



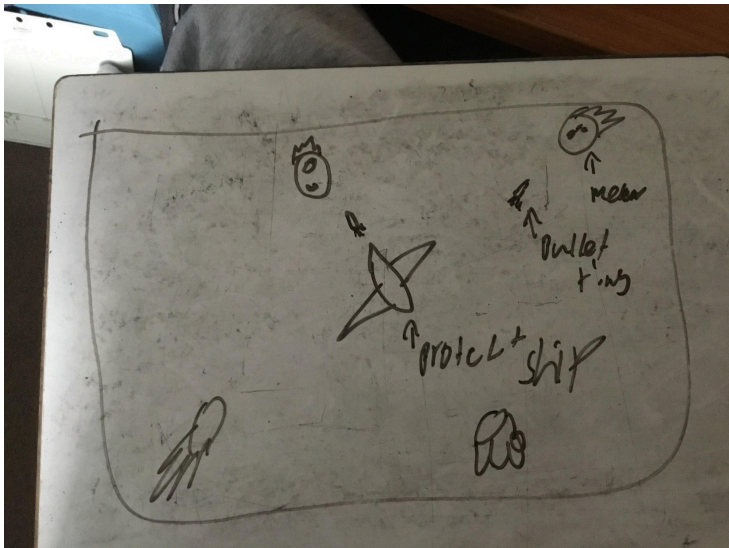
Fire Meteor Game

Game research comparison

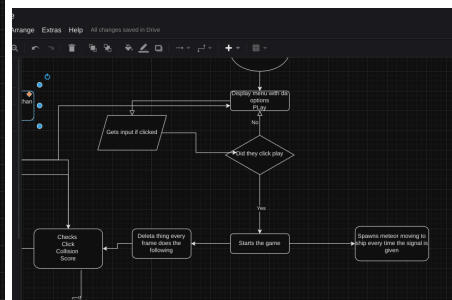
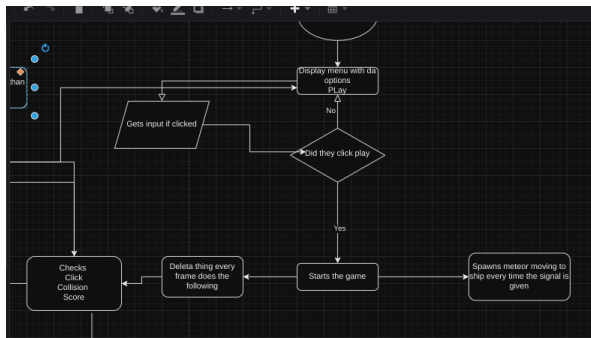
Among Us Meteor Mini-Game	Mario Party Looking For Love Mini-Game
 <p>https://youtu.be/ptVrzbkivbY?si=onYRQUKwlgkAmmY4</p>	 <p>https://youtu.be/gulwhcgoKFg?si=Cl9xlexmtgj_-CLq</p>
The user interacts to complete tasks and eliminate threats.	The user looks in the direction of the heart to score points based on order and correct action.
This game gets a random location on the path2D/ right side of the green screen and spawns a meteor that travels across the screen.	This game randomises from the four different option positions and then sets the heart to that position. Unlike using a mouse as there are only four inputs a joystick is used instead.
If there is a click it puts an image of a target to where the mouse is. It then destroys the meteor and increases the score.	Points would then be recorded remotely to keep track of the score which will check the order of who put the inputs in first.
Meteor interactive minigame task.	Looking For Love fun party game.

