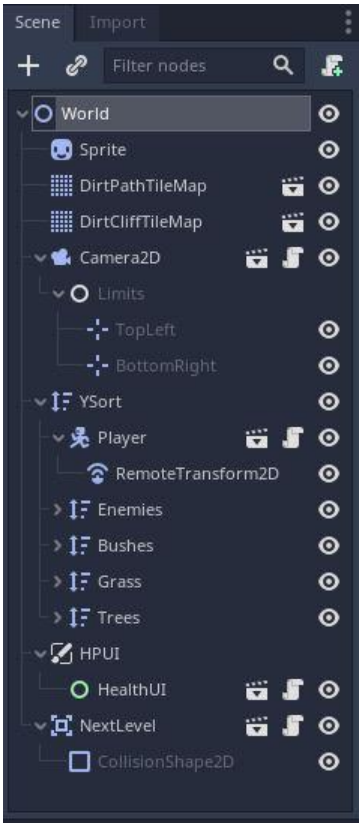
Criterion C: Development

1. Collections (lists, sets, maps, stacks, queues, and tree structures)
2. I/O for GUI controls in visual interaction
3. Inheritance to link code with similar functionality
4. Overriding to use the same or similar methods, and changing their functionality
5. Enum for player and enemy states
6. Array for enemy soft collisions
7. Rhythmic looping play



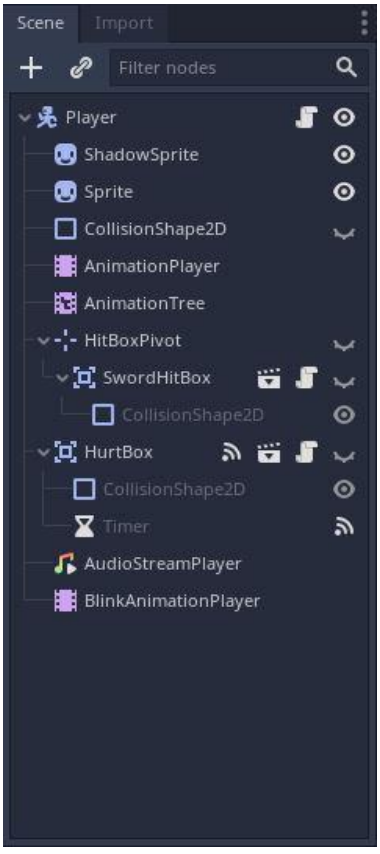
1. One of the first things that I needed to consider at the beginning of this project was how it was to be carried out. At first I was going into a java hard coded musical game but that would have been inefficient. A colleague of mine led me in the direction of game engines and specifically the Godot Game Engine. Even if I had to learn most of the syntax and organization of the engine, it would be much more efficient to use in collections of data. This allowed me to use several complex tree structures of nodes to use at my advantage. Moreover queues are often used to free certain nodes from a tree structure if needed.

```
46 func _ready():
47     randomize()
48     # Connects the stats to the rest of the objects inside 'Player'.
49     # Moreover if the player has no_health then the player is freed
50     stats.connect("no_health", self, "queue_free")
51     animationTree.active = true
52     swordHitbox.knockback_vector = roll_vector
```

2. Once I understood the network architecture of a tree, I focused on receiving the inputs of the client to develop and output. The I/O works to receive the input movement, roll, and attack of the player and display them. While most is simple, like the attack and the roll, the movement required more complexity in receiving movement as it had to listen for the direction that the client wishes to move in and then correctly output animation and velocity.

```
func move_state(delta):
    var input_vector = Vector2.ZERO
    input_vector.x = Input.get_action_strength("ui_right") - Input.get_action_strength("ui_left")
    input_vector.y = Input.get_action_strength("ui_down") - Input.get_action_strength("ui_up")
    input_vector = input_vector.normalized()
```

```
animationTree.set("parameters/Idle/blend_position", input_vector)
animationTree.set("parameters/Run/blend_position", input_vector)
animationTree.set("parameters/Attack/blend_position", input_vector)
animationTree.set("parameters/Roll/blend_position", input_vector)
animationState.travel("Run")
velocity = velocity.move_toward(input_vector * MAX_SPEED, ACCELERATION * delta)
```



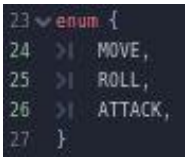
3. For the project, abstract where hardly present due to the fact that there aren't any interfaces of abstract classes. Instead this project uses inheritance like no other technique, it's the most simple but the most present throughout any of the code. This is used when other classes need used variables or override functions that are already present in the other classes. This is mostly seen in the player and enemy classes with their respective stat, hurtbox, and hitbox classes.

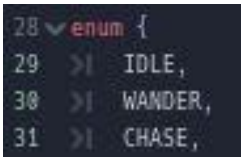
```
37  onready var animationPlayer = $AnimationPlayer
38  onready var animationTree = $AnimationTree
39  onready var animationState = animationTree.get("parameters/playback")
40  onready var swordHitbox = $HitBoxPivot/SwordHitBox
41  onready var hurtbox = $HurtBox
42  onready var blinkAnimationPlayer = $BlinkAnimationPlayer
```

4. Overriding was key to be able to change already present functionality for both the player and enemy nodes. I need to create similar classes but for different users, so I implemented the nodes carrying the script into scenes so that they may inherit the past functionality but also be susceptible to change in either the player or enemy nodes. Moreover, nodes have already preset functions that are being overridden by new code.
   Ex. Reset velocity to another function already present in the KinematicBody2D

```
func move():
    velocity = move_and_slide(velocity)
```

5. Once all the actions had been implemented, I required a way switch in between them. These actions are similar to each other so I placed them in an enum to then implemented based on the player in puts and enemies own rng for choosing a state. A state machine is an object that behaves accordingly to their present or past input/actions. This works wonderfully for the bat as the bat AI can switch states easily when a player gets noticed and needs to get back to their previous position after chase is failed, and for the clients present inputs into what they wish to carry out.





6. After some thoughtful debugging and error fixing, I noticed that the bats would visually merge, and this felt strange, so I seeked to fix this. I found that making some soft collisions would work but I needed it to work for all bats and not just two, so I used an array to detect how many bats are colliding and then push them against each other in opposite directions but only slightly.

```
func is_colliding():
    var areas = get_overlapping_areas()
    return areas.size() > 0

# This function get the direction in which the bats are colligin and pushes the
# in the opposite direction.
func get_push_vector():
    var areas = get_overlapping_areas()
    var push_vector = Vector2.ZERO
    if is_colliding():
        var area = areas[0] # An array is being called to detect all collisions.
        # Will be filtered to just enemies so they can push each other
        # without colliding on the hurtboxes.
        push_vector = area.global_position.direction_to(global_position)
        push_vector = push_vector.normalized()
    return push_vector
```

7. While I struggled working on the logic for a loop so that damage would occur according to the beats that occur in the song, I found a neat code on how to create a timer loop in code. While I could make this in a node, I was struggling to use two timers at once, so once I implemented this into code, I left it as is as it's much more comprehensible to the logic to change the damage. Moreover I adde beat meter so the client knows when it's beneficial to attack.[1]

[1] Godotengine.org. 2021. *How to make a part of a script always repeat (in loop) after X seconds? - Godot Engine - Q&A*. [online] Available at: <https://godotengine.org/qa/15160/how-to-make-part-of-script-always-repeat-in-loop-after-seconds> [Accessed 25 March 2021].