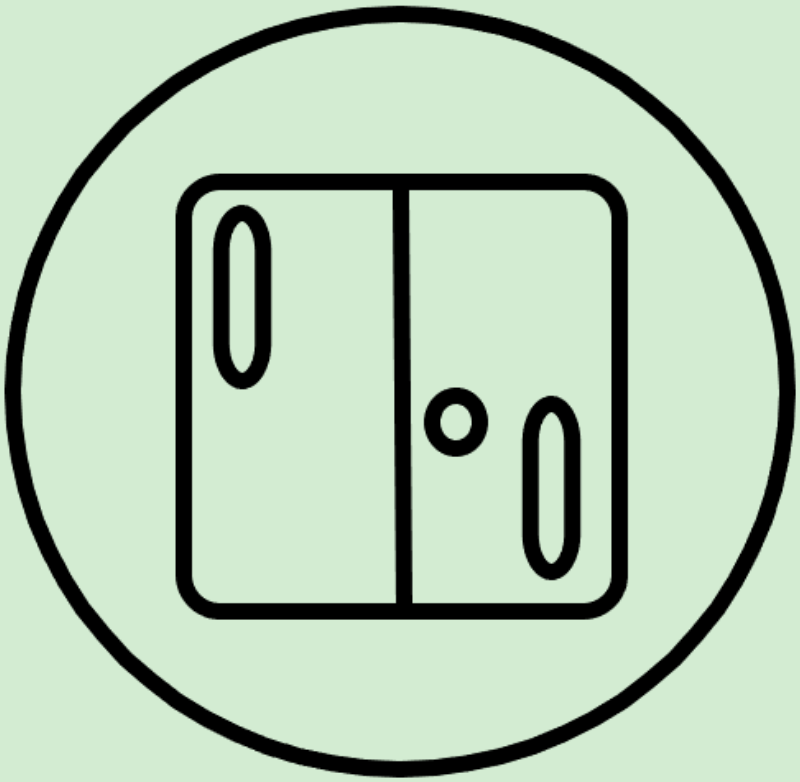


PONGO

GAME BY: AHMED M KARIM

DOCUMENTATION

PROJECT MANAGEMENT REPORT
&
EXPLANATORY NOTES

Table of Contents:

Defining

And

Understanding

The Problem

Problem:

The pandemic caused by COVID-19 took us all by storm. The world has collectively become fragile and has had a deep impact on the way we perceive our world and everyday lives. The economic and social disorder which was brought to the table due to this pandemic led millions of people into severe poverty and caused a sense of discomfort between individuals all around the world. Safety measures have been taken into consideration to protect everyone from harm's way physically by social distancing and refraining from contact. Online learning and work have been put into place to further lessen the contact between people as well. Due to most people attending work or school from home, they have gained extra time to spend on their casual hobbies. The gaming community has grown significantly over the past year.

Nowadays, people have started playing games online such as Grand Theft Auto, Fortnite, League of Legends, Valorant and CSGO. New games have started to grow exponentially and developers are working harder than ever to create even better games with higher production and graphical enhancements. This increase in popularity and demand has created an issue of disconnection from old-school classic games such as Pacman, Snake and Minesweeper.

Simpler games are becoming less popular over time. The issue with this is that the brain is being more pressurised to follow the complex rules and functions in order to play complicated new games. The beauty in simplicity is slowly but surely being forgotten over time.

Solution:

The intention of this project is to solve two issues:

Revive a ‘forgotten’ classic game known as ‘Pong’

Today’s generation is flooded with high-quality immersive three-dimensional video games alongside carefully constructed two-dimensional games available on all platforms (PC, Xbox, Playstation, Nintendo Switch, Android, IOS, etc). Companies are pouring huge amounts of money into developing and advertising these games extensively. Unfortunately the downside of having these innovations and a seemingly endless collection of games is that their ancestors are being forgotten or unappreciated.

'Pong' is a table tennis-themed twitch arcade sports video game with rudimentary two-dimensional visuals that was first launched in 1972. The Pong arcade games produced by Atari were a massive success. It was the first commercially successful video game, and it contributed to the development of the video game industry. Several companies began making games that closely resembled its gameplay soon after its debut.

My goal is to revive the game that was a necessary creation for the booming industry of gaming and the start of a new technologically advancing era- ‘Pong’

Make a game suitable for casual play which **relieves** the brain from stress

Simplicity is the main theme of this project. This remastered version of 'Pong' which I call 'Pongo' includes an aesthetic glowing colour palette and design along with fitting music in order to appeal to the modern community who enjoy playing games with trendy designs such as Rocket League. I have chosen to remaster this specific title because I enjoyed playing it when I was younger and established an emotional connection with it.

Modern games are extremely complex. They have missions, side quests, achievements, pointing systems, in-game currency, time limits and so much more. My goal was to create a simple game that would ease the player's brain from all this unnecessary data processing.

The colour palette consists of black, green and a little bit of white. The use of black all across the menus makes my game consistent and makes it visually less eye-straining. The minimalist menus were created to be simple but convenient at the same time. The use of only 2 large buttons on each menu makes it easy for the user to decide what to do. Convenience can be seen in the mute button at the top right corner on the main menu so that the people who do not like the music can get rid of it straight away.

The music primarily uses a classical guitar and is called 'Fiesta de Guerra' made by Nikawa Kayasaki. I have used it to convey my personal attachment to this project, as it is one of my all time favourites. Personally, the music is astonishing and serves the brain well.

This project is also a creation necessary for my school HSC major assignment.

Identification of Development approach - Rapid application development

The development approach of this project is the **RAD (Rapid Application Development)** approach which prioritises developing applications rapidly through frequent iterations and continuous feedback. In simpler terms, various versions of the game are built and tested in a remarkably short period of time repeatedly until a suitable program has been created. The RAD approach focuses on building **small scale** applications/games in a **short period** of time with a **low budget**. In this approach, the client is often a small business and the developer an individual, in this case, a teacher and a student.

The advantages of a student using the RAD approach include:

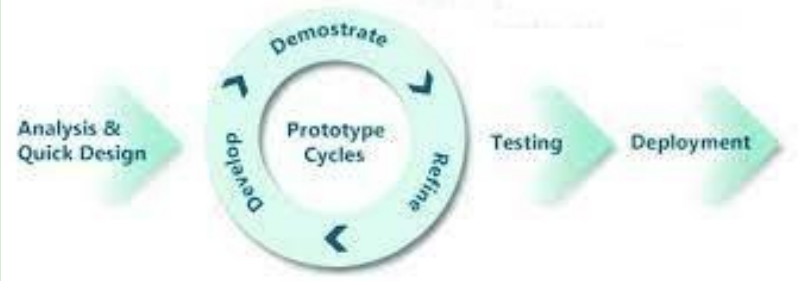
- Enhanced flexibility and adaptability as adjustments can be made quickly during the development process.
    - The project can be adapted to fit certain needs
- Lack of formal stages
    - Stages might be skipped or rushed due to the tight time frame
- Requirements can be changed at any time
    - Allows flexibility
- Works well with a small number of developers
    - An individual in this case
- Quick iterations reduce the development period.
    - More time to make documentation and focus on other subjects simultaneously
- Encouragement of code reuse means less manual coding, less room for errors, and briefer testing times.
    - Since I am a beginner in programming, this will make my development much smoother
- Increased customer satisfaction due to a high level of collaboration and coordination between developers, clients and end-users.
    - Asking the teacher or classmates for feedback

The disadvantages I will tackle as a student using the RAD approach include:

- Cannot work with large teams
    - Solution: I am an individual, therefore I am the smallest 'team' possible
- Needs highly skilled developers
    - I am creating a simple game with little skill necessary
- Only suitable for projects which have a small development time
    - I have a small time to work on this project, so it fits perfectly
- Only systems which can be modularised can be developed using Rapid application development.
    - As an individual, I will have to work on everything myself so it is not necessary to modularise

The reason I chose this specific approach lies within the definition of the RAD approach. The main aim of the RAD approach is to create a usable software version in the shortest time at the lowest cost to the client. Since I am a graduating student with limited experience in the programming field and my software teacher is the recipient of this game who is not paying me to build this game- this project has time constraints due to the existence of other assignments and studies, restricted resources due to the lack of design and development teams and no funds. This approach appeals to individual developers and encourages the use of existing code which is why it is the most suitable for me.