Planning

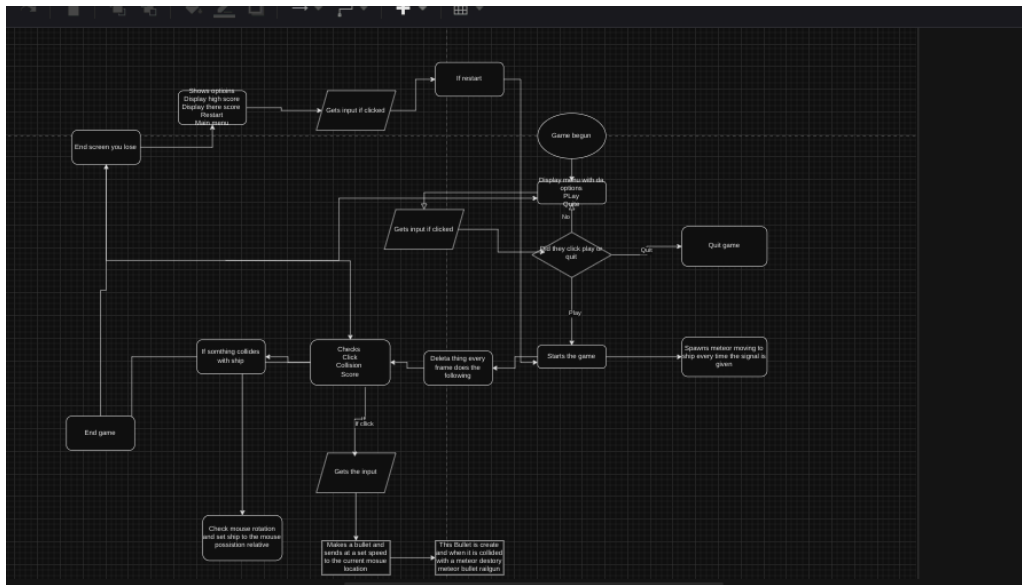
Original designs



After Researching and contrasting both these games the game that was decided was an Asteroid/Meteor game in which the ships have to shoot with their score increasing for how long they survive. As Shown in the diagram above.



The above flow charts plan out a game with a ship in the middle and meteors spawning around it. This is done through a timer signal that spawns meteors based on the signal it emits after you click play. Then once your score reaches a certain point a final boss meteor will spawn and once destroyed the game ends. Due to the limited time available, the boss meteor fight had to be taken out resulting in the below game.



The flowchart above is the final game design. With a game loop when you shoot a Meteor it shoots a bullet at it and destroys it. Increasing your Score the longer you survive. Once they hit you you die and navigate the menu to restart. This game can be done much faster in the limited time given.

This finalised design can be viewed here or accessed on classroom

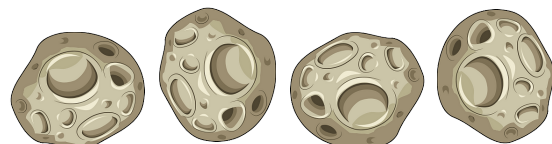
https://drive.google.com/file/d/1AKoKRyr_ddVpX1UCYcuqX_O2K4DB9CQG/view?usp=sharing

Type of Game

This type of mini-game would target anyone into retro space games and enjoy looped gameplay. Though this game could be changed quite easily and altered to fit in the creation of a survival game, Space Simulator is a mini task just like among us and an adventure game as something that the user encounters as some sort of trial. The game's characteristics would fit in general to any age.

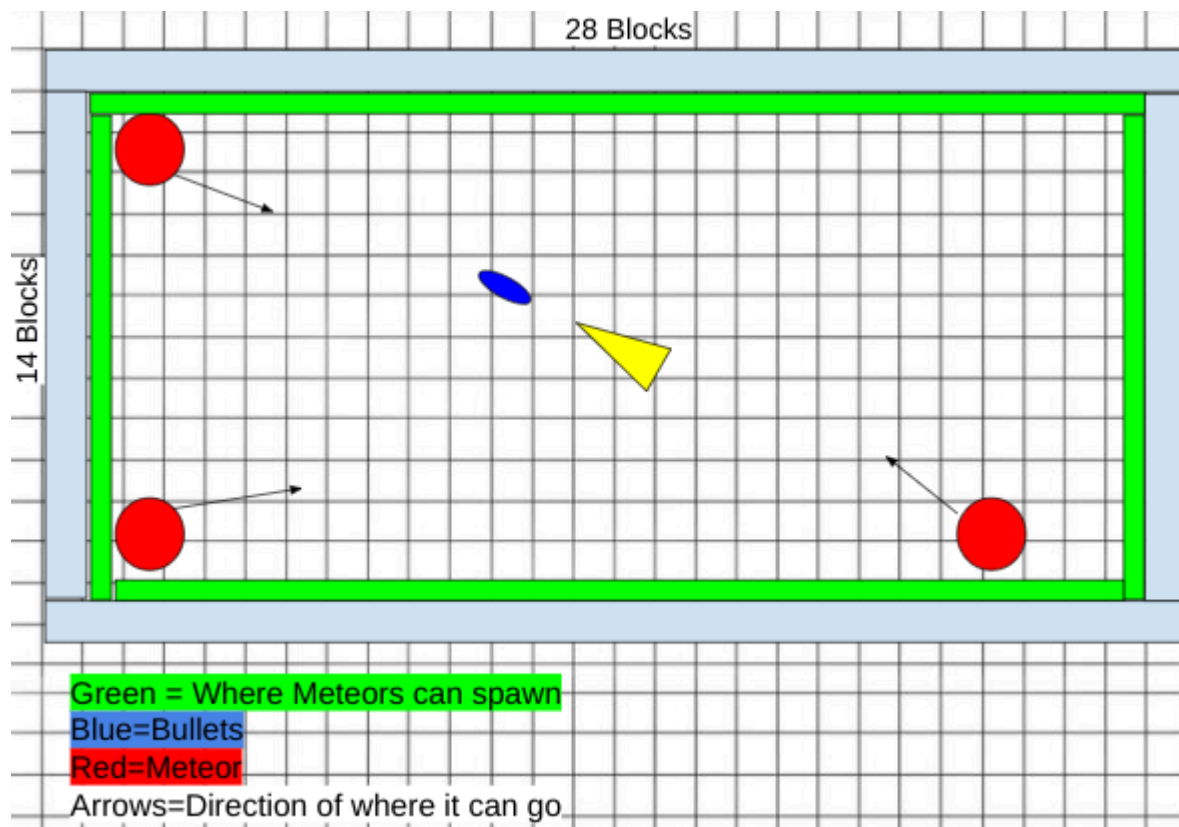
Game Assets

Meteor Sprite animation is its rotation



Menu Image	
Death Image	
Bullet	
Ship	
Main game Background	

Main Game map



The game uses a coordinate system to allow nodes to linearly follow other nodes. A ship is centrally positioned, with meteors spawning in the highlighted green area going to the ship node. Bullet trajectories are linked to the ship's location.

Sprite and Sound Bibliography:

artpictures. club. (n.d.). Edit image, resize image, crop pictures and apply effect to your images. [online] Available at: <https://artpictures.club/autumn-2023.html>. [Accessed March 18 2024]

iStock. (2020). Pixel art star sky at night. Starry sky seamless backdrop. Vector... [online] Available at: <https://www.istockphoto.com/vector/pixel-art-star-sky-at-night-gm1208374725-349251453> [Accessed 18 Mar. 2024].

Wallpapers.com. (n.d.). Download Explore the limitless possibilities of the universe. [online] Available at: <https://wallpapers.com/background/cartoon-space-background-4pttluty0wrv98vc.html> [Accessed 18 Mar. 2024].

itch.io. (n.d.). Asteroids Game Engine by 2indie_dev. [online] Available at: <https://2indie-dev.itch.io/asteroids-game-engine> [Accessed 18 Mar. 2024].

ExilePt (2021). Deep Space. [online] Available at:
https://www.reddit.com/r/PixelArt/comments/odvyko/deep_space/. [Accessed 18 Mar. 2024].

PixaBay (2017). Beam 8. [online] pixabay Available at:
<https://pixabay.com/sound-effects/beam-8-43831/> . [Accessed 23Mar. 2024].

XtremeFreddy (2023). Game Music Loop 3. [online] pixabay. Available at:
<https://pixabay.com/sound-effects/game-music-loop-3-144252/> [Accessed 23 Mar. 2024].

Pixabay (2021). Game Over Arcade. [online] pixabay. Available at:
<https://pixabay.com/sound-effects/game-over-arcade-6435/> [Accessed 23 Mar. 2024].

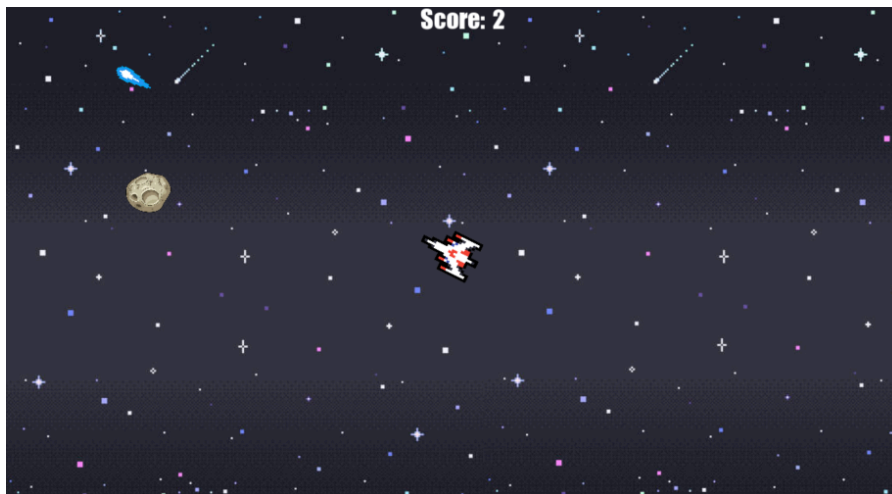
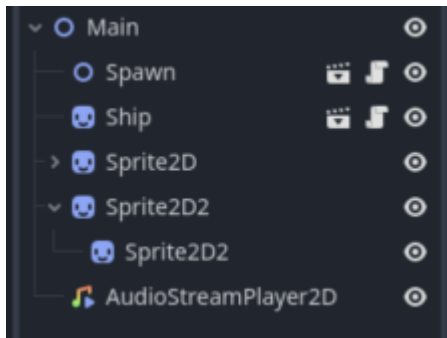
The Final Product: Mini-Game:

Structure

Note not all of the below scenes will be critically explained below

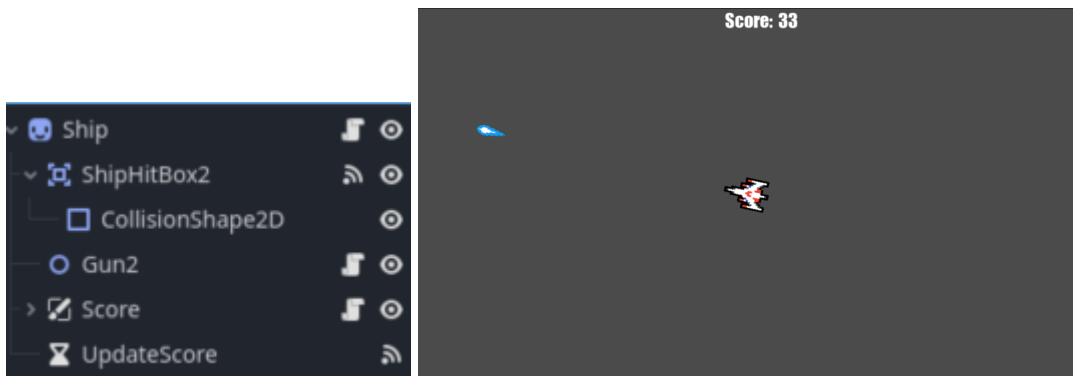
- Main menu - Scripts for input on buttons pressed and music
- Autoloader- Saves different variables
- Main- This is called on the play and has the following
 - The player can shoot meteors and if a meteor hits it sends to the death screen - The player is its own scene that instantiated called ship
 - Music for main scene
 - Spawner which spawns is an instantiated scene called fire which spawns the meteors around a path2D through a path2D follow
 - Makes a background through sprites
- Death scene
 - Allows the player to click the buttons reset to restart the scene or go back to menu
 - Shows high score and the score
 - Plays Woop Woop game over like sound effect
- Player/ship scene
 - Users an area 2d and collision shape 2D to detect collision
 - Has a Gun Which will instantiate a bullet Scene when clicked and go to the mouse
 - Shows a score which updates and on death sends it to the Global AutoLoad code
- Bullet scene
 - Has a collision
 - Sound effect for when instantiated
- Spawner
 - Uses the GD randomise and on a timer it will spawns meteor on the path2d in a random place and goes to the player

The Main Game Scene Structure



The above scene uses a Node2D called Main as its parent, with Spawn and Ship instantiated as they are in other scenes. There is also a background shown above, as well as an audio file that plays the theme music. This does not have code but instantiates other scenes through the navigator.

Player/Ship Scene



The structure of the above uses a sprite as its parent with a gun for shooting the meteors and a Score and timer to display the score.

Ship Code

```
C/C++
using Godot;
using System;
using System.Threading.Tasks; //namespace for task using for the await

public partial class ShipCode : Sprite2D
{
    private Area2D shipHitBox; //Make a variable for the hitbox for shooting

    public override void _Ready()
    {
        shipHitBox = GetNode<Area2D>("ShipHitBox2"); //Actually gets the
hitbox
    }

    public override void _Process(double delta)
    {
        Vector2 mouseGlobalPosition = GetGlobalMousePosition(); //On
double delta gets the mouse position
        LookAt(mouseGlobalPosition); //sets the ship to mouse position
with look at
        if (Input.IsActionJustPressed("click"))
        {
            Immortal(); //calls function immortal which makes the ship
immortal for 0.1 seconds when shooting to avoid a bug
        }
        public async Task Immortal() //Uses the async and the import system
threading for a timer to make the ship immortal as it gets the hitbox info
        {
            shipHitBox.GetNode<CollisionShape2D>("CollisionShape2D").SetDeferred("disabl
ed", true);
            await Task.Delay(100); //awaits for a sec

            shipHitBox.GetNode<CollisionShape2D>("CollisionShape2D").Disabled = false;
            GD.Print("Finish");
        }
    }
}
```

The code retrieves the mouse's location and positions the ship accordingly using the LookAt and GetGlobalMousePosition functions. When firing, it disables the collision to prevent the bullet from destroying the ship. An alternative, like raycasting, could reduce issues with await and enhance performance. However, it was not used due to the increased complexity and potential difficulty for the client to understand.

Gun2 Shooting

```
C/C++
using Godot;
using System;

public partial class Gun : Node2D
{
    int click;
    int max_clicks_per_second = 5;
    [Export] public PackedScene BulletScene;
    [Export] public float BulletsPerSecond = 5f;
    [Export] public float BulletSpeed = 1000f;
    [Export] public float bullet_damage = 30f;
    float _fireRate;
    float time_until_fire = 0f; // Makes a variety of variables some which can
    be edited in the scene drop down

    public override void _Ready() // on ready sets the fire rate to 1/5
    used so bullets cant be spammed
    {
        _fireRate = 1 / BulletsPerSecond;
    }
    public override void _Process(double delta)
    {
        if (Input.IsActionJustPressed("click") && time_until_fire >
        _fireRate)
        {
            click++;
            if (click > max_clicks_per_second)
            {
                GD.Print("bob");
            }
            RigidBody2D bullet =
            BulletScene.Instantiate<RigidBody2D>(); // Gets the Rigid body bullet scene
            bullet.GlobalPosition = new Vector2(576, 324); // sets
            spawn point to the the middle area could also be realtive to sprite 2d
            // or set to a specific angle
            Vector2 mouseGlobalPosition =
            GetGlobalMousePosition(); // Sets its direction to mouse
            bullet.LookAt(mouseGlobalPosition); // Sets direction to
            mouse
            bullet.LinearVelocity =
            bullet.Transform.X * BulletSpeed; // Makes it move at the bullet speed set above
            GetTree().Root.AddChild(bullet); // Adds it so it is
            visible
            time_until_fire = 0f; // allows for shooting again
        }
    }
}
```

```

        }
        else
        {
            time_until_fire +=(float)delta;//for delay
        }
    }
}

```

The Gun2 node Scrip detects clicks and instantiates a scene of a bullet in a controlled amount to avoid spamming bullets. This is done by checking if a click input is pressed and if the time until fire is greater than the fire rate then it allows the bullet to shoot relative to the mouse position. Instead of using this code-implemented timer an additional timer node could have been added but this would require additional nodes and signals decreasing runtime and giving less control over the delay between shots.

Score

```

C/C++
using Godot;
using System;

public partial class Score : CanvasLayer
{
    int score;
    public override void _Ready()
    {
    }

    private void _on_update_score_timeout()
    {
        score += 1;//When the timer emits signal sends the current
        score and sets it to string
        GetNode<Label>("ScoreLabel").Text = score.ToString();
    }
    private void _on_ship_hit_box_2_area_entered(Area2D area)
    {
        var global = (Global)GetNode("/root/Global");
        global.current_score = score;
        score=0;//On death resets score sends to death screen along
        with info
        var newScene = (PackedScene)GD.Load("res://Death.tscn");
        GetTree().ChangeSceneToPacked(newScene);// calls the packed
        scene
    }
}

```

```
}
```

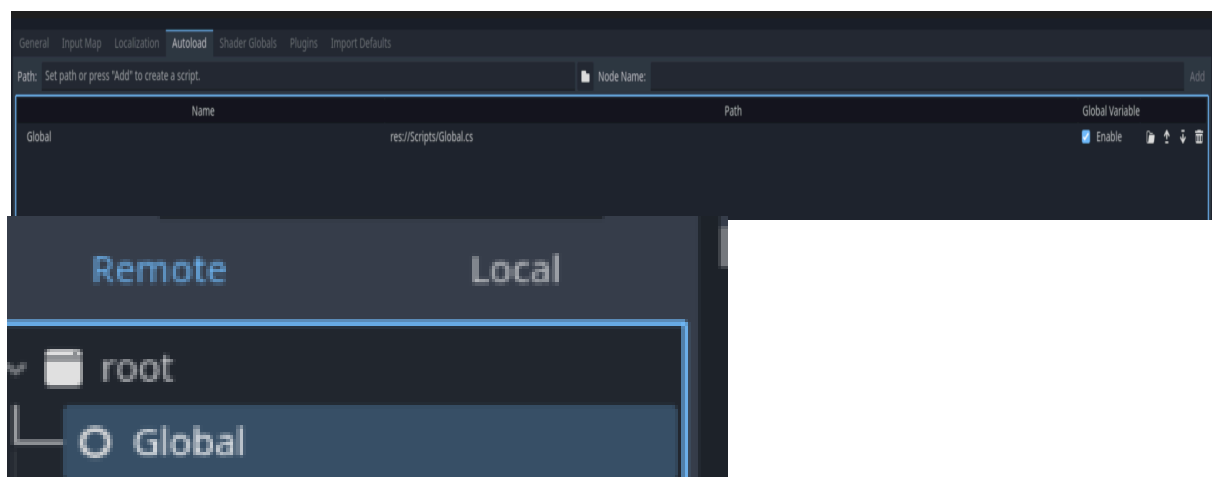
The above code displays the score using a timer to increase it over time. Then once a player gets hit it resets the timer. Though this is not necessary as it resets when a new scene is instantiated this is important for error handling. After it resets the value it sets an autoloader variable to the score to display it across the scene. Unlike the assignment requirements which require a simple game loop this stores information from one scene to another instead of simply displaying the current score. This storing of values could have been in the following way.

C/C++

```
GD.Print(thescore, "The score thing", score);
var newScene = (PackedScene)GD.Load("res://Death.tscn");
var newSceneInstance = (Node)newScene.Instantiate();
newSceneInstance.Set("receivedScore", thescore); // Set the
variable before adding to the scene tree
GetTree().CurrentScene.QueueFree();
GetTree().Root.AddChild(newSceneInstance);
```

This loads a packed scene and sets a public variable in this loaded scene to the score. However, this would result in making a remote scene that would be loaded forever producing possible bugs and an unnecessary amount of loaded scenes.

Auto Load



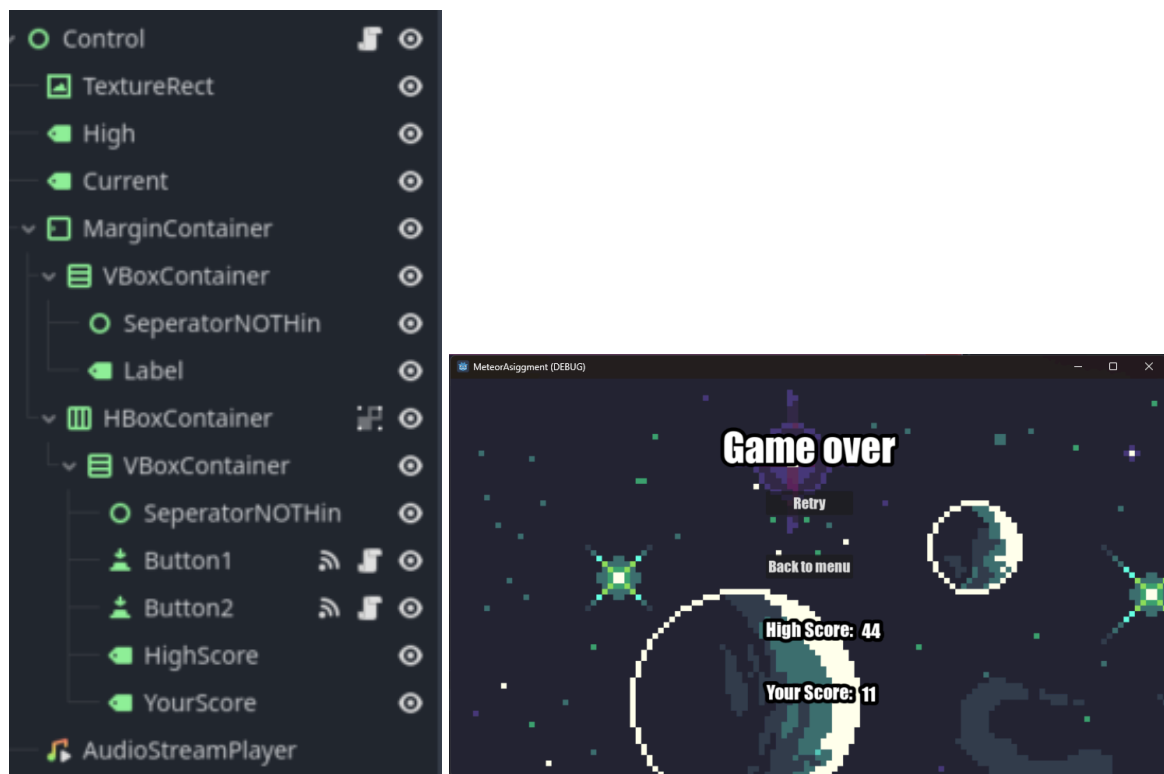
C/C++

```
using Godot ;
using System ;
```

```
public partial class Global : Node
{
    public int high_score = 0;
    public int current_score = 0;
}
```

The score system is run through this autoloader which loads remotely all the time to save values and be accessible to edit.

Death Screen and its use of autoloaders



The above Scene uses GoDots Features to their best with the UI utilising all its settings to allow good and nice visual spacing just like the above Main_menu with the use of buttons to get the user input and the AutoLoader to check the high score and display the current score as well as playing a game over sound when instantiated.

Buttons

```
C/C++
using System;
using Godot;

public partial class Button1 : Button
{
    public override void _Ready()
    {
    }

    private PackedScene click = (PackedScene)GD.Load("res://Click.tscn");
    private void _on_button_down()
    {
        GetTree().ChangeSceneToPacked(click); // calls the packed scene
    }
}
```

The above button code and the others run on a similar code checking if the button is pressed and if it loads a packed scene associated with the button pressed. Instead, a SceneManager could have been implemented to change scenes but this would require additional set-up considering only 3 different scenes need to be loaded not including the audio.

Managing High Score and The score

```
C/C++
using Godot;
using System;

public partial class Death : Control
{
    public override void _Ready()
    {
        // Use CallDeferred to ensure the Label node is ready before accessing it
    }
}
```

```

        CallDeferred(nameof(UpdateScoreLabel));
    }

    private void UpdateScoreLabel()
    {
        // Now it's safe to access the Label node
        var scoreLabel = GetNode<Label>("Current");
        var scoreLabelHigh = GetNode<Label>("High");
        var global = (Global)GetNode("/root/Global");
        scoreLabel.Text = global.current_score.ToString();

        if (global.current_score > global.high_score)
        {
            //checking the gloabls of scens and pritning score to the label
            GD.Print("checking");
            global.high_score = global.current_score;
            scoreLabelHigh.Text = global.high_score.ToString();
        }
        else
        {
            scoreLabelHigh.Text = global.high_score.ToString();
        }
    }
}

```

The code is executed once the ship is destroyed as this is when the scene is instantiated. It checks if the current global score is higher than the high score. If so, it updates and displays the new high score. While the game requirement could have been met by displaying only the current score, including the high score comparison makes the game more competitive and engaging for the audience.

Self-Evaluation

The game Meteor fits the client's requirements well with a clear game loop and a restart button. Though there is no victory as the game is infinite, it does restart with a victory being to beat your old high score. A variety of scenes is used to make the code easy to navigate, as well as comments for the client to understand and use to help implement the features into their desired game.

Unlike what the client required, this game goes beyond how it can be implemented from a simple Among Us-like game to how the game is laid out, being able to be added to a broader spectrum of survival games as a mini-task. With autoloaders to send different information across scenes, the game also takes into account error handling with nulls being returned if the code breaks.

However, the mini-game does have its flaws, as well as doing things beyond what the client first required. It allows for an interactive experience, but some things could be improved, such as bug handling, as there may be hidden bugs that are harder to notice. As well as optimization, due to the many calculations that occur during the `_process` function/every frame. Features could be expanded with additions such as power-ups and a variety of difficulties for the player. The Gun2 Node script set up variables for the client with these potential power-ups in mind, but time took a toll on such implementations. By incorporating these elements, the game loop would have felt more interactive for the player and more appealing to the client.