Godot project documentation

Introduction:

After doing a simple research on what is possible to make using Godot game engine I came to the conclusion that the most interesting project idea would be to create a first person view horror game.

General description:

- The game will consist of several levels. Levels would be precreated (not randomly generated) to be able to create a simple storytelling. Each level should ideally be a chain of connected premises (something close to indoor actions of Outlast 1). Some of the rooms will have a source of light, some not. Doors are optional, too, and can be opened.
- Higher the level - more enemies will be spawned. Enemies will have a simple detection/chase logic. If they see you, they will go after you. After they collide with the protagonist the game is finished, that means you shell avoid them.
- Protagonist will be a human-like figure with the ability to hold some objects (possible lanterns of night vision cameras). He will be able to jump, run, shift and bend. The interaction with the world, say doors, is also possible.
- Textures will be stone-like mimicking the insides of an old abandoned building. The torch or lantern will be the main lighting source, but in some rooms there will be some dim bulbs or curtained windows.

Some of the aspects of the game can be changed during development as well as the set of features. For now I am doing research on internal image filters inside Godot to be able to mimic old infrared cameras, which will be usable by the player.

All of the in game mechanics will be presented in some sort of block schematic (they are now in development), as well as schemes of the levels and some rough 3d models of the objects.

Textures:

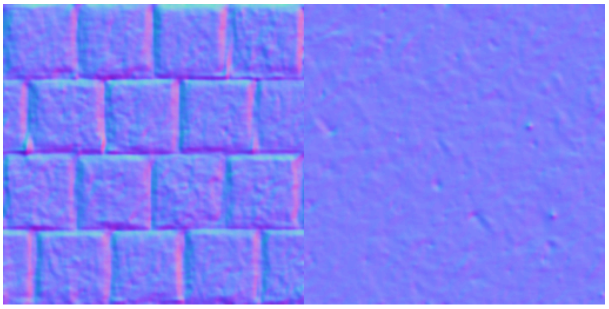
I have taken open source textures and reworked them.

On the following atlas image there are bricks - walls and sand - floor.

Using “Krita” I have made them tileable and changed the resolution to 256x256 for each of them.

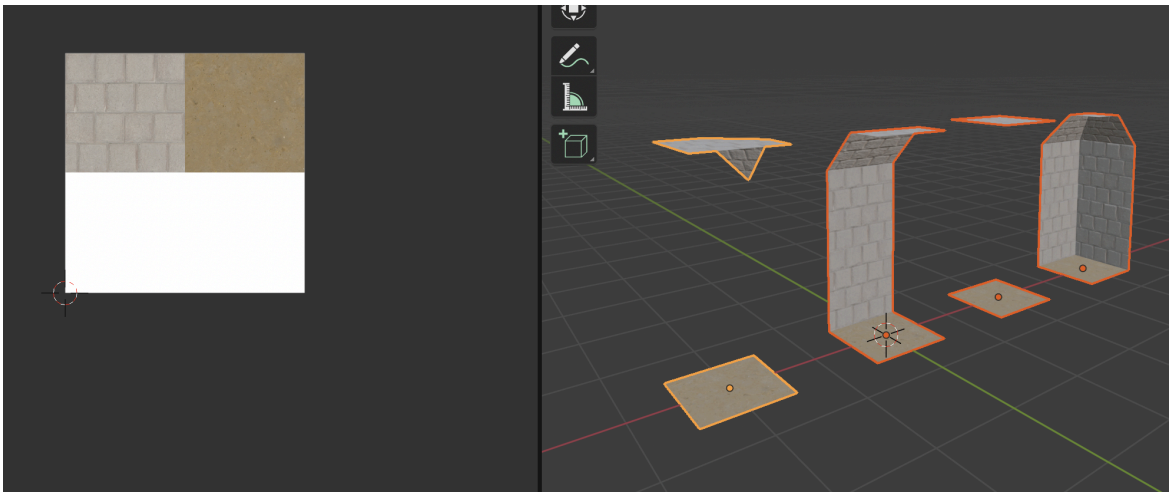
Atlas

Using “Laigter” I made a normalised map of the textures to make them 3D.