RAD approach diagram:

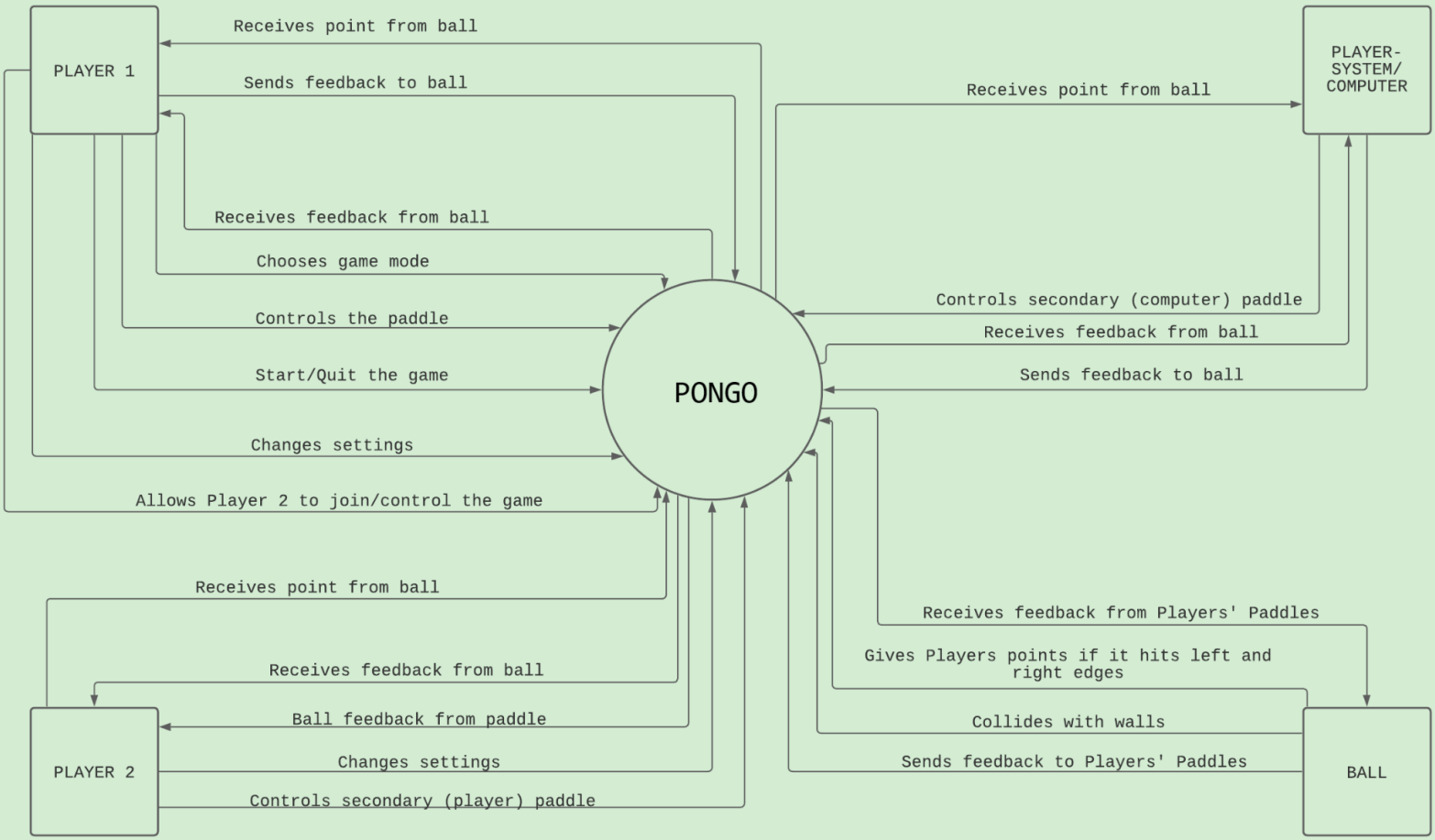


Reference used to determine whether this approach would suit the development process:

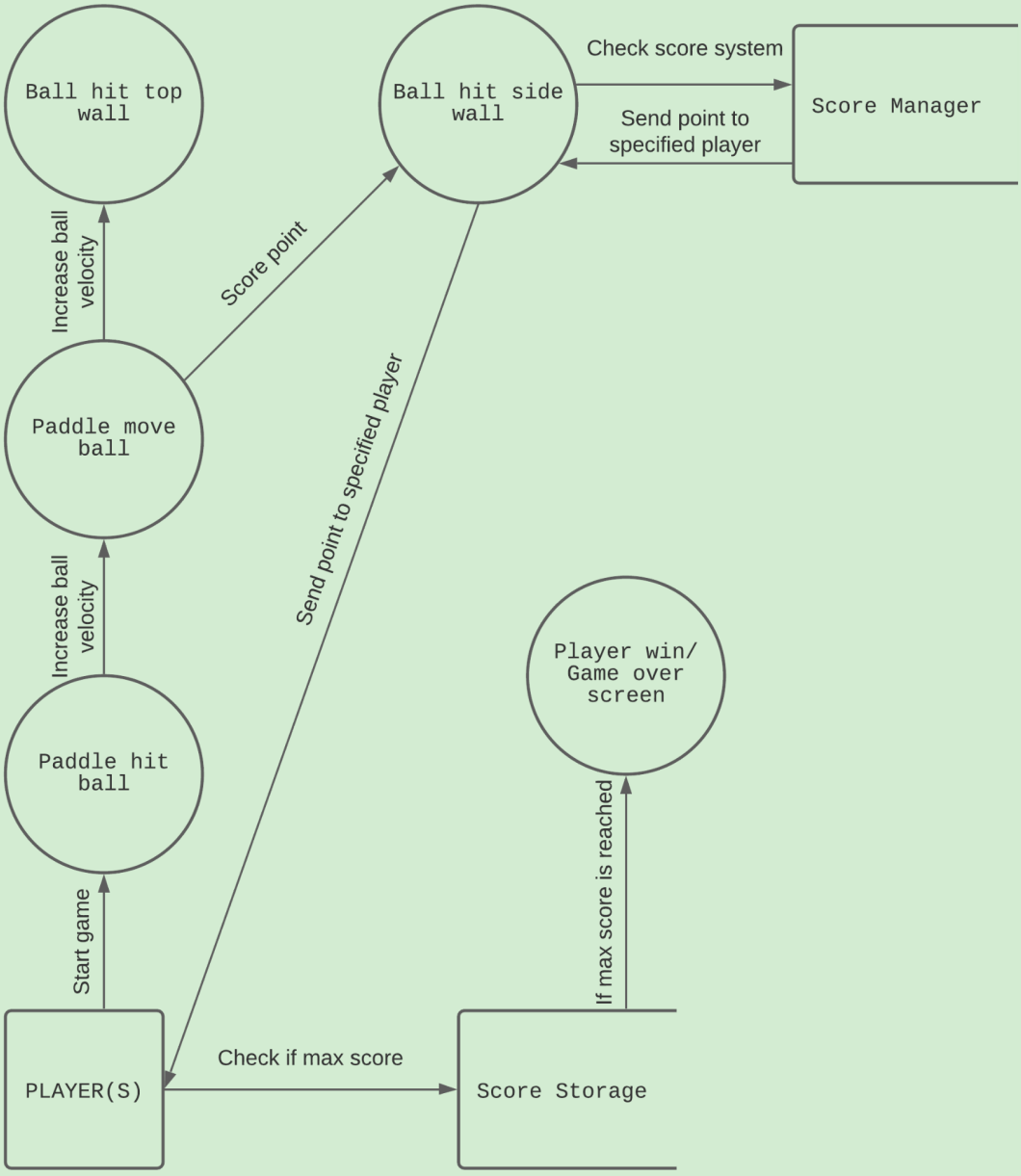
Advantages of RAD	**Disadvantages of RAD**
Requirements can be changed at any time	Needs strong team collaboration
Encourages and priorities customer feedback	Cannot work with large teams
Reviews are quick	Needs highly skilled developers
Development time is drastically reduced	Needs user requirement throughout the life cycle of the product
More productivity with fewer people	Only suitable for projects which have a small development time
Time between prototypes and iterations is short	More complex to manage when compared to other models
Integration isn't a problem, since it integrates from project inception	Only systems which can be modularised can be developed using Rapid application development.

By: Ahmed Karim

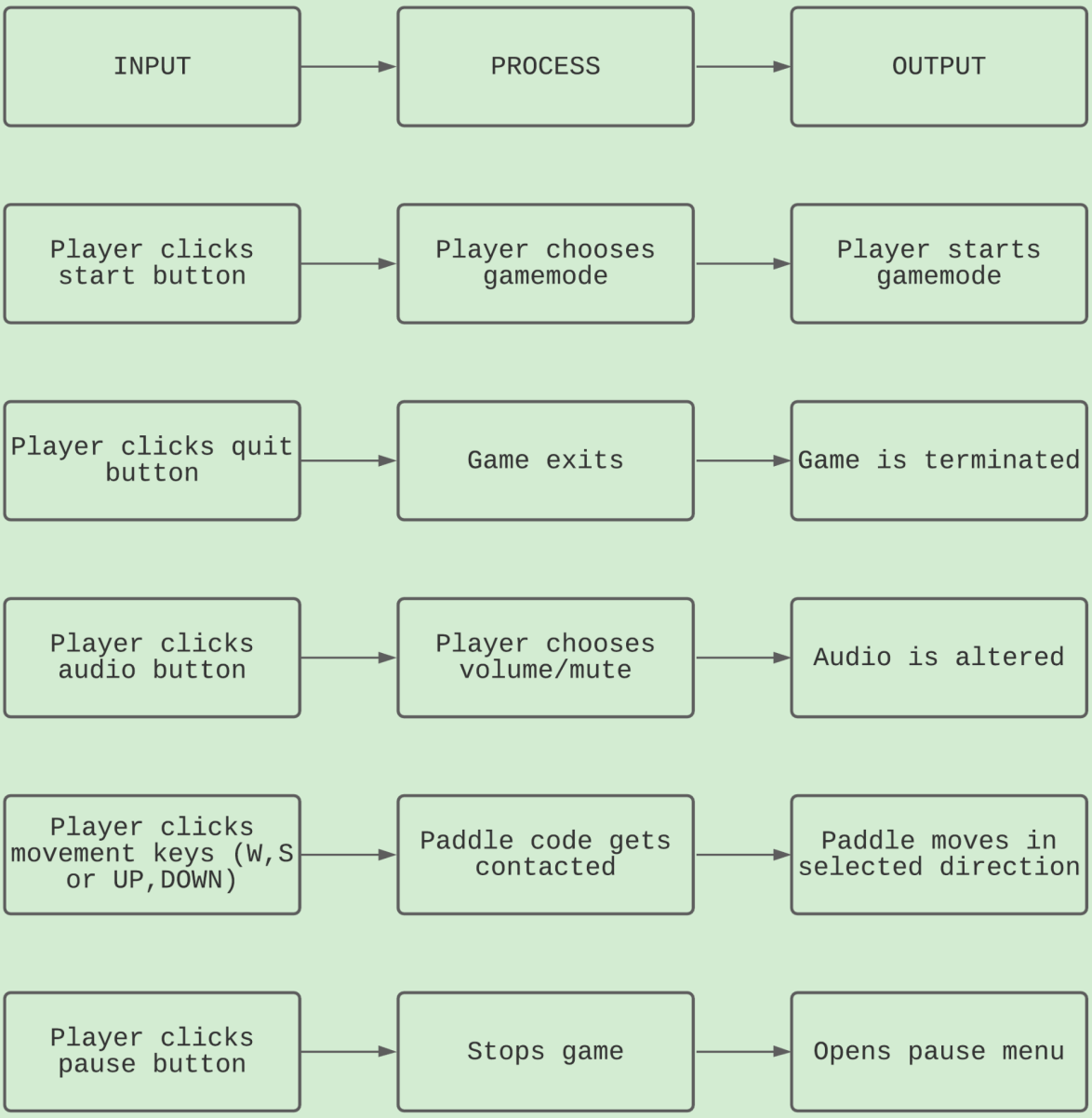
Identification of Input processing and output - Context diagram



