DREAMER

By: Aurelio Medina, Basil Henry, Erik Myers

Executive Summary

The final project represents the development of a 2D side scrolling platformer game created using the Godot game engine. The main features of the game are a playable character that can run, jump, collect collectibles, and attack. The player navigates through dangerous obstacles. The objective is to survive and collect 5 diamonds on each level, then advance to the next level 3 times.

The main purpose of this project was to demonstrate core game development skills. Also providing us with an excellent portfolio piece for our job hunt endeavors. We handled animations, enemy behavior, collision detection, UI design, and sound integration. The game brings a single player gameplay, survival challenges, sprite animations, enemy logic, and full menu systems.

The game was developed using Godot 4.3 and GDScript. We used custom assets from itch.io. The game architecture design follows a structured modular design. This report will go further in depth in the purpose, architecture design, requirement specifications, design, and our test plans used, while also providing a user manual for our game.

Table of Contents

1. Introduction

GitHub Link:

1.1. Purpose and Scope

The main purpose of this project was to show our understanding of core game development concepts. We demonstrated our knowledge of Godot game engine, collaboration, and our ability to research. The project serves as a portfolio piece for each of our resumes. The scope of the project includes a single player 2D platformer. It features the basic menus, combat abilities, and interactions.

1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

The main time to use our product is when you want a nice challenging experience via a video game. The goal is to survive and collect diamonds to advance to the next level, while dodging/attacking enemies.

1.3. Structure of the Document

The structure of the document is introducing the product, elaborating on the project management plan, showing the requirement specifications, architecture, design, and the test plan we used.

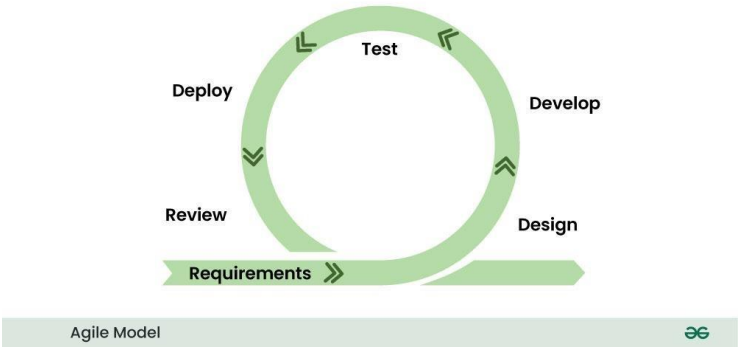
2. Project Management Plan

2.1. Project Organization :

Team members and Roles:

- Aurelio Medina: Main Designer of level 1 along with User Interface and Enemies design
- Basil Henry: Main Designer for level 2
- Erik Myers: Main Designer of level 3 along with Implementing music

2.2. Lifecycle Model Used



2.3. Hardware and Software Resource Requirements (also describe what new software or hardware features each team member learned during the project)

Throughout the project we decided to use the windows operating system

Operating system	CPU	GPU	Storage	DirectX	Resolution
Windows 10 and 11	Intel Core i3	Integrated Graphics	At least 200mb	Version 11	At least 1280x720

2.4. WBS, Deliverables and Schedule

Work Break Down Structure:

Task	Description	Assign to
Game concept	discussing what we want the overall game to be about	All Team members were involved
Level 1 Overall Design	The overall design of Level 1	Aurelio Medina
Level 2 Overall Design	The Overall design of Level 2	Basil Henry
Level 3 Overall Design	The Overall design of Level 3	Erik Myers
UI Design	What the overall UI of the game would look like	Aurelio Medina
Music	The music use and implemented in the game	Erik Myers
Testing	Making sure the game run smoothly by fixing all major bugs	All Team members were involved

Schedule:

Week 1: During the first week of the developing project dreamer, we mainly focus on how the game story and gameplay style for the video game and eventually we decided to go with a 2d platformer like that of Mario or Sonic the hedgehog game series.

Week 2 to 3: During week two and three we our team decided to gather assets from various websites with the main one we used to know as “itch.io” that various pixel and 2d asset that can be used for free.

Week 4: We decided that it would be faster progress if each team member focused on an individual level and then we began building our own individual level based on the theme we decided to use.

Week 5: After experimenting with parallax background, we decided to switch our focuses to tile map

which not only improves the game visually but also makes progression much easier.

Week 6: Enemies were implemented into the game to make it more challenging

Week 7 to 8: Testing and polishing up the overall demo of the game

2.5. Monitoring, Reporting, and Controlling Mechanisms

To effectively monitor and report our progress we use git-up

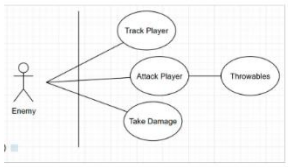
- **GitHub** was used to track code changes with each member contributing updates.
- **Weekly Team Meetings** were held either in person who online meet up via discord in order to discuss progress, assign tasks, and solve problems.
- **Internal Playtesting** was done regularly to monitor the game's progress and catch bugs early.
- **Progress Updates** were shared with the instructor as needed, and changes were made based on feedback.

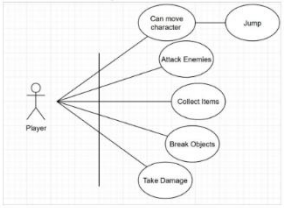
3. Requirement Specifications

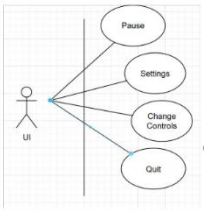
3.1. Stakeholders for the system

- **Development Team** – Responsible for building and maintaining the game, including coding, art, and design
- **Players** – These achnare the main users of the game who will play and provide feedback on gameplay, visuals, and controls
- **Instructor** – Evaluates the game based on project goals, grading criteria, and learning outcomes.
- **Classmates** – Peers who may test the game during presentations and offer feedback.
- **Future Developers** – Students or developers who may use this game as a reference or expand it in the future

3.2. Use case model







3.2.1. Graphic use case model

3.2.2. Textual Description for each use case

- The first use case model shows how enemies track players, attack players, and take damage
- The second use case model shows the use cases for the player which has the most interactions. The player can take damage, deal damage to enemies, collect items and break objects.

- The third use case model is the UI and shows that the user can pause, change settings, change controls and quit the game.

3.3. Rationale for your use case model

Use Case	Type	Rationale
Move Character	Core Function	Essential for gameplay; allows player navigation.
Pause Game	System Control	Necessary for user convenience and breaks.
Adjust Settings	Configuration	Lets players modify controls, audio, etc.
Take Damage	Game Mechanic	Critical for challenge/health systems.
Collect Items	Gameplay	Includes "Cookie Items" (power-ups) and "Boots Objects" (equipment).
Change Controls	Configuration	Allows rebinding keys/buttons for accessibility.

3.4. Non-functional requirements

- Must run smoothly on mid-range laptops
- Controls should be responsive
- Art should match dream theme

4. Architecture

4.1. Architectural Design

The game uses a modular architecture design while utilizing Godots scene system. We use multiple different scenes for different purposes. We have scenes for characters/enemies/menus/levels. We also have scripts for various activities. We have manager scripts for menus, music, and overall game structure.

4.2. Technology, software, and hardware used

We use Godot 4.3 version to aid in the creation. We utilized GDScript to program. We found custom art on itch.io for the sprites and layer map. We also created our own music. Hardware should be any basic pc/laptop. It is on a windows executable file.

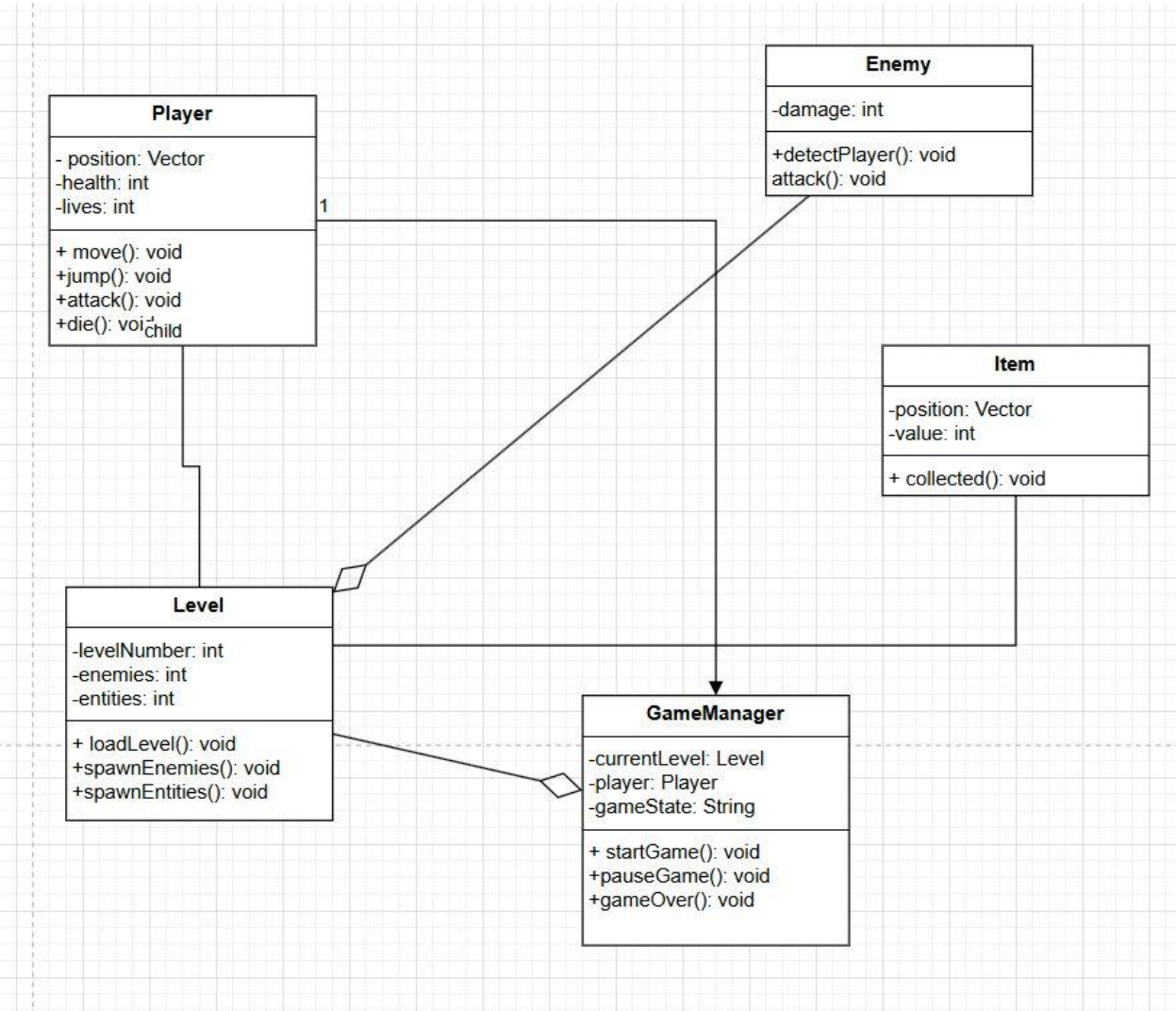
4.3. Rationale for your architectural design

The modular design of our scenes was used for ease of access. We had a specific structure in file management, where all level 1 assets go into that specified folder, level 2, level 3, etc. Godot is a beginner friendly free game development engine, so it was easy to pick.

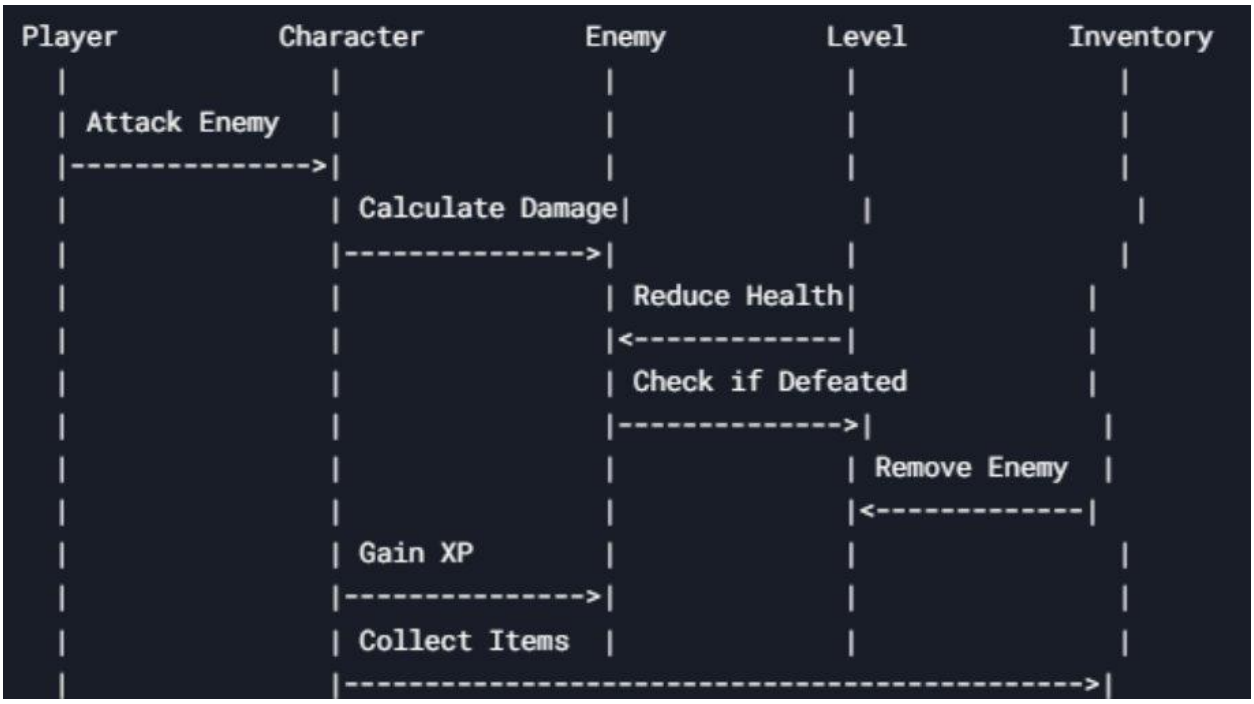
5. Design

5.1. GUI (Graphical User Interface) design

5.2. Static model – class diagrams



5.3. Dynamic model – DFD and/or sequence diagrams



5.4. ERD, Database design

N/A

6. Test Plan

6.1. Requirements/specifications-based system level test cases

- windows 7/8/10 or higher computer

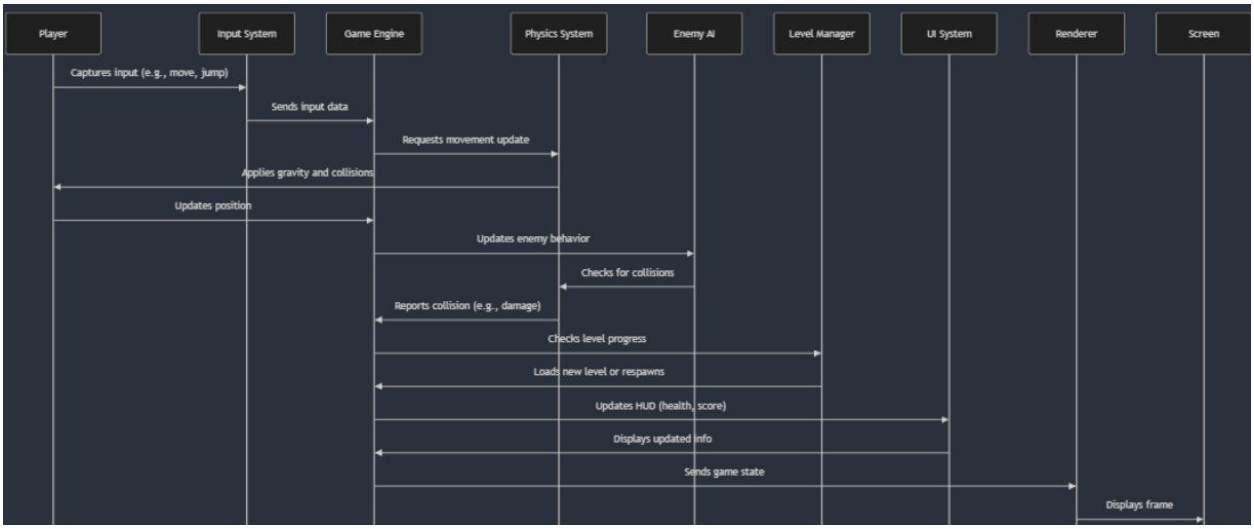
- 8gb of ram

-keyboard

-integrated graphics card

-3gb of storage

6.2. Class/ Unit test cases



6.3. Traceability of test cases to use cases

Test Scenario	Steps taken during the test	Expected output	Status
Start game from menu	1. Launch the game 2. Click 'Start Game'	Game scene loads successfully	Pass
Navigate to settings	1. Launch the game 2. Click 'Settings'	Settings screen opens	Pass
Pausing the game	Clicking the pause button during a gameplay	Game pause after clicking the pause button	Pass
Player movement: jump along with moving left and right	Click the right and left keys on the keyboard and space bar for jumping	Player moves left and right after pressing these input Player jumps after pressing the input	Pass
Player attack enemies when pressing the Z key on the keyboard	Pressing Z will make the player attack	Player will attack enemies ai and deal damage point	Pass
Enemies will attack player in a certain range	Player enter enemies' range and will attack	Player will lose health when encounters enemies ai	pass
Player item collecting	Player item count goes up as he collect more item	The player will gain a positive one numerical value depending when collecting each item	Pass

6.4. Integration Plan

Input System

- Verify key binds (e.g., Z = attack, Space = jump).
- Test UI buttons (Start, Settings) trigger correct actions.

Physics System

Confirm collisions:

- Player + items → Collection works.
- Player + enemies → Damage applied.

Game Engine

Validate pause functionality:

- Time stops, UI menu appears.

Track health (Hao) changes from enemy attacks.

UI System

Check HUD updates:

- Item count, health, pause menu.

Energy AI

Test enemy behavior:

- Attacks only in range.
- Responds to player actions.

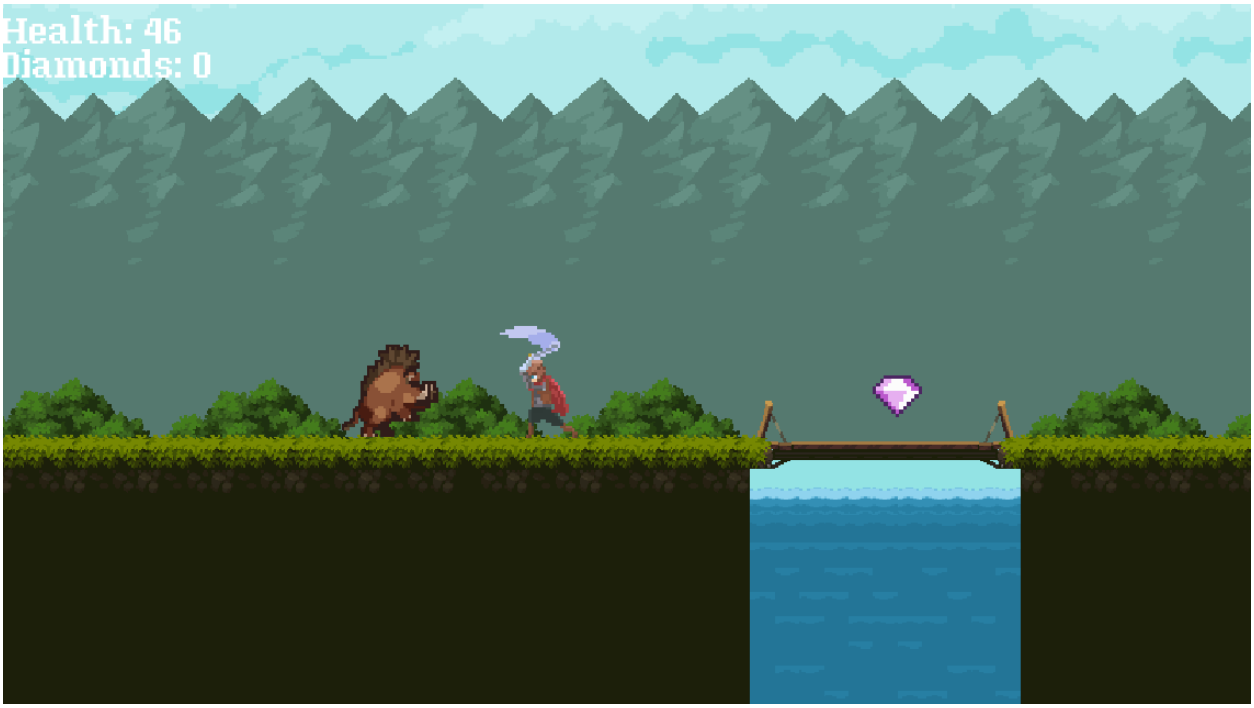
7. User Manual

Installation/Setup – The user may download it on GitHub and unzip the folder. Then run the executable file “Dreamers”.

Controls - The player can move up/down/left/right via the arrow keys. While jumping and attacking with space and z respectively. Esc button is used to open the pause menu. While clicking with the mouse is viable as well.

Gameplay – The player will navigate through levels by running/jumping/attacking enemies/avoiding obstacles. Defeat the enemies via attacking. Collect diamonds to advance to the next level, to unlock the door.

Appendix A: Screen shots


Health: 46
Diamonds: 0


Health: 44
Diamonds: 0

Appendix B: Code

Here are some notable scripts, not all of the scripts we used just some main one s that were used.

GAME.GD

```
extends Node
var Diamonds = 0;
var Health = 50;
func reset():
	Health = 50
	Diamonds = 0
```

PLAYER.GD

```
extends CharacterBody2D

@onready var pause_menu = $PauseMenu
@onready var animated_sprite_2d = $AnimatedSprite2D
@onready var attack_area = $AttackArea

const SPEED = 500.0
const JUMP_VELOCITY = -500.0

var gravity = ProjectSettings.get_setting("physics/2d/default_gravity")
var is_attacking = false

func _ready():
	animated_sprite_2d.animation_finished.connect(_on_animation_finished)
	attack_area.monitoring = false
	add_to_group("player") # So the trigger recognizes it

func _physics_process(delta):
```

```
	if not is_on_floor():
		velocity.y += gravity * delta

	var input_axis = Input.get_axis("ui_left", "ui_right")
	if input_axis:
		velocity.x = input_axis * SPEED
	else:
		velocity.x = move_toward(velocity.x, 0, SPEED)

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

	if Input.is_action_just_pressed("Attack") and not is_attacking:
		attack()

	move_and_slide()
	update_animations(input_axis)

	if Game.Health <= 0 or global_position.y > 800:
		queue_free()
		MusicManager.play_menu_music()  # ✅ switch back to menu music
		get_tree().change_scene_to_file("res://menu.tscn")

func update_animations(input_axis):
	if is_attacking:
		return

	if not is_on_floor():
		animated_sprite_2d.play("DoubleJump")
	elif input_axis != 0:
		animated_sprite_2d.flip_h = input_axis < 0
		animated_sprite_2d.play("Walk")
```

```
        else:
                animated_sprite_2d.play("Idle")

func attack():
        is_attacking = true
        animated_sprite_2d.play("Attack")
        attack_area.monitoring = true

        # Flip attack hitbox based on sprite direction
        if animated_sprite_2d.flip_h:
                attack_area.scale.x = -1  # Flip the hitbox left
        else:
                attack_area.scale.x = 1   # Default hitbox to the right

func _on_animation_finished():
        if animated_sprite_2d.animation == "Attack":
                is_attacking = false
                attack_area.monitoring = false

func _on_attack_area_body_entered(body):
        if body.has_method("take_damage"):
                body.take_damage(1)  # Deals 1 damage (customize as needed)
```

DIAMONDS.GD

```
extends Area2D


func spawn_feedback():
        var scene_to_spawn = preload("res://Pickups/Feedback/feedback.tscn")
```

```
	var new_scene_instance = scene_to_spawn.instantiate()
	get_tree().current_scene.add_child(new_scene_instance)  # Add the instance as a child of the current scene
	new_scene_instance.global_position = global_position



func _on_body_entered(body):
	if body.name == "Player":
		Game.Diamonds += 1
		spawn_feedback()
		queue_free()
#  Check if player reached 5 diamonds
		if Game.Diamonds == 5:
			var level = get_tree().current_scene  # Assuming Door nodes are in the current scene
			var door_closed = level.get_node("DoorClosed")
			var door_open = level.get_node("DoorOpened")
			# Enable level portal
			door_open.get_node("EnterArea").monitoring = true



			# Open the door visually
			door_closed.visible = false
			door_open.visible = true
```