Currently, the game folder includes assets from the learning to make games in Godot by Brackeys as placeholder.

# Player

* Currently needs reworking. Too wild.

## Variables

speed : Integer

mouse\_position : Vector2

input\_vector : Vector2

aim\_vector : Vector2

roll\_vector : Vector2

## Constants

ROLL\_FRICTION

DASH\_FRICTION

# Sword

* currently the attack effects of the Player class. Handles the player’s attack and skill animations. Will probably need to be reworked

# Slime

* Class I made myself. Instance of a slime enemy w/ constructors for the variables.

## Variables

hp : float

speed : float

baseatk : float

target : CharacterBody2D

## Functions

new\_enemy -> Slime

* + - makes a new instance of the enemy and returns that instance

# Spawner

* Followed a video to learn how to create new instances of mobs from within the code itself. Currently makes a Slime enemy at a set point on ready and regularly at a random point of a set interval.
* Upgraded into Enemy\_Spawner

# Enemy\_Spawner

* handles the spawns. Utilizes a custom class “Spawn\_info” to hold the enemy scene to be spawned, and some variables pertaining to the time when the mobs can spawn, how many to spawn, and how long in between each spawn.
* currently has a Timer node to tick every second and do the spawn algorithm.
* gets a random position through the current viewport\_rect of the player (I assume game screen) with a bit of a padding from 1.1x viewport to 1.4x viewport. It picks a random side to spawn a mob in and uses randomly generated x and ys from that bigger rect. After getting that specific small bound, it does one more random generation to spawn the mob.

# Attack

* class created to hold all necessary information for attacks. The idea is to have attacks use this class to pass Attack objects instead to decouple the various objects.

# Hitbox

* area wherein objects can be hit from. Currently takes in a set amount of damage.
* group using the physics collision layers and masks.

# Hurtbox

* area wherein objects can be hurt from. Remember to set up the collision shape for each enemy type.
* group using the physics collision layers and masks.

# State Machine

* followed video to implement. Automatically takes in the child nodes/scripts to be grouped as states for the object.
* Handles the transitioning and properly directs the functions depending on the given State object

# State

* empty class to hold the functions, variables, signals. Should be overrided/extended to create custom states for the objects that need it.