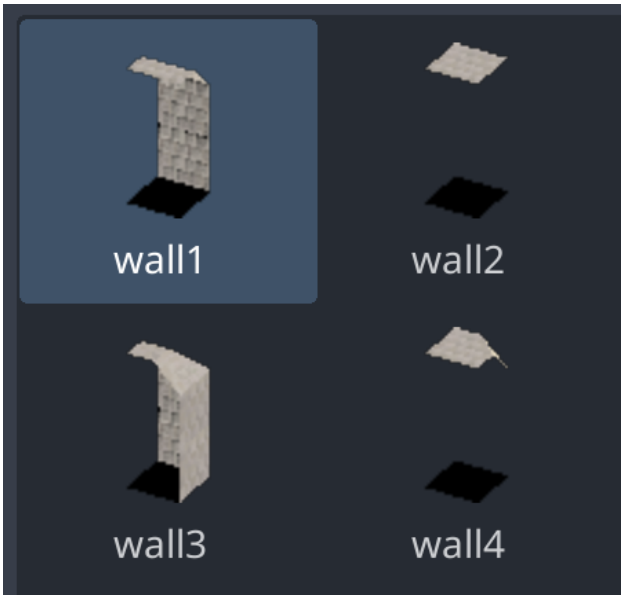


Atlas_n

Models:



The models are created in “Blender”. On the picture above the main walls, ceiling and floor schema’s / mesh sets are shown.

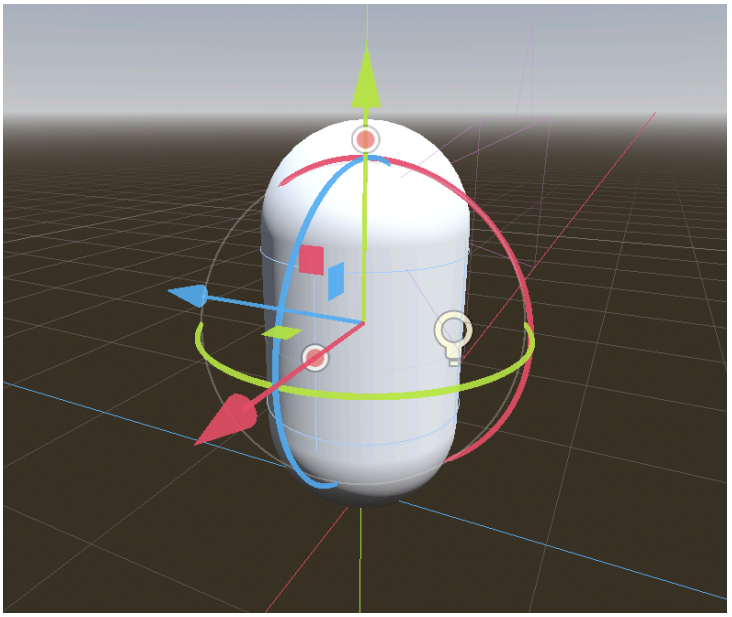


They are imported from Blender to Godot as a tileset scene. Before exporting from Blender it is important to add “-col” tag to all parts of the collection to make it collidable inside the Godot.


On the pictures above some of the models are shown (they are still in production and will be simplified).