Identification of Input processing and output - Data flow diagram



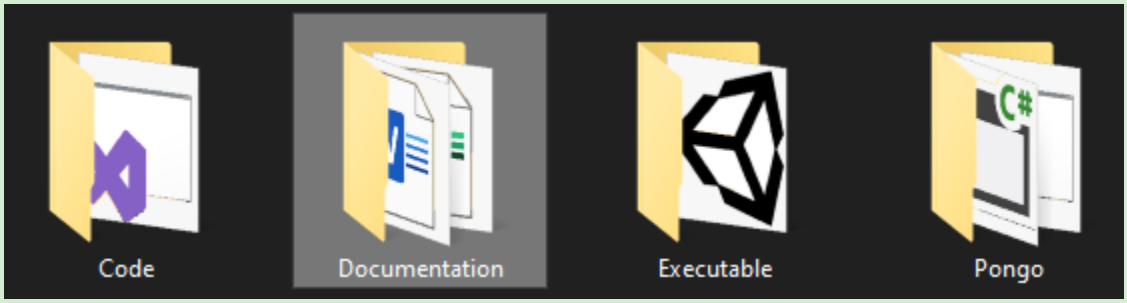
Identification of Input processing and output - IPO diagram



By: Ahmed Karim

Project Management - Gantt chart

Can be seen in the 'Documentation' Folder



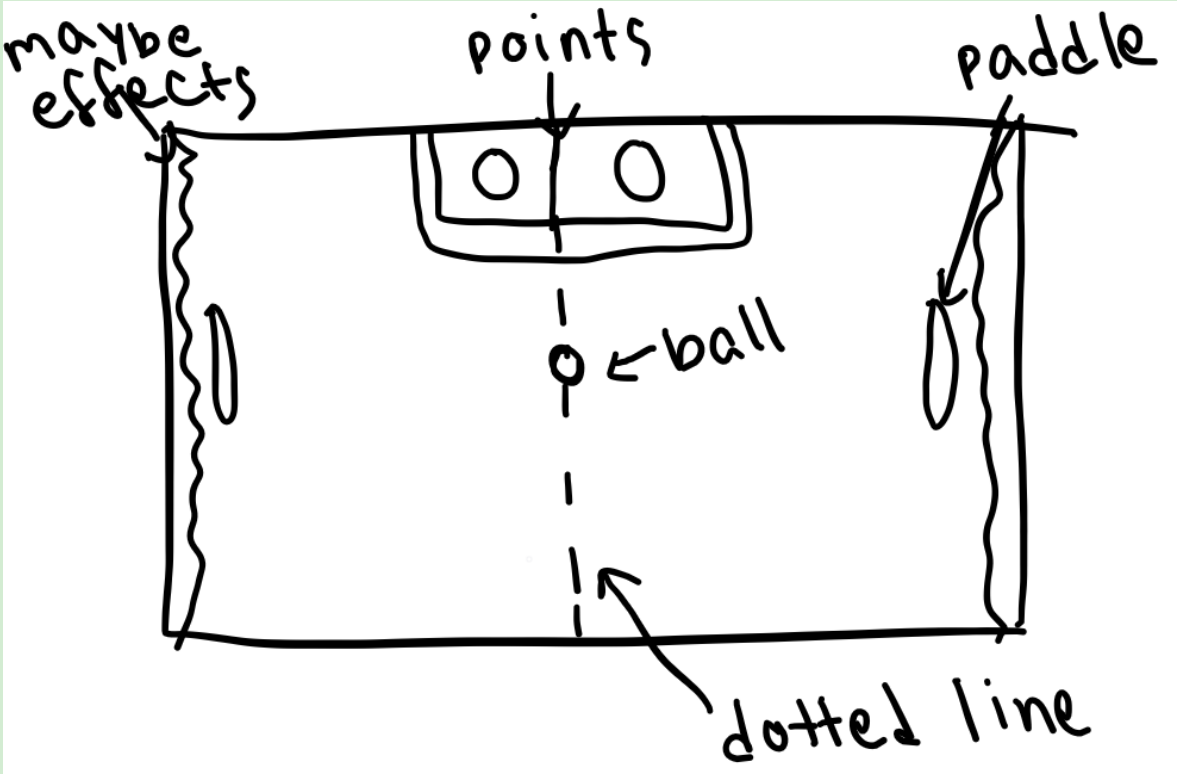
TASKS:	Week 7	Week 8	Week 9	Week 10	Holiday W1	Holiday W2	Holiday W3	Holiday W4	Holiday W5	Holiday W6	Holiday W7	Holiday W8
Define & understand the problem												
Plan idea												
Draw IPO diagram												
Draw data flow diagram												
Draw context diagram												
Identify development approach												
Justify development approach												
Start creating game												
Create and find resources												
Write out and explain all code in doc												
Music Creation												
Coding game												
UI and project explanation												
Evaluation												
Learn coding method												
Intrinsic documentation												
Documentation												
Gantt chart												
KEY FOR TIMEFRAMES:												
Unsuccessful (missed out completely)												
Unplanned (but done at this time)												
Successful (done at planned time)												

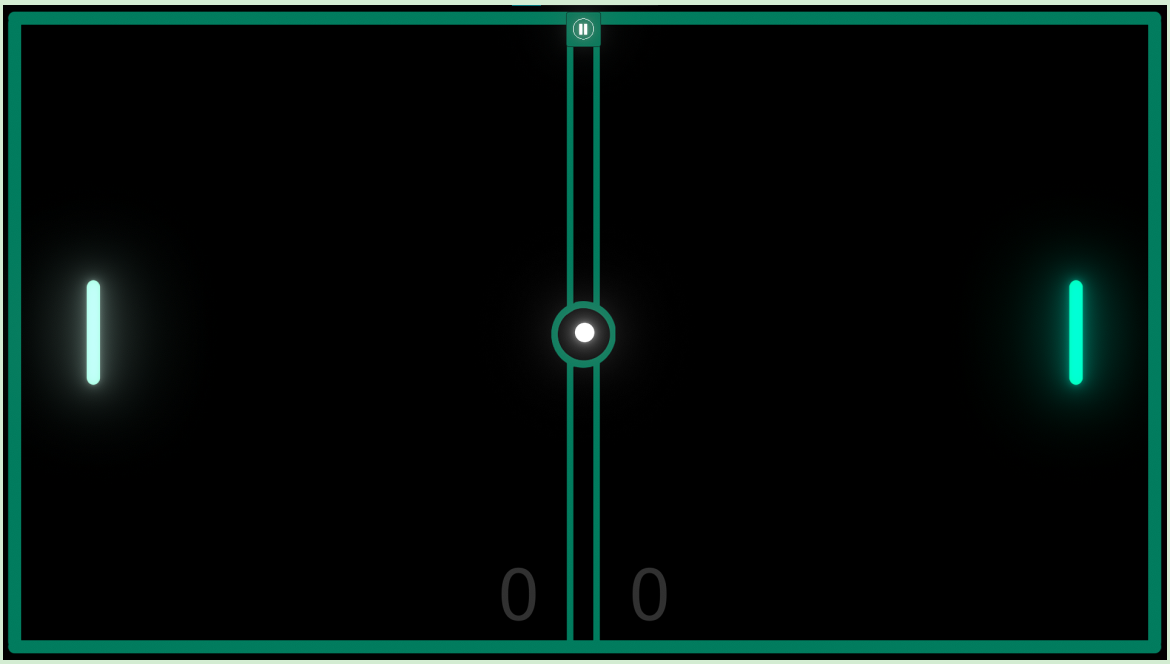
Explanation:

- Unsuccessful- For various reasons such as study or personal activities, I was unable to do the tasks in these planned times
- Unplanned- I did not intend to do any tasks in these time frames but ended up doing it due to preference/convenience
- Successful- The tasks were completed at the planned times

Design and Function

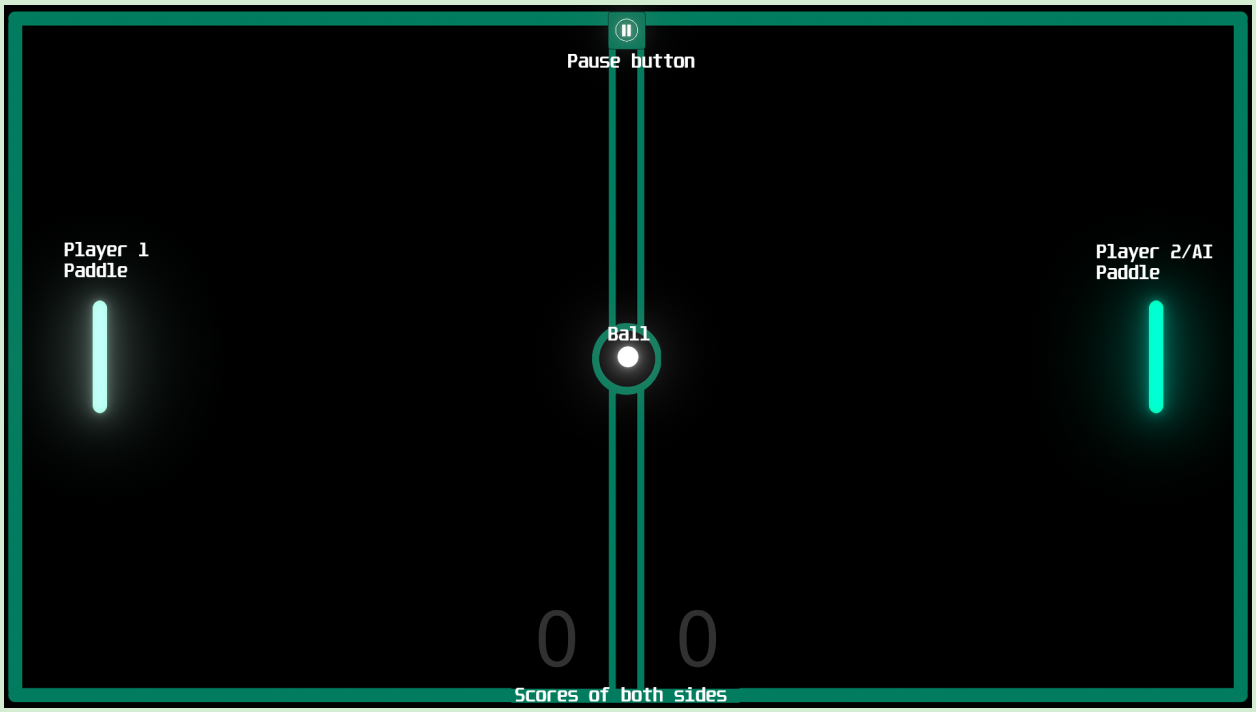
Main game screen draft vs end product:



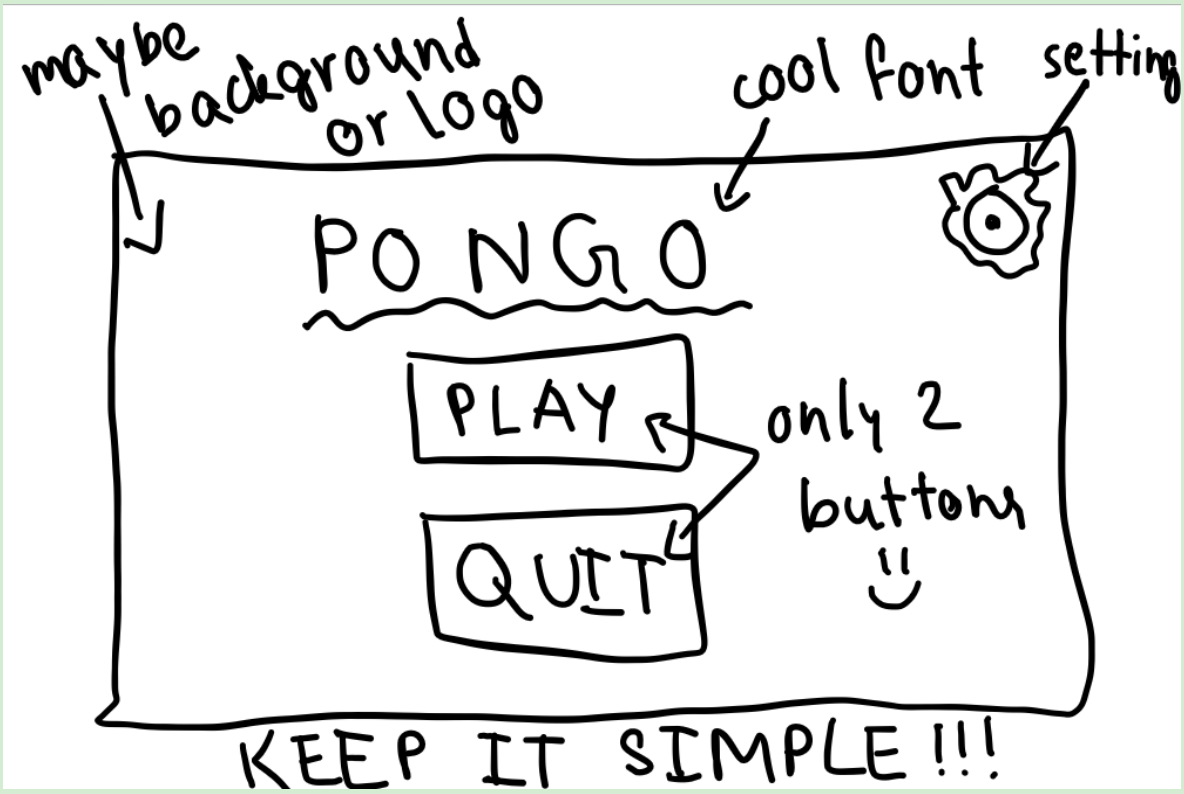


By: Ahmed Karim

Game screen with explanation:



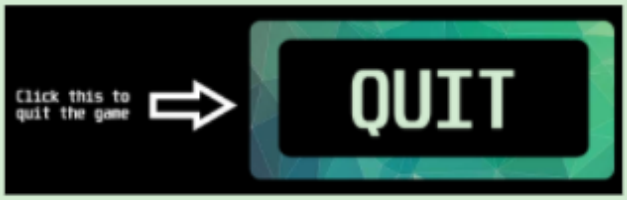
Main menu draft vs end product:




Audio
PONGO
Mute
START
QUIT

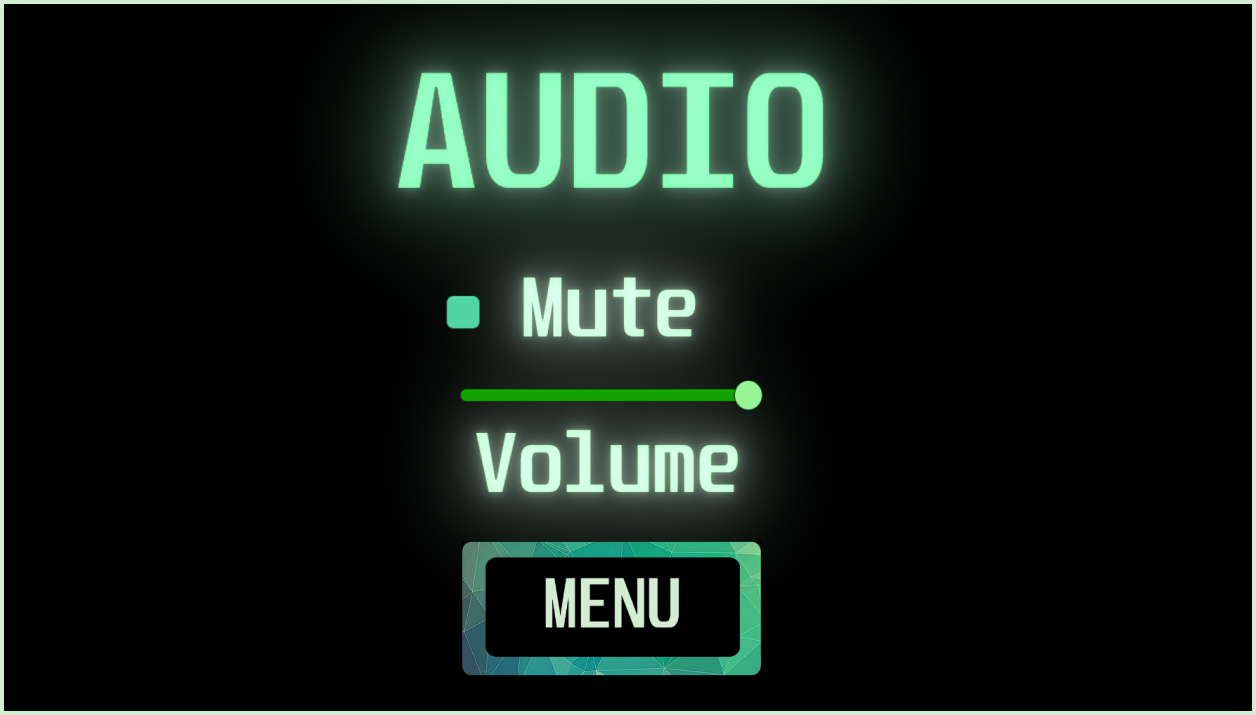

Audio
Click this to
open audio menu

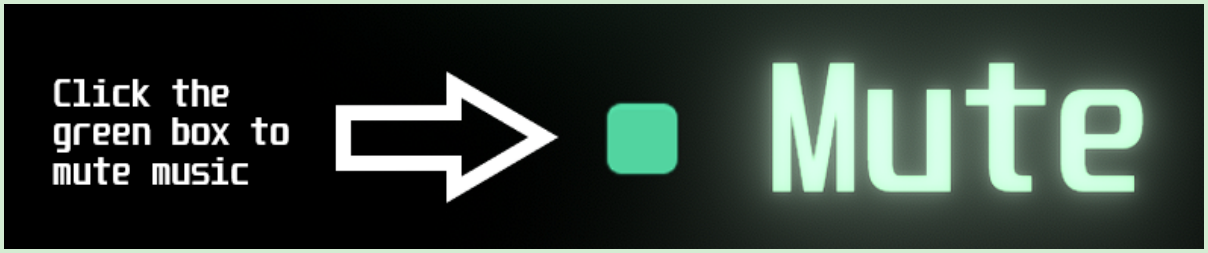

Mute
Click this to
quick-mute

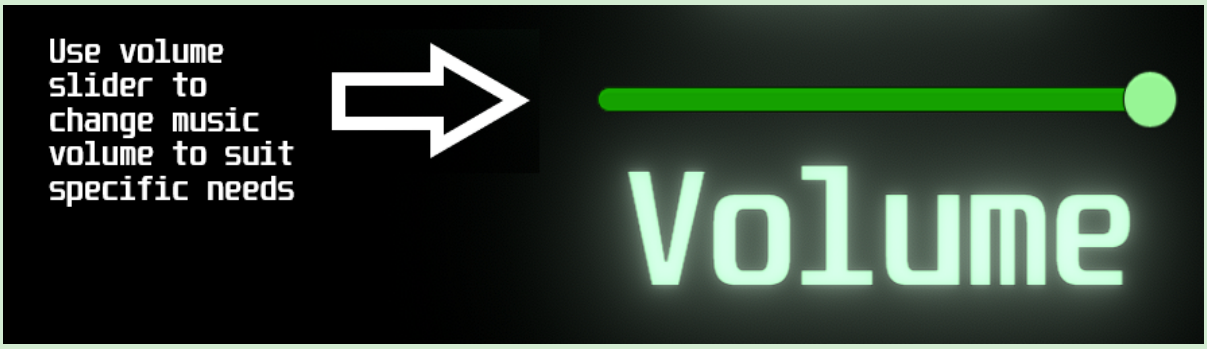

Click this to
quit the game
QUIT


Click this to
proceed to the
next screen
and choose
your gamemode
START

Audio menu:

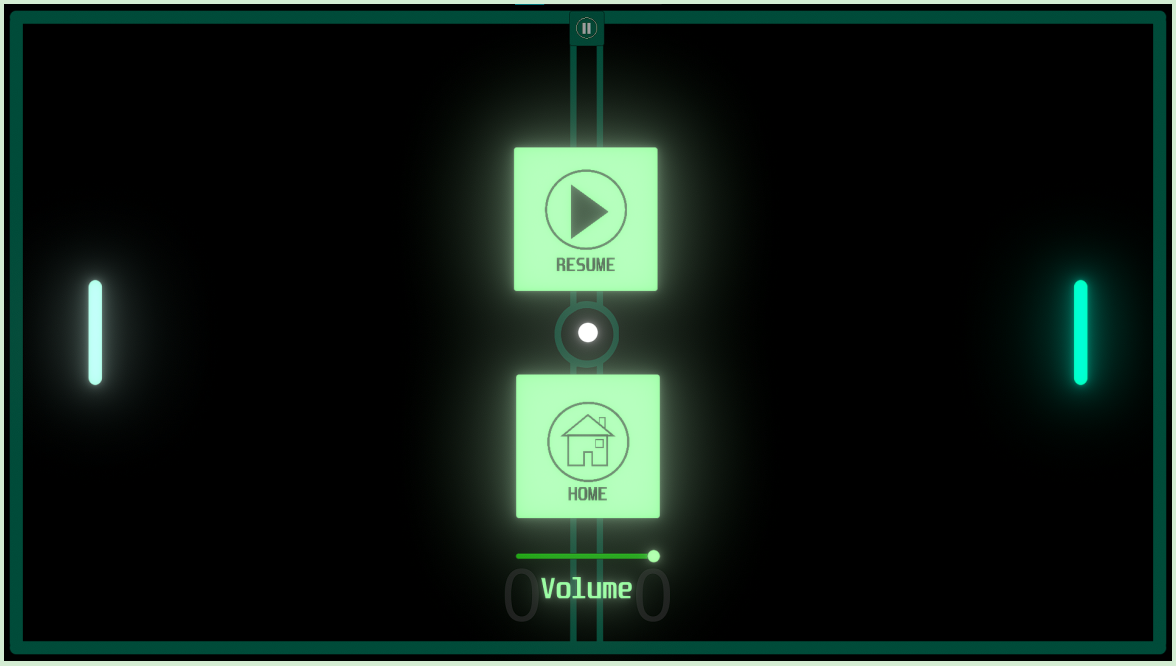

AUDIO
Mute
Volume
MENU

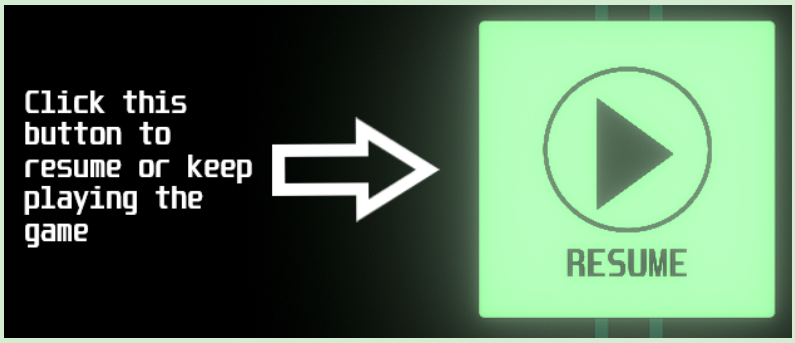

Click the green box to mute music
Mute

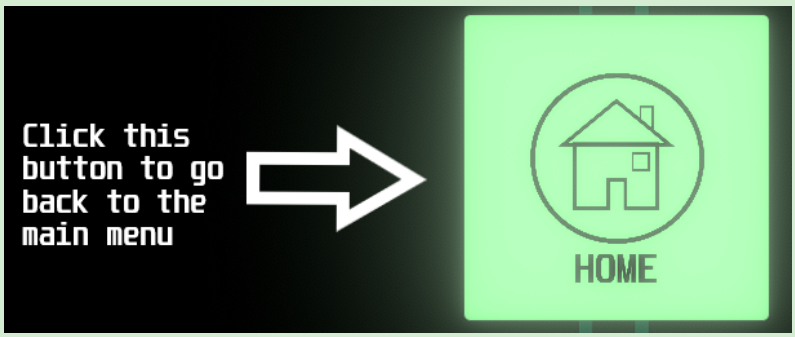

Use volume slider to change music volume to suit specific needs
Volume


Click this button to return to main menu
MENU

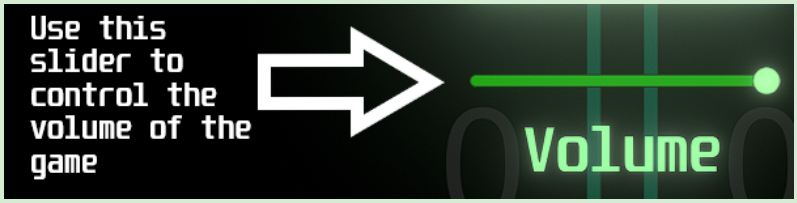
Pause menu:


RESUME
HOME
Volume


Click this button to resume or keep playing the game
RESUME

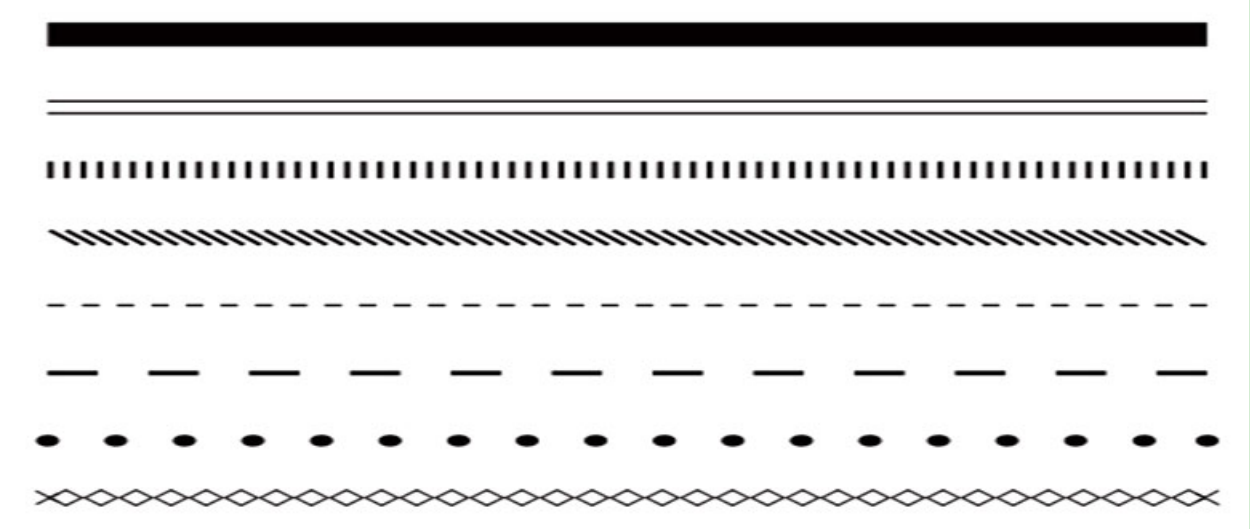

Click this button to go back to the main menu
HOME


Pause menu can be accessed by clicking this button

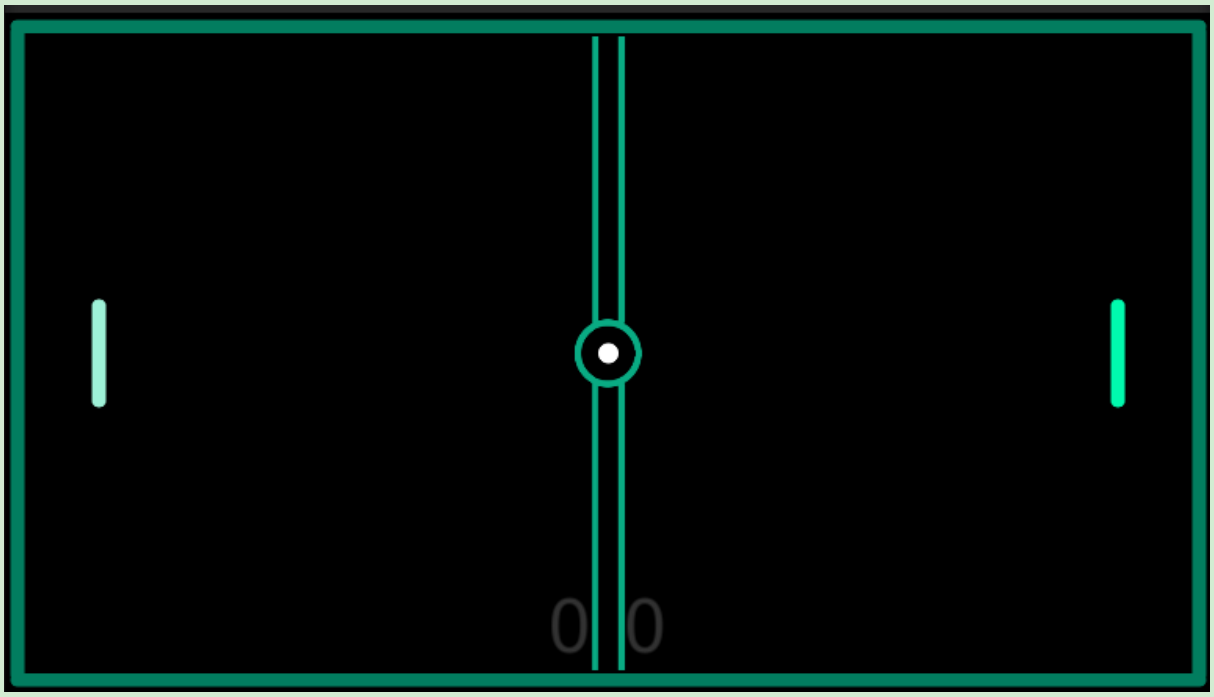

Use this slider to control the volume of the game
Volume

Testing with different names and designs

Centre line inspiration:



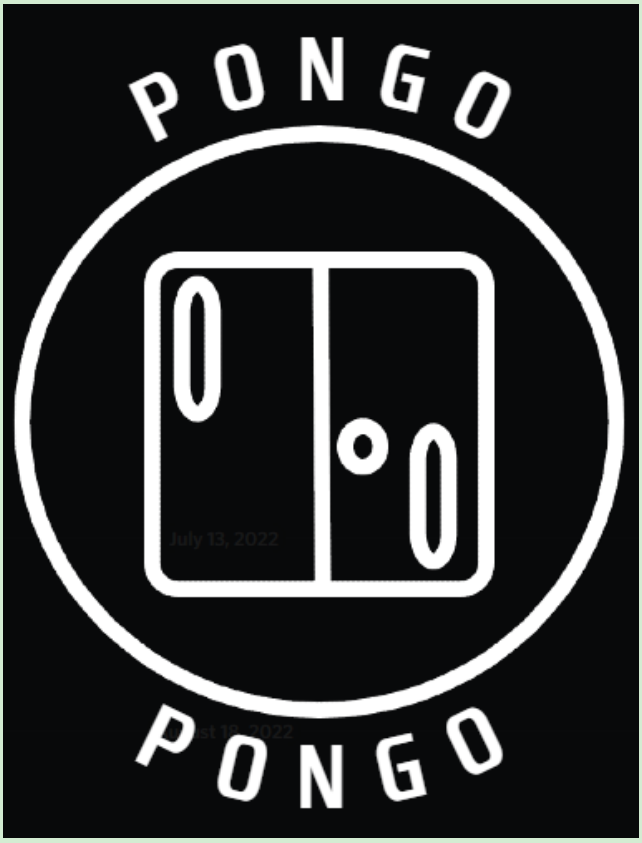
Pongo without the glow effect:

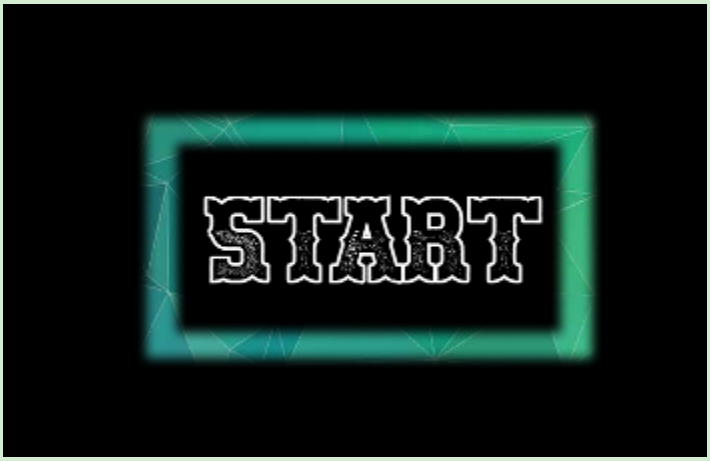


Different name and colour


GAME OVER :(
RETRY
MENU

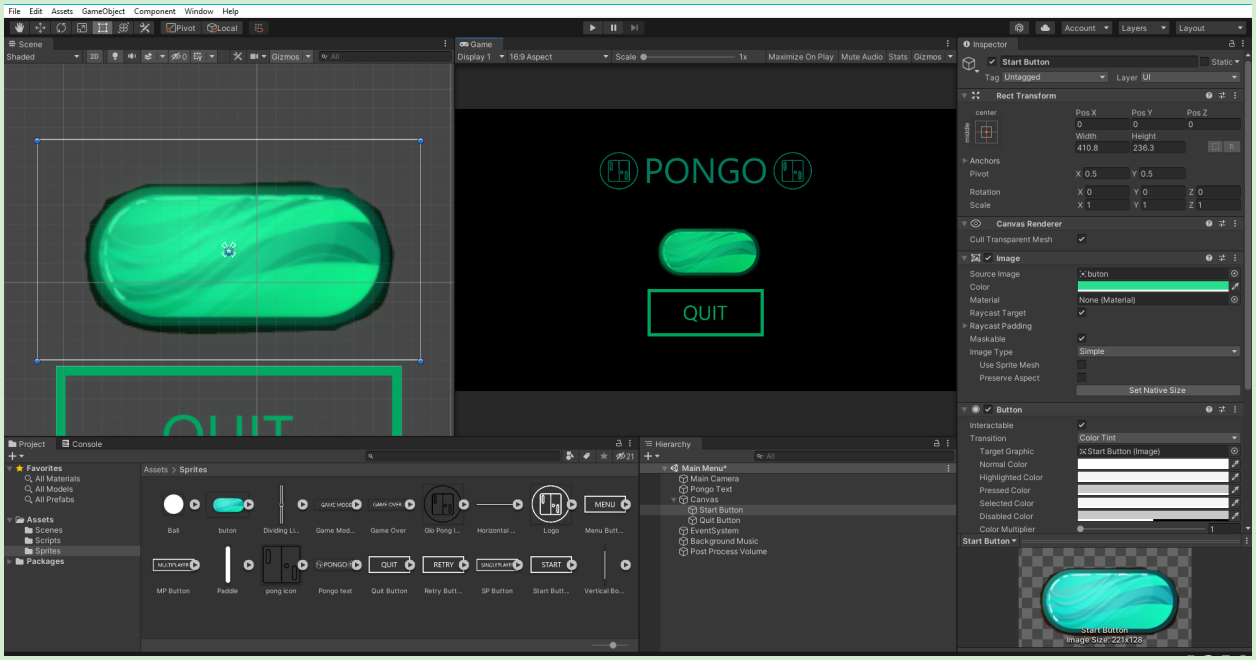

GAME MODE
GAME MODE


PONGO
PONGO

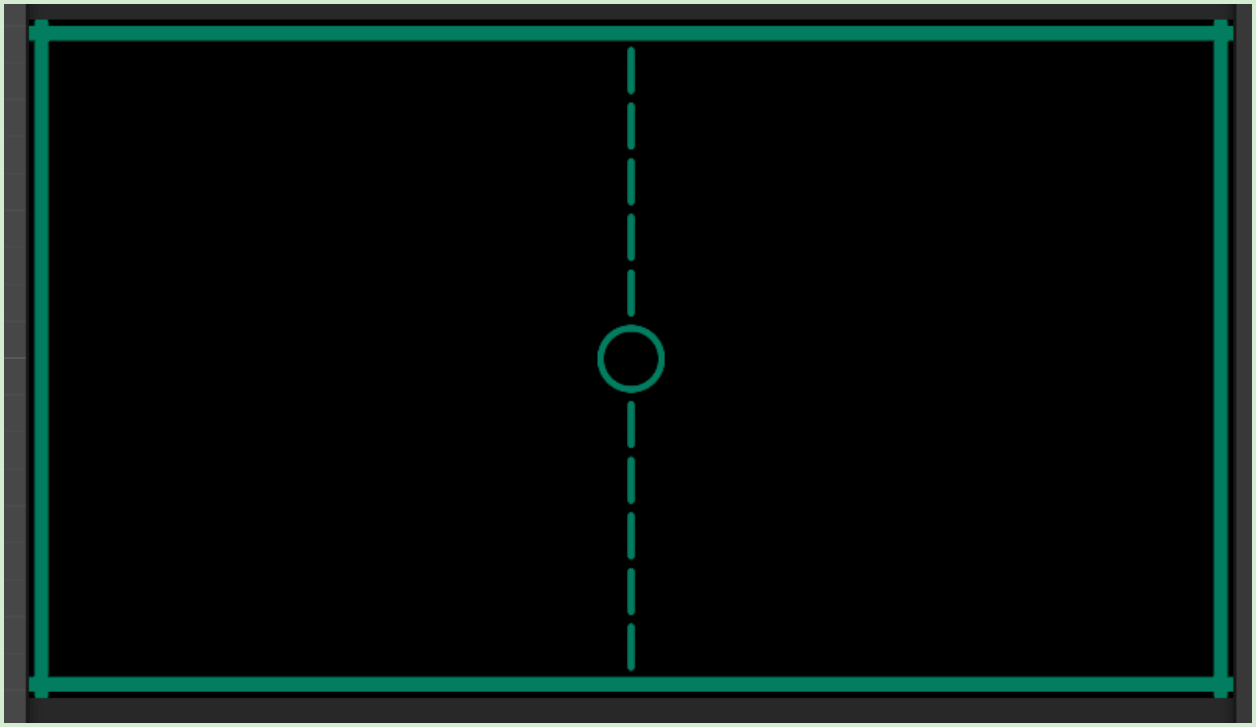

START

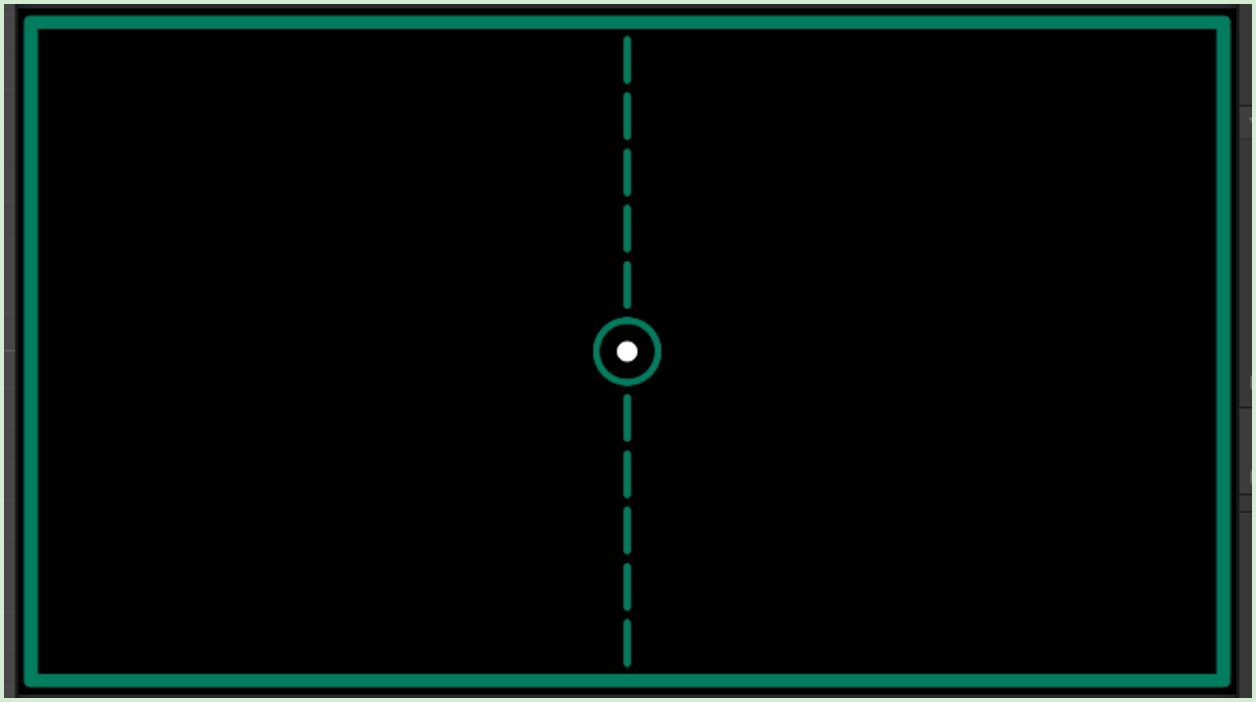

START

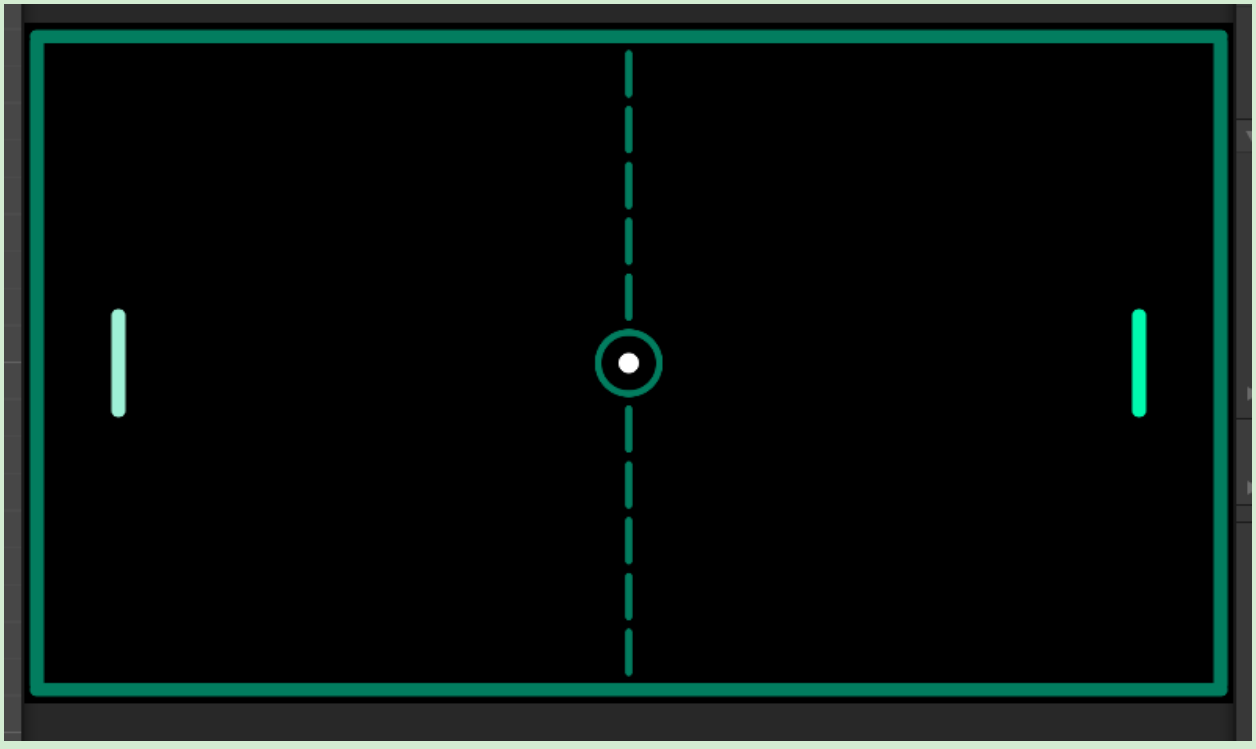
Testing out different buttons

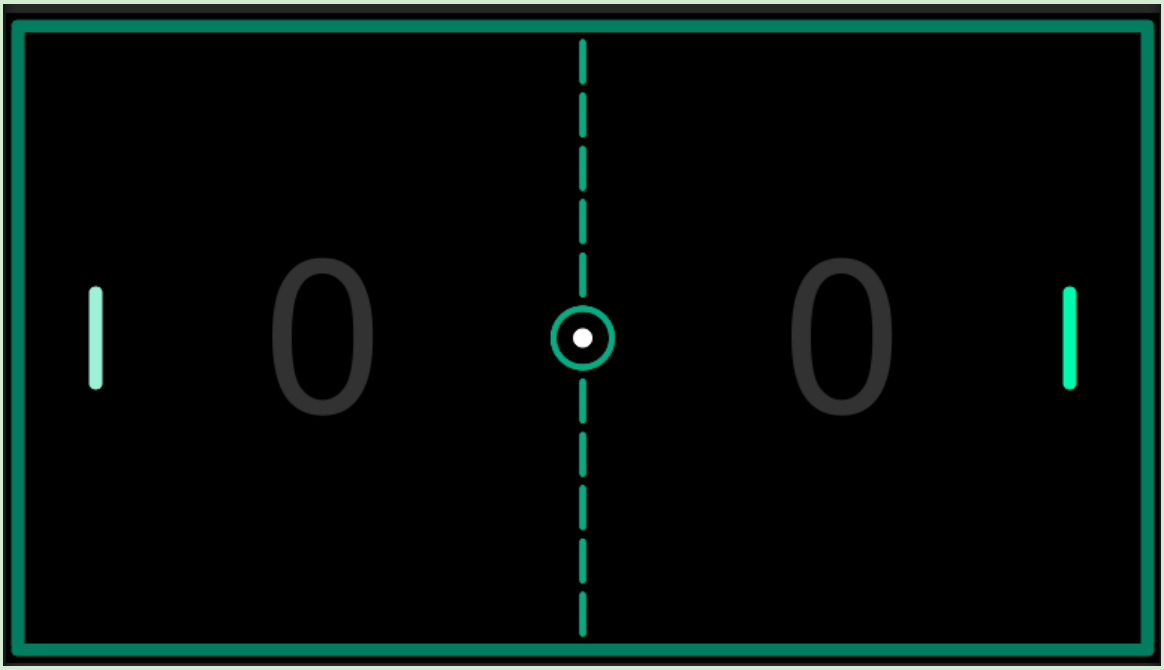


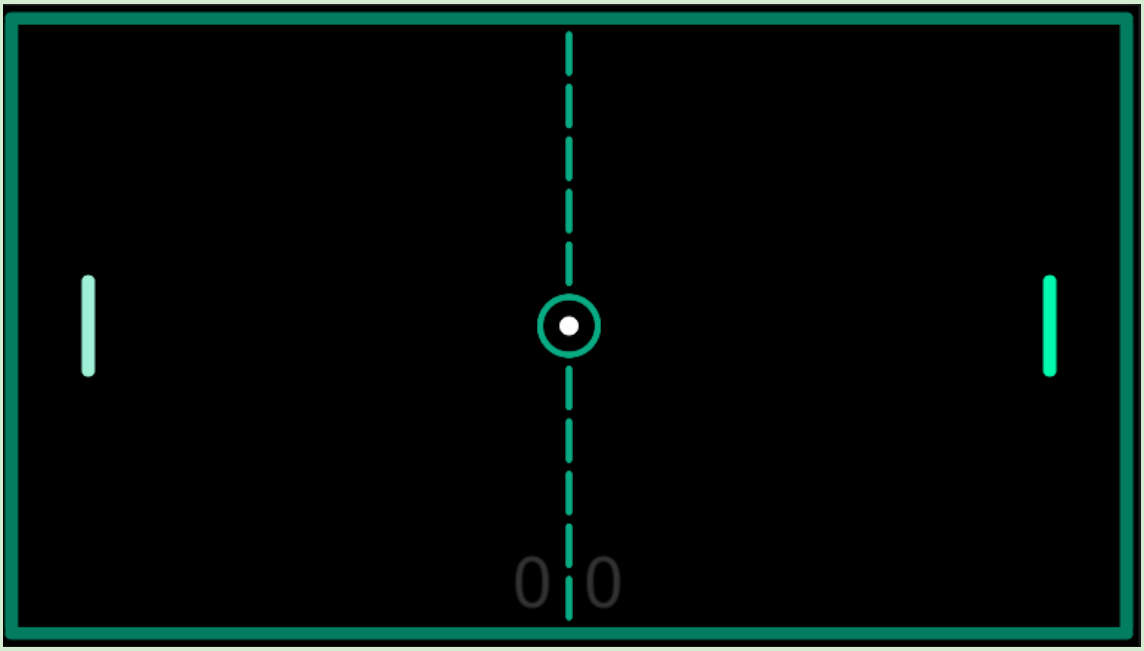
Evolution of the main game:







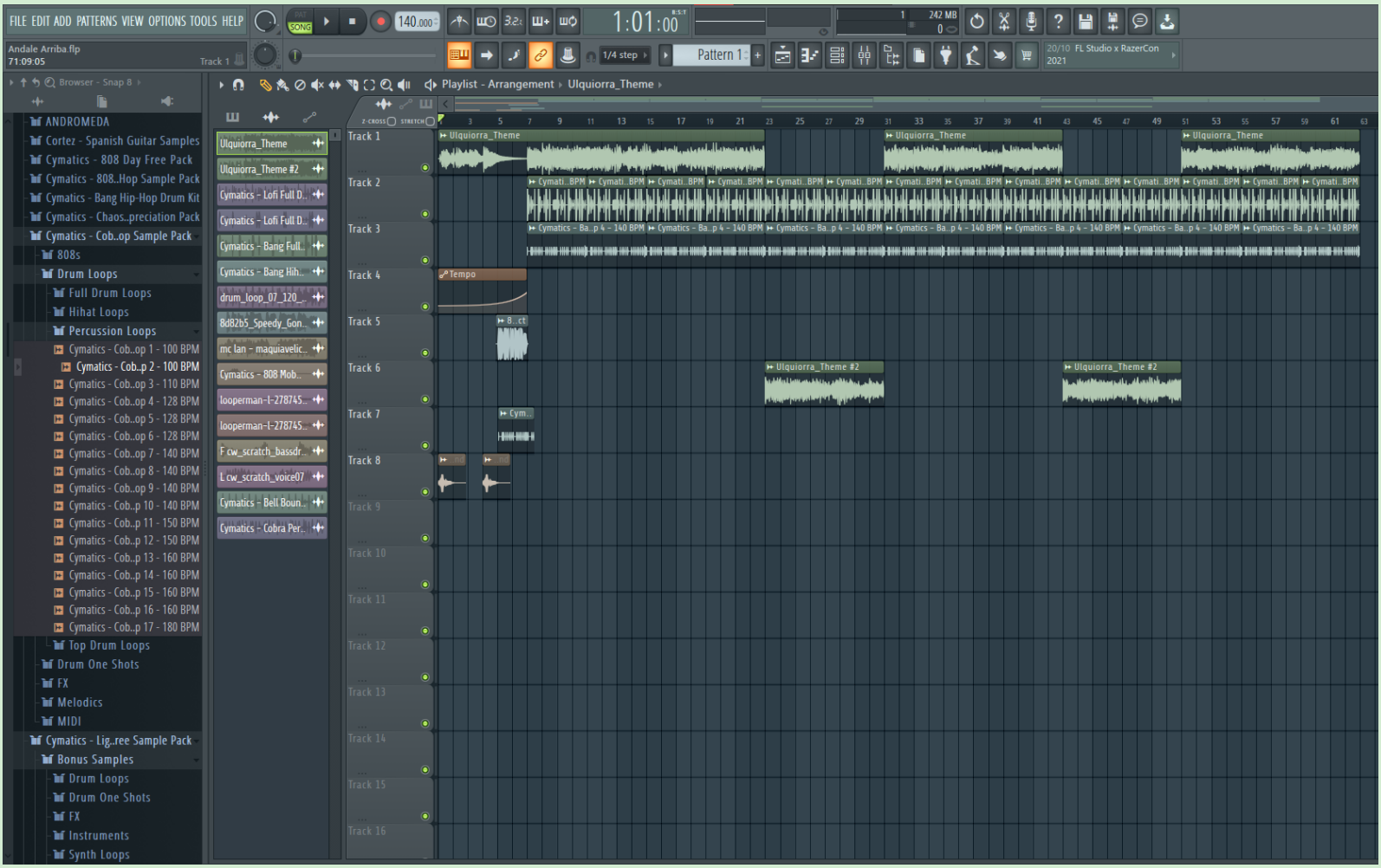

0
0


0 0

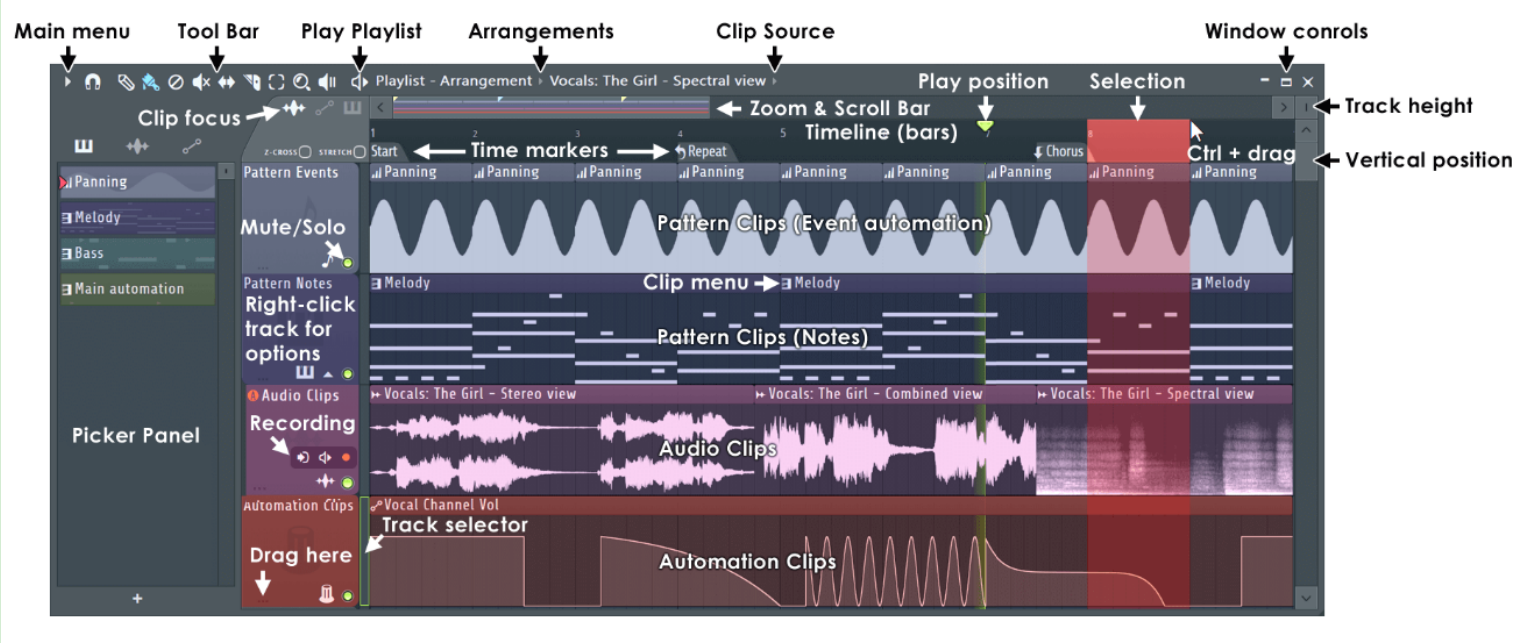
By: Ahmed Karim

Music creation

I have remixed an astonishing instrumental track called '[Fiesta de Guerra]' made by Nikawa Kayasaki to convey my personal attachment to this project using FL Studio.



Tools used to create this



The music took almost a week to create since I had to learn how to use the software, starting from the basics.

Understanding the software- tool bar icon description:

1. Playlist Menu - Includes: Edit, Tools, View, Snap, Select, Group, Zoom, Time Marker, Clip Source, Performance Mode, playhead and Detach options.
2. Playlist Snap - Snap determines how Clips will move and quantization aligns events relative to the background grid. For a detailed list of Snap options see the Piano roll menu > Snap section.
3. Add clips by Draw mode - Left-click to add the currently selected Clip.
4. Add clips by Paint mode - Left-click to adