GAME DESIGN DOCUMENT FOR METEOR BLITZ: SPACE SURVIVAL.

This is the entire documentation of the process of making the game in GODOT for future references and ease.

What this document will consist of?

This documentation will include the Information about the Game, the idea and concept, references, planning of the project, how each step is executed including with the steps and script(code) if needed.

Any links from where the assets were downloaded will be mentioned accordingly.

Concept of the Game:

The basic concept is, You Start the Game on main menu, start the level, meteors fly randomly and you survive against time by shooting them. If you die (deplete all your health) you go to game over screen with your score and can start again.

Planning of the Project:

I intend to create the game in iteration format. This will be done by creating basic functional scenes at first and then getting creative with it. For better understanding:

- Creation of Level scene.
- Creation of Game over scene.
- Creation of Main menu scene. (Experimental)
- Creation of 2 levels (Normal and Hard)
- Creation of Pause button.

CHAPTER WISE EXECUTION

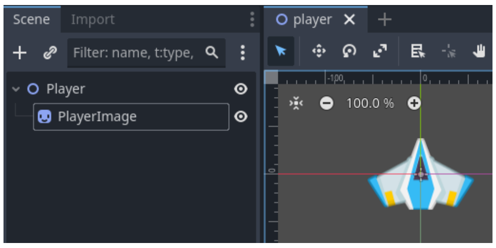
IMPORTING ASSETS AND SETTING UP THE ENGINE

Downloaded assets from: https://kenney.nl/

Load the assets into the engine by adding the assets folder into your game folder.

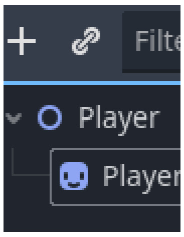
CREATING SCENES

Create a Node2D in the Scene Section and add a Sprite2D Node to it. Rename the Node2D to Player and Sprite2D to PlayerImage. Add ship png to the Sprite2D.



Create a new Scene called Level as Node2D.Save it.

LINKING PLAYER SCENE TO LEVEL SCENE: DONE BY THIS ICON



We need to create one more Sprite2D node for the level for giving it a background. Create a Sprite2D and give it the png.

Similarly Create a Meteor Scene and place it inside the Level Scene.

SCRIPTING:

You always attach a script to a node and that script can change the attribute of the node. You can change position, size, rotation, texture etc. Whatever the node has in the inspector can be changed.

It is done by this icon.

BASIC OF SCRIPTING:

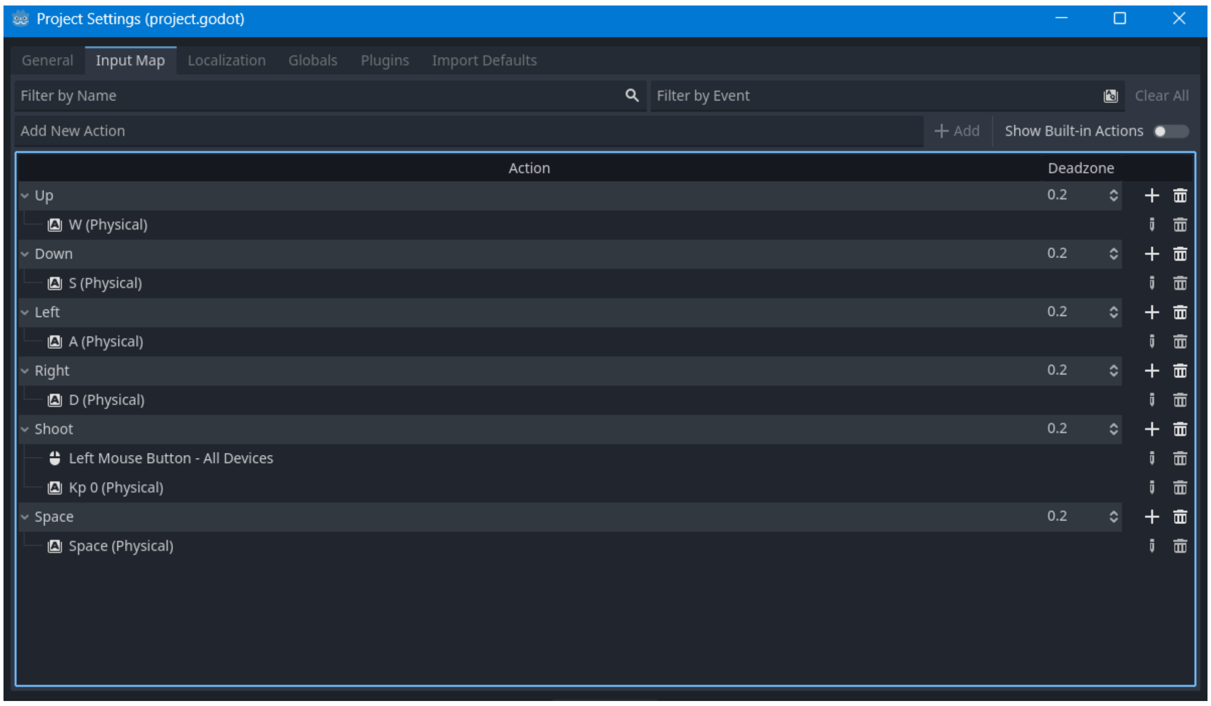
- Any node's starting state is defined in this code:
  # Called when the node enters the scene tree for the first time.
  func _ready() -> void:
  pass # Replace with function body.
- Any node's continuous state is defined in this code:
  # Called every frame. 'delta' is the elapsed time since the previous frame.
  func _process(delta: float) -> void:
  pass

STARTING POSITION OF THE PLAYER AND MOVEMENT INPUT

For starting position of the player. In the player script in ready function add position for the player.

```
func _ready() -> void:
	position = Vector2(550,550)
```

For Input. Go to Project>Project Settings> Input map. Add and name the input and give the input the keyboard or controller mapping.


Project Settings (project.godot)

General | Input Map | Localization | Globals | Plugins | Import Defaults

Filter by Name | Filter by Event | Clear All

Add New Action | + Add | Show Built-in Actions

Action	Deadzone
Up	0.2
W (Physical)	
Down	0.2
S (Physical)	
Left	0.2
A (Physical)	
Right	0.2
D (Physical)	
Shoot	0.2
Left Mouse Button - All Devices	
Kp 0 (Physical)	
Space	0.2
Space (Physical)	

Inside Player script: For movement of the Player.

```
func _process(delta: float) -> void:
	var direction = Input.get_vector("Left","Right","Up","Down")
	velocity = direction * speed
```

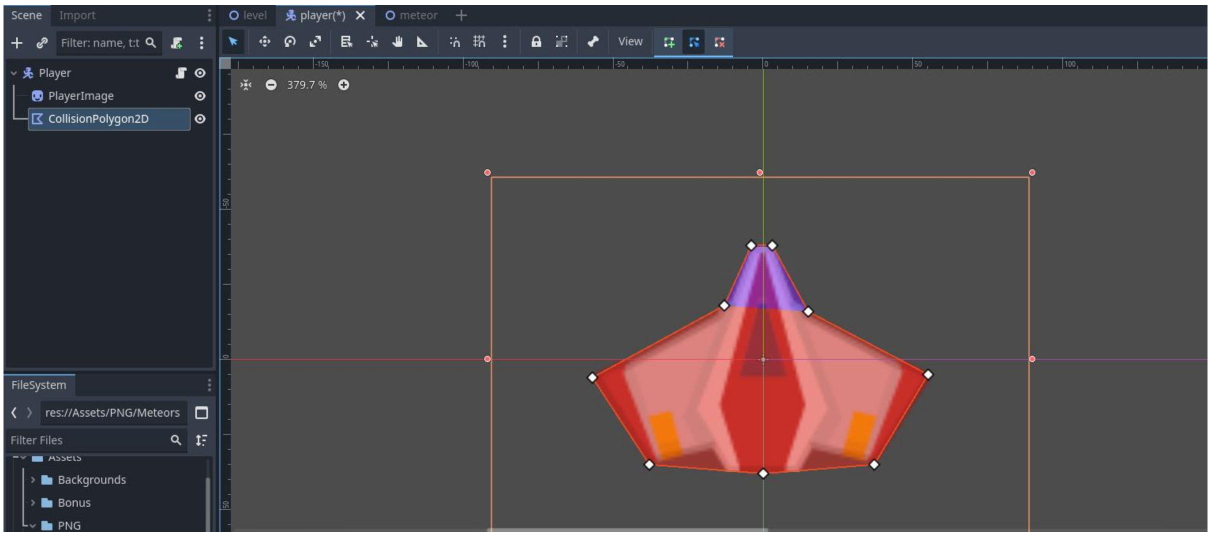
COLLISION PHYSICS AND SIGNALS

Types of collision Nodes in Godot:

- **Area2D:** Just an area but it is good for detecting collision.
- **StaticBody2D:** Has collisions but cannot move.
- **CharacterBody2D:** Has collisions and can move (via code)
- **RigidBody2D:** Has collisions and can move (via physics)

PLAYER SCENE:

Convert the type of the Player scene to CharacterBody2D. And also change it in the script. Assign Shape to the Player by adding a new node called CollisionPolygon2D.