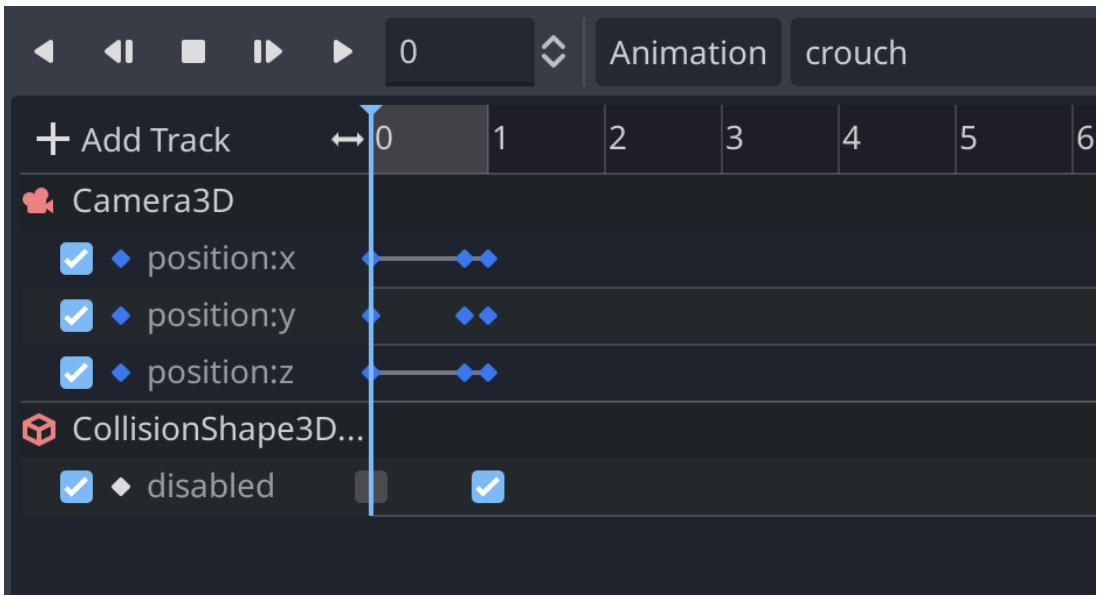
Character:

For the development purposes the character is for now capsule shaped.



It has two different collision shapes mapped to it. First is for an uncrouched position, second is for crouched position. Consequently, it has two different camera positions.

There are several animations connected to the Character_Body. One is specifically to be able to “crouch”.



On the picture above there is an example of the crouch animation, where the position of the camera changes and the collision shape for uncrouched position is being disabled.

Script for the movement and other actions is attached in the script section.

Enemy:

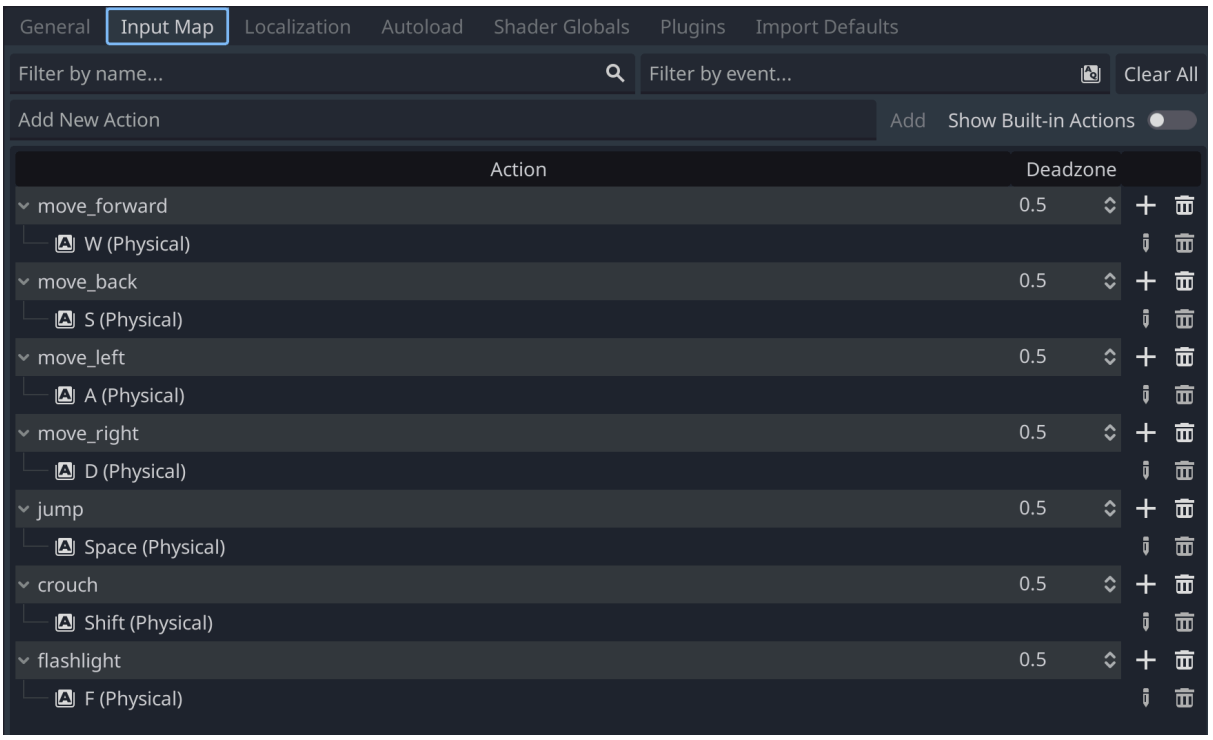
There will be two types of enemies. First type is a static ones, they stand still until you make a contact (collide) with them. Script for such type is shown in the corresponding section. Other type is moving enemies, which “walk” on precoded line until you make an eye contact with them (they will have a predefined field of view).

Levels:

There is 3 main levels and one level for tutorial purposes. Tutorial level does not have any enemies, it is only there to teach the player how to move, use the lantern and so on.
Other 3 levels will have enemies on them, and the main task is to get through the labyrinth either unnoticed, or fast.

Game mechanics:

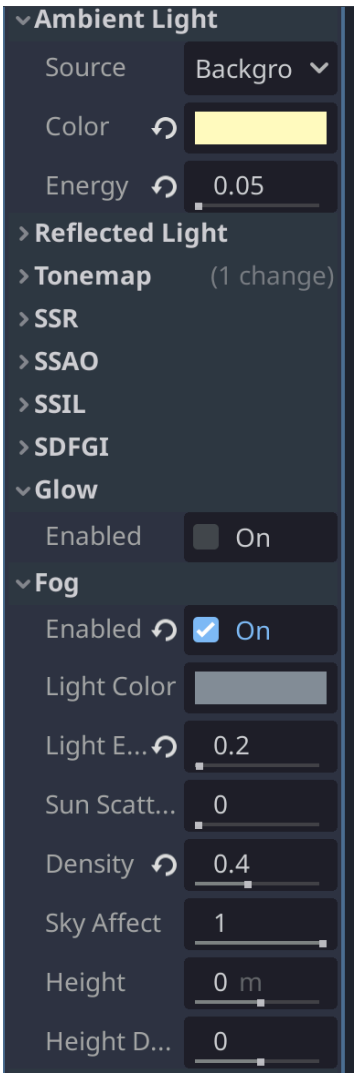
On the picture below the input map is shown

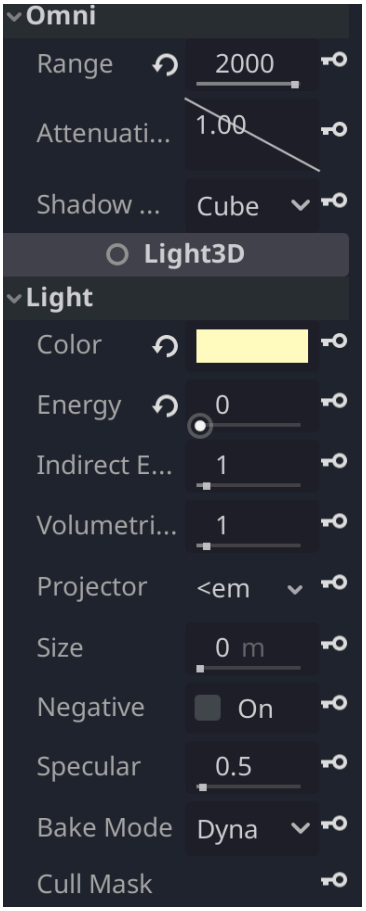


Movement is set up to “WASD” buttons. In addition, you have a mapped button “F” to use a lantern and “Shift” to crouch, as well as “Space” to jump.

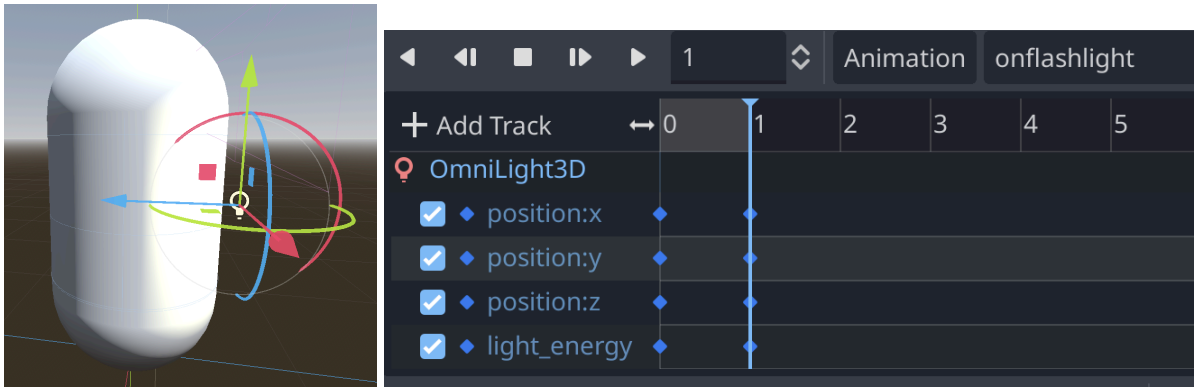
Lighting settings:

The game has 3 types of lighting set up. First one is basic wold_environment with directional light to create a base atmosphere. It also has a fog setting to shorten the view distance of the player.

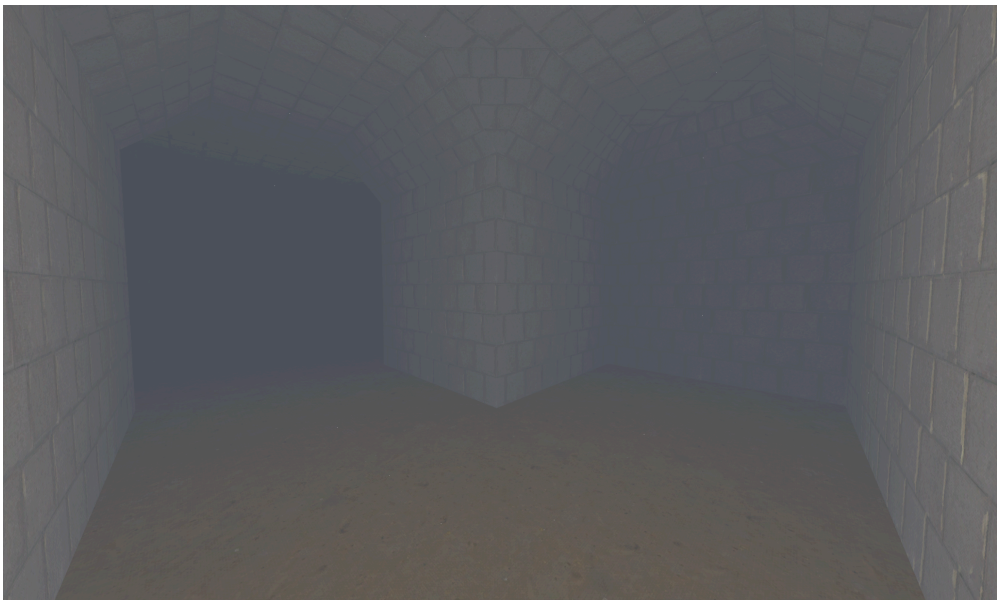




The player has an ability to use the lantern. The Omni_light node is used here. It is accessible with the press of “F” button and it has its own animation. It is located a bit to the side of the character body.

