite terrain, GUI that updates based on user inputs, and interactions between player and enemy.

Within the upcoming weeks, we plan on flushing out the AI by recording player actions, expanding on what AIs there are (aggressiveness scale), and potentially adding bosses. If we have time, we can expand on player interaction by introducing movement. However, this may disrupt our implementation of blocking.

We don't anticipate any setbacks at this time. Implementing an AI that is robust and complicated will be a bit complex, but we hope that our design decisions now will make it easy to store user actions and pass it to the enemy to make decisions.