LEVEL SCENE:

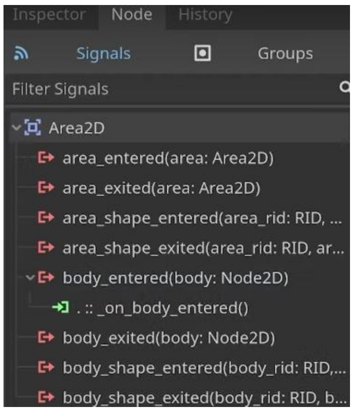
Create a new StaticBody2D node called Borders. Add a child node in it called CollisionShape2D. And Give Rectangle shapes for the Invisible wall.

METEOR SCENE:

Change the type of Meteor node to Area2D, change the script and add a CollisionShape2D to it as a node and give it shape.

- Every node can execute a code when an event occurs. For example, Area2D can run a function when a physics body is enters its area. This code execution is called Signal.

Connect the Body entered signal to meteor script:


Inspector Node History

Signals Groups

Filter Signals

Area2D
- area_entered(area: Area2D)
- area_exited(area: Area2D)
- area_shape_entered(area_rid: RID, ...
- area_shape_exited(area_rid: RID, ar...
- body_entered(body: Node2D)
  - . :: _on_body_entered()
- body_exited(body: Node2D)
- body_shape_entered(body_rid: RID,...
- body_shape_exited(body_rid: RID, b...

Script inside meteor:

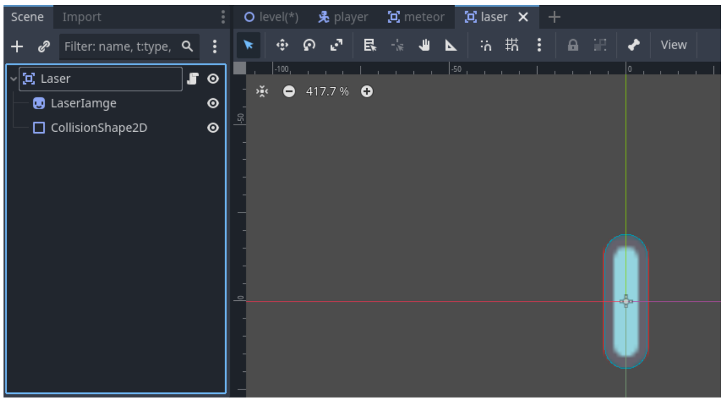
func _on_body_entered(body: Node2D) -> void:

pass # Replace with function body.

What this basically does is, it runs the function inside the code when every time any Body enters its area.

CREATE LASER SCENE:

Create a Area2D Scene called laser and give it a Sprite2D and CollisionShape2D node. Add Shape, Sprite and Script



SPAWNING METEORS IN LEVEL:

We need to spawn meteors every time the timer runs out in the level scene. This will be done with the help of signals and script.

Step 1: In the Level scene create a timer node.

Step 2: Connect the timer_timeout signal to the level script.

Step 3: Create a variable in which the meteor scene will be loaded:

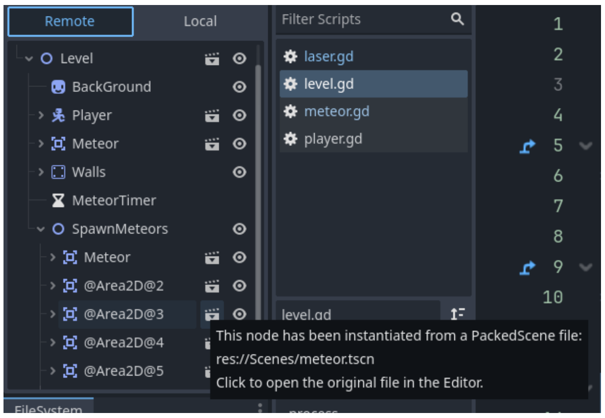
var meteor_scene: PackedScene = load("res://Scenes/meteor.tscn")

Step 4: Create an instance where the meteor_scene will start. And add the Instance to a new Node2D in Level called SpawnMeteors. What this will do is, create a new child node the SpawnMeteors every time the timer runs out.

```
func _on_meteor_timer_timeout() -> void:

        var meteor = meteor_scene.instantiate()

        $SpawnMeteors.add_child(meteor)
```

We can later add movement to the meteors that are spawned.

Expected output in remote:



```
Level
    BackGround
    Player
    Meteor
    Walls
    MeteorTimer
    SpawnMeteors
        Meteor
        @Area2D@2
        @Area2D@3
        @Area2D@4
        @Area2D@5
```

laser.gd
level.gd
meteor.gd
player.gd

level.gd

This node has been instantiated from a PackedScene file:
res://Scenes/meteor.tscn
Click to open the original file in the Editor.

Step 5: For Randomly spawning the meteor and then bringing it down we need to add code to the meteor script particularly in the **func _ready ()** because we need these attributes when the meteor scene starts every time. For that:

Step 6: Create a variable which generates a random number:

var rng: = RandomNumberGenerator.new ()

Step 7: In **func _ready ()**

```
# Declaring width, Random X and Y cordinates
    var width = get_viewport().get_visible_rect().size[0]
    var random_x = rng.randi_range(0,width)
    var random_y = rng.randi_range(-150,-50)
# Spawn Position of the meteor
    position = Vector2(random_x,random_y)
```

Step 7: In **func _process(delta):**

```
func _process(delta):
    # Bringing the meteors down, giving angle and rotation.
    position += Vector2(0,1.0) * meteor_speed * delta
```

Step 8: Giving random speed, rotation, direction and graphic png to the meteors.

```
# Declare global variables:
var direction_x : float
var rotation_speed : int
```

In the ready function:

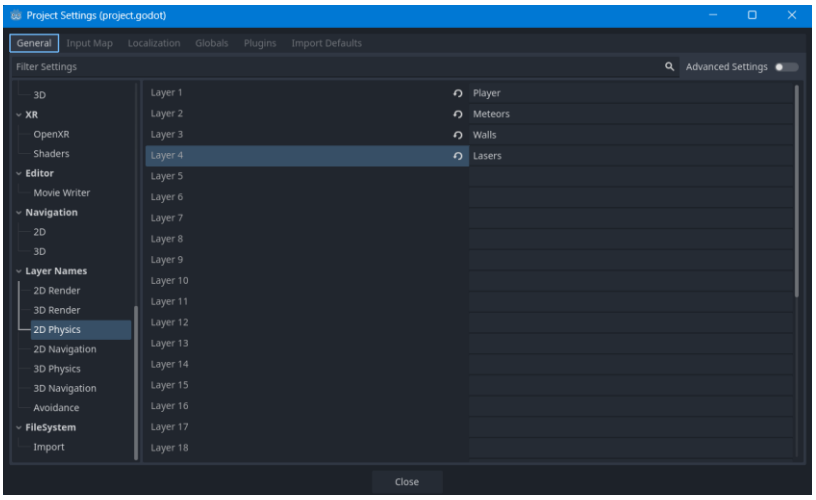
```
# Random Rotation Speed / Speed / Direction
    meteor_speed = rng.randi_range(200,500)
    direction_x = rng.randf_range(-1,1)
    rotation_speed = rng.randi_range(150,250)
# Random meteor graphics
    var path: String = "res://Assets/PNG/Meteors/" + str(rng.randi_range(1,12)) +
".png"
    $MeteorImage.texture = load(path)
```

In the process function:

```
    position += Vector2(direction_x,1.0) * meteor_speed * delta
    rotation_degrees += rotation_speed * delta
```

IMPORTANT STEP FOR COLLISION DETECTION

When working the first time on the project I encountered an issue where I wasn’t able to collide my laser with the meteor. This happened because I did not set the Physics layers. To do this, Head to Project>Project settings>Layer names> Physics2D and Set layer names.



Project Settings (project.godot)

General | Input Map | Localization | Globals | Plugins | Import Defaults

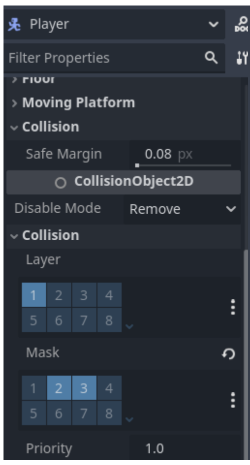
Filter Settings | Advanced Settings

Layer 1	Player
Layer 2	Meteors
Layer 3	Walls
Layer 4	Lasers
Layer 5	
Layer 6	
Layer 7	
Layer 8	
Layer 9	
Layer 10	
Layer 11	
Layer 12	
Layer 13	
Layer 14	
Layer 15	
Layer 16	
Layer 17	
Layer 18	

Close

Now in the Inspector of EACH SCENE we need to set the collision layer to such sequence that:

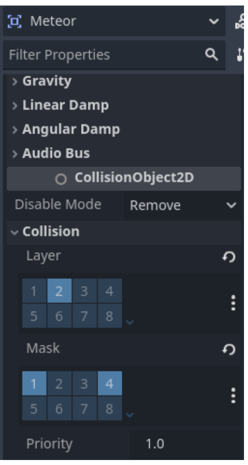
- Player should be on Layer player and should only collide with Meteor Layer and Walls Layer
- Meteor Should be on Layer meteor and should only collide with Player Layer and Laser Layer
- Walls Should be on Layer Walls and should only collide with Player Layer.
- Laser should be on Layer laser and should only collide with Meteor layer.



Player

Filter Properties

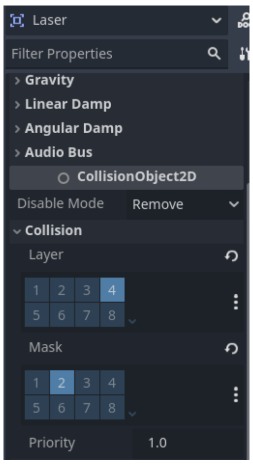
Floor
Moving Platform
Collision
Safe Margin 0.08 px
CollisionObject2D
Disable Mode Remove
Collision
Layer: [✓ 1] 2 3 4 / 5 6 7 8
Mask: 1 [✓ 2] [✓ 3] 4 / 5 6 7 8
Priority 1.0



Meteor

Filter Properties

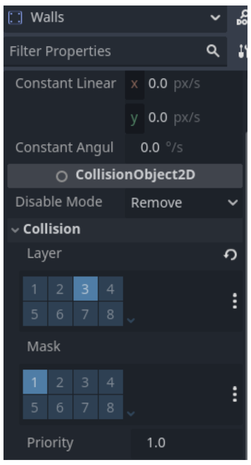
Gravity
Linear Damp
Angular Damp
Audio Bus
CollisionObject2D
Disable Mode Remove
Collision
Layer: 1 [✓ 2] 3 4 / 5 6 7 8
Mask: [✓ 1] 2 3 [✓ 4] / 5 6 7 8
Priority 1.0



Laser

Filter Properties

Gravity
Linear Damp
Angular Damp
Audio Bus
CollisionObject2D
Disable Mode Remove
Collision
Layer: 1 2 3 [✓ 4] / 5 6 7 8
Mask: 1 [✓ 2] 3 4 / 5 6 7 8
Priority 1.0



Walls

Filter Properties

Constant Linear x 0.0 px/s
y 0.0 px/s
Constant Angul 0.0 °/s
CollisionObject2D
Disable Mode Remove
Collision
Layer: 1 2 [✓ 3] 4 / 5 6 7 8
Mask: [✓ 1] 2 3 4 / 5 6 7 8
Priority 1.0

CREATING LASERS WITH CUSTOM SIGNALS

We want to create a custom signal such that when we press shoot in player we need to capture it inside the level Scene.

In Player Scene create a signal called laser

And in process function:

Give condition that if the action shoot is pressed then the signal is emitted:

```
if Input.is_action_just_pressed("Shoot"):
                laser.emit()
```

And in level scene connect it to the script

And inside the signal in the script give some command to confirm that the signal is being captured. For example: print("laser")

Add parameters to the signal for getting the position of the player.

```
signal laser(pos)

laser.emit(pos)
```

and simply print the position in the level function

Now In the Level Scene just like we spawned meteors, we will spawn lasers

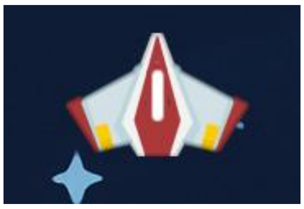
Create a variable that will load laser scene:

```
var laser_scene: PackedScene = load("res://Scenes/laser.tscn")
```

Inside the _on_player_laser:

```
func _on_player_laser(pos) -> void:
        # Creating variable to start the scene
        var laser = laser_scene.instantiate()
        # Adding child node to SpawnLaser Node
        $SpawnLaser.add_child(laser)
        # Update position of the laser to position of the player
        laser.position = pos
```

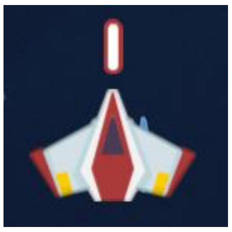
Now that the laser is spawned. Our problem is that the laser is spawning right in the middle of the ship because we have assigned the spawn position to she ship's position.



Now we need to clear this and spawn it little bit up.

Inside Player Scene, create a node called Marker2D called LaserStartPos and set its position above the ship. Now in Player scene instead of **position** update the script to **$LaserStartPos.global_position.**

Problem should be fixed



Now we need to move the laser up after we press shoot. So here when we press shoot the laser is spawned, so that we have already done. Now we need to do so that whenever the laser scene is processing, the laser's position should go up (on negative y axis) with a certain speed. These changes in the script we need to do in the laser script.

```
# Declare a global speed for lasers

@export var speed: int = 1000

# Process after the laser is shot

func _process(delta: float) -> void:

	position.y -= speed * delta
```

DESTROYING METEOR UPON COLLISION WITH LASER

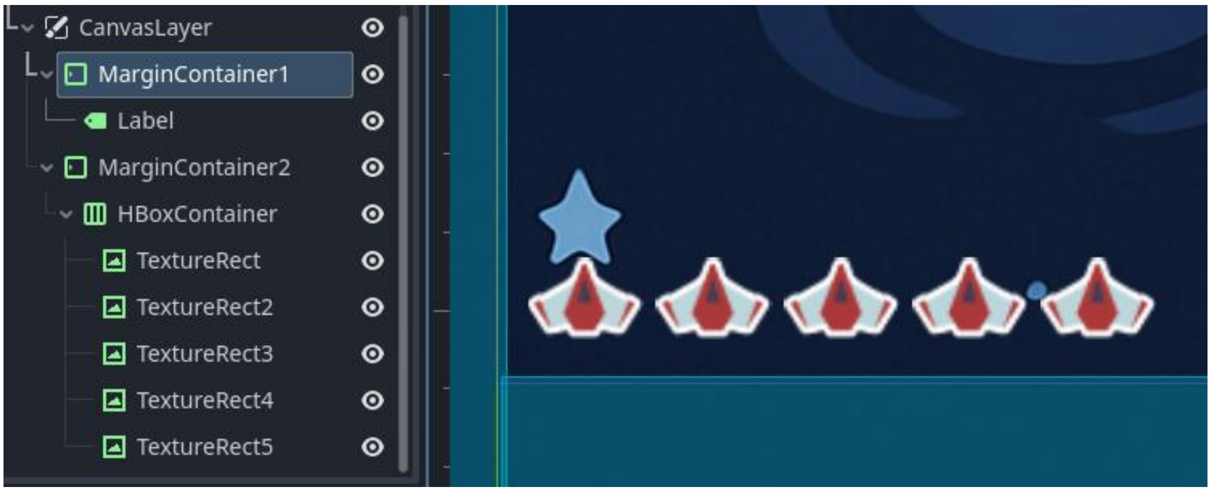
In meteor scene connect a signal Area2D entered and in that script

```
# Signal that reads if Laser(Area2D) is entering the meteor body.

func _on_area_entered(area: Area2D) -> void:

	# Destroying the laser and the meteor

	area.queue_free()

	queue_free()
```

CREATING BASIC HUD: HEALTH, SCORE BORAD AND GAME OVER SCREEN

For adding Text labels to the scene, the formal way to do is to create Canvas Layer Node names UI > Margin Container> Label. Add some text to the label node which will be our score.

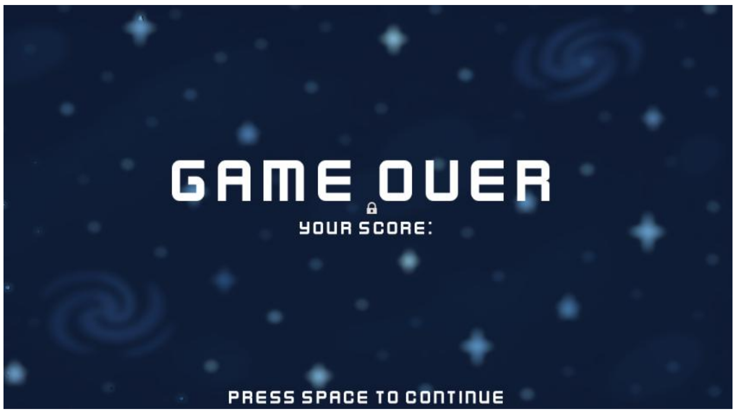
Now we will add health to the Level scene. Add a Margin container to UI and add HBox Container to it as child and further add TextureRect child node to it. Duplicate it 5 times after adding health texture to it from assets.



```
CanvasLayer
  MarginContainer1
    Label
  MarginContainer2
    HBoxContainer
      TextureRect
      TextureRect2
      TextureRect3
      TextureRect4
      TextureRect5
```

Save the UI Branch as a Scene

IN THE SAME MANNER CREATE A GAME OVER SCENE



SCENE TRANSITIONS AND MOVING DATA ACROSS

We need to update data in the UI Scene whenever the player gets hit and we need to transition to Game over screen with the score whenever the player runs out of health.

UPDATING HEALTH

In the level Script we need some kind of function to check the collision between the meteor and the player.

In the meteor script:

```
# Creating a signal to check the collision between player and meteor
signal collision


# Signal created collision function that runs code when a body is collided with meteor.
func _on_body_entered(body: Node2D) -> void:
	# Emit the collision signal when a body is entered
	collision.emit()
```

In the level Script:

```
# Creating a function for collision
func _on_meteor_collision():
	print("meteor collision in level")
```

In meteor time out signal function:

```
# Connecting the signal from meteor script
	meteor.connect("collision",_on_meteor_collision)
```

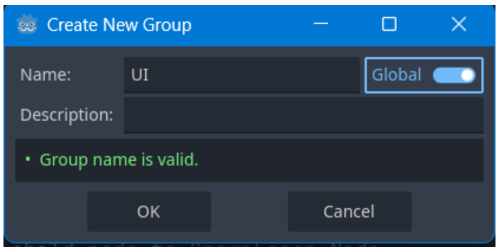
Now we need to update the health of the player every time the collision occurs. For that we need to create a new variable in level called health with int datatype.

```
# Creating variable for health
 var health: int = 5 # Number of health
```

For checking the health, we need to decrease the health every time the collision occurs in the **func _on_meteor_collision():**

```
func _on_meteor_collision():
	health -= 1
	print(health)
	if health <= 0 :
		print("dead")
```

Now the obstacle is that we actually need to update the UI every time the player gets hit. And for that we need to connect the UI scene with the level. This can be done by creating a global group in the inspector of the UI scene. Name the group UI.


Create New Group

Name: UI Global

Description:

• Group name is valid.

Now connect that group to level scene in collision detection function by this script:

```
get_tree().call_group("UI","set_health",health)
```

The arguments in the syntax are functions that we will create in the UI script

UI Script:

Now the logic behind updating health is that when the player will collide, all his health will be deleted and new health will be added depending on what his current health is. Meaning deleting all child nodes upon impact and creating new nodes after impact. This will be in for loop.

For that loading the texture of health image every time we need to create a new variable to store that image. The entire script would look like this:

```
extends CanvasLayer


# Creating a new variable to store health image
var health_image = load("res://Assets/PNG/UI/playerLife2_red.png")


# Create a function to set the health UI
func set_health(amount):
	# Deleting child node in Health upon impact
```

```
    for child in $MarginContainer2/Health.get_children():
        child.queue_free()

    # Creating new children nodes in Health after every impact
    for i in amount:
        var text_rect = TextureRect.new()
        text_rect.texture = health_image
        $MarginContainer2/Health.add_child(text_rect)
        text_rect.stretch_mode = TextureRect.STRETCH_KEEP
```

CHANING TO GAME OVER SCENE

What we need is when health is below zero change to scene game over. That we can do in the level script in collision function by this command:

```
    if health <= 0 :
        get_tree().change_scene_to_file("res://Scenes/game_over.tscn")
```

And for going back to level scene we do the same

In Game Over Script:

```
extends Control

# Creating variable to load the level scene
var level_scene: PackedScene = load("res://Scenes/level.tscn")

func _process(_delta):
    # Taking Input as Space and changing the scene to level
    if Input.is_action_just_pressed("Space"):
        get_tree().change_scene_to_packed(level_scene)
```

CHANGING THE SCORE

For this create a timer node in UI Scene ang connect a signal to it.

Create a node called elapsed_timer with int datatype value 0

And in the timer_timeout signal increase the value of elapsed_timer

And after that write a command calling label in the score container to be updated with the text

Script:

# Creating a variable for time elapsed in the level

var time_elapsed : = 0.

# Updating the score

func _on_score_timer_timeout() -> void:

time_elapsed += 1

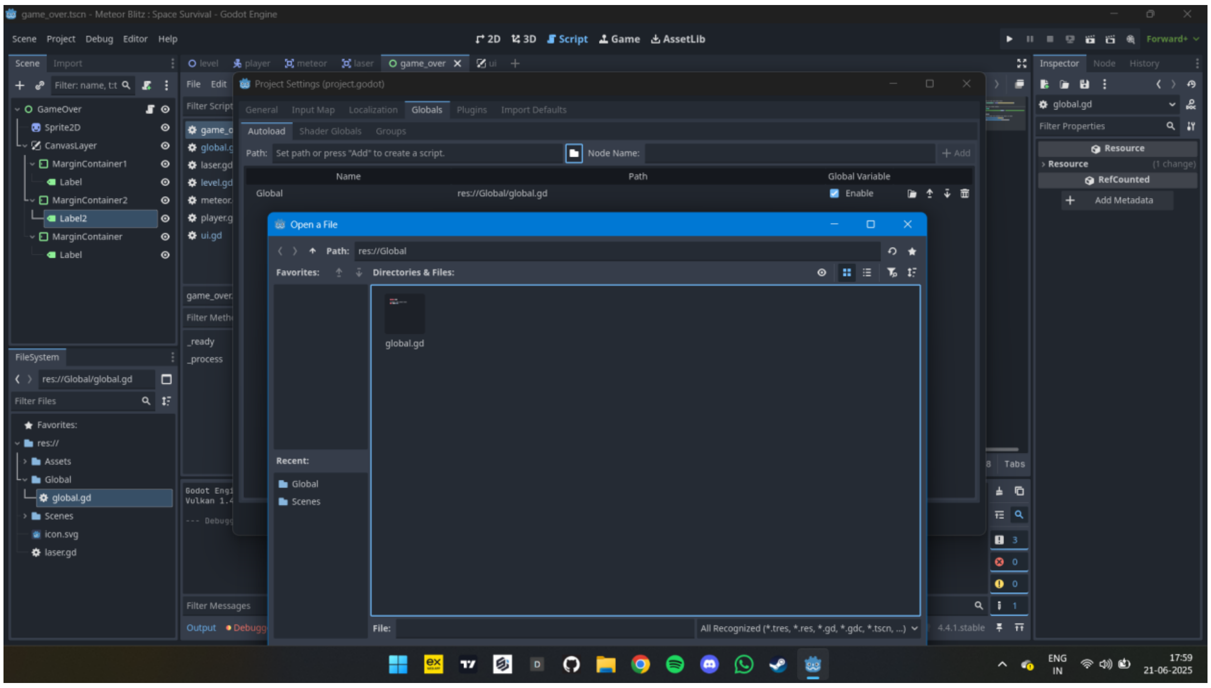
$MarginContainer1/Label.text = str(time_elapsed)

CHANGING THE SCORE IN GAME OVER SCREEN

Step1: Create a new folder called global and create a new script inside it called global.

Step2: In that script create a variable called score and set it to 0

Step3: Inside Project>Project Settings>Autoload. Add the global script as scene. Enable it


game_over.tscn - Meteor Blitz : Space Survival - Godot Engine
Scene Project Debug Editor Help
2D 3D Script Game AssetLib
Project Settings (project.godot)
General Input Map Localization Globals Plugins Import Defaults
Autoload Shader Globals Groups
Path: Set path or press "Add" to create a script. Node Name: + Add
Name Path Global Variable
Global res://Global/global.gd Enable
Open a File
Path: res://Global
Favorites: Directories & Files:
global.gd
Recent:
Global
Scenes
File:
All Recognized (*.tres, *.res, *.gd, *.gdc, *.tscn, ...)
GameOver
Sprite2D
CanvasLayer
MarginContainer1
Label
MarginContainer2
Label2
MarginContainer
Label
FileSystem
res://Global/global.gd
Favorites:
res://
Assets
Global
global.gd
Scenes
icon.svg
laser.gd
Inspector Node History
global.gd
Resource
RefCounted
Add Metadata

Step4: Inside the UI Scene in the time elapsed function set the Global.score = time_elapsed

Step5: Inside the game_over script create a new script that sets the text inside the container for score. To the score global.score

# Setting score when the scene starts

func _ready() -> void:

$CanvasLayer/MarginContainer2/Label2.text = $CanvasLayer/MarginContainer2/Label2.text + str(Global.score)

DONE

ADDING SOUND EFFECTS TO THE GAME

For game music. Add AudioStreamPlayer node to the level scene and then add the audio and select autoplay

For game over sound do the same. Except, instead of selecting autoplay, inside the ready function **name_of_the_audio_player_node.play()**

For Laser, add laser sound node inside the level and in laser spawn script add **LaserSound.play()**

THANK YOU