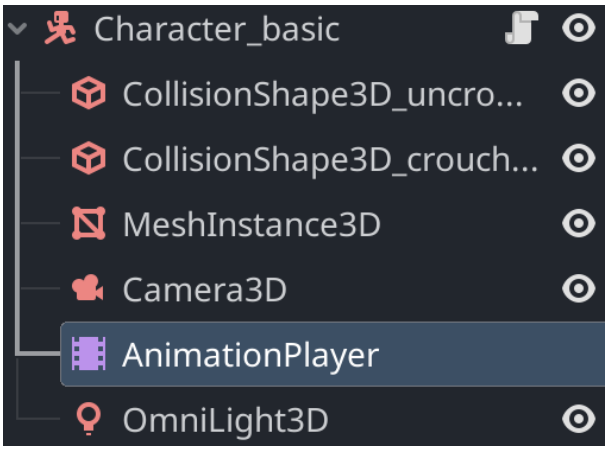


Game Screenshots:



Scripts and Nodes set:

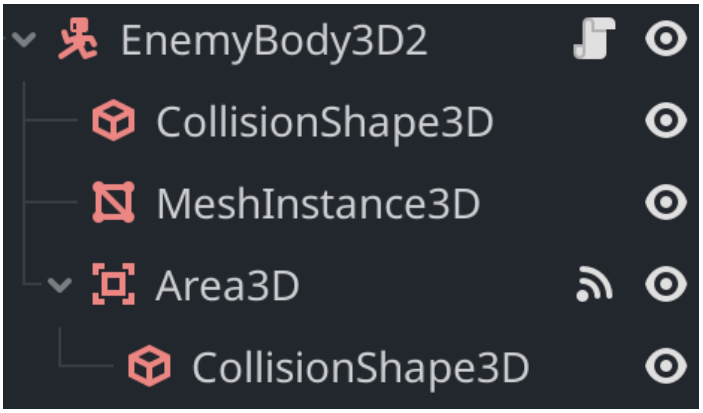
Character_body:




```
extends CharacterBody3D


const SPEED = 7.0
const CROUCHSPEED = 1.5
const JUMP_VELOCITY = 4.5
var crouched : bool
var flash : bool

# Get the gravity from the project settings to be synced with RigidBody nodes.
var gravity = 10
var mouse_sensitivity = 0.001
var camera_rotation_limit = 70.0

func _rotate_player_and_camera(relative_motion):
	# Rotate player left/right
	rotation.y -= relative_motion.x * mouse_sensitivity

	# Get current camera rotation
	var camera_rotation = $Camera3D.rotation_degrees
	# Rotate camera up/down
	camera_rotation.x -= relative_motion.y * 100 * mouse_sensitivity
	# Clamp the camera's vertical rotation
	camera_rotation.x = clamp(camera_rotation.x, -camera_rotation_limit, camera_rotation_limit)
	$Camera3D.rotation_degrees = camera_rotation

func _input(event):
	if event is InputEventMouseMotion:
		_rotate_player_and_camera(event.relative)

func _physics_process(delta):
	# Add the gravity.
	if not is_on_floor():
```

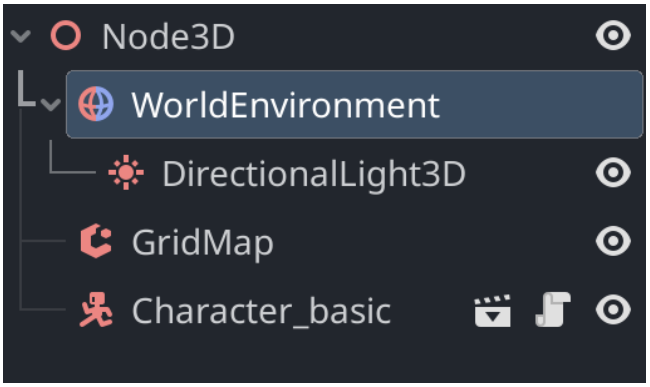
```
                velocity.y -= gravity * delta

        # Handle jump.
        if Input.is_action_just_pressed("ui_accept") and is_on_floor():
                velocity.y = JUMP_VELOCITY

        # Get the input direction and handle the movement/deceleration.
        # As good practice, you should replace UI actions with custom gameplay actions.
        var input_dir = Input.get_vector("move_left", "move_right", "move_forward", "move_back")
        var direction = (transform.basis * Vector3(input_dir.x, 0, input_dir.y)).normalized()
        var speed = SPEED
        if Input.is_action_pressed("crouch"):
                speed = CROUCHSPEED
                if !crouched:
                        $AnimationPlayer.play("crouch")
                        crouched = true
        else:
                if crouched:
                        var space_state = get_world_3d().direct_space_state
                        var result =
space_state.intersect_ray(PhysicsRayQueryParameters3D.create(position, position + Vector3(0,2,0), 1, [self]))
                        if result.size() == 0:
                                $AnimationPlayer.play("uncrouch")
                                crouched = false

        if Input.is_action_just_pressed("flashlight"):
                if flash:
                        $AnimationPlayer.play("offflashlight")
                else:
                        $AnimationPlayer.play("onflashlight")
                flash = !flash

        if direction:
                velocity.x = direction.x * speed
                velocity.z = direction.z * speed
        else:
                velocity.x = move_toward(velocity.x, 0, speed)
                velocity.z = move_toward(velocity.z, 0, speed)

        move_and_slide()
```

Enemy:

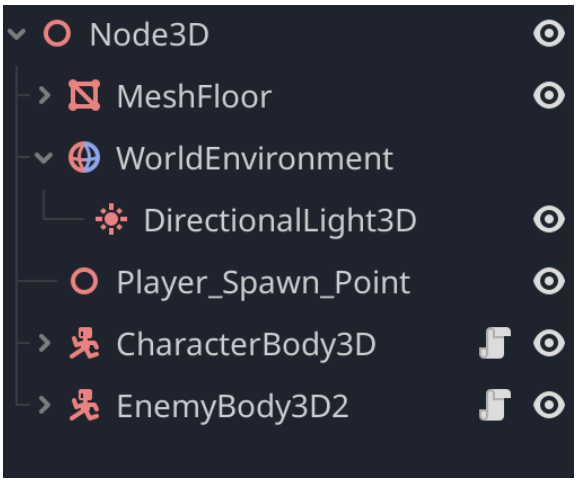


```
extends CharacterBody3D

var player_detected = false
var player_position = Vector3()
var speed = 5.0
var gravity = -0.1


func _ready():
	# Assuming 'Area3D' is the correct node name. Adjust if it's different.
	var area = $Area3D
	area.connect("body_entered", self, "_on_area_3d_body_entered")
	area.connect("body_exited", self, "_on_area_3d_body_exited")

func _on_area_3d_body_entered(body):
	if body.name == "CharacterBody3D": # Make sure to replace this with your player's node name
		player_detected = true
		player_position = body.global_transform.origin

func _on_area_3d_body_exited(body):
	if body.name == "CharacterBody3D": # Make sure to replace this with your player's node name
		player_detected = false

func _physics_process(delta):
	if player_detected:
		var direction = player_position - global_transform.origin
		direction.y = 0 # Keep the movement horizontal
		direction = direction.normalized()
		velocity.x = direction.x * speed
		velocity.z = direction.z * speed
	else:
		velocity.x = 0
		velocity.z = 0

	# Gravity is applied separately to ensure it affects the character
	velocity.y += gravity * delta

	# move_and_slide expects velocity as the first parameter
	move_and_slide()
```

First level:



Test level:



Project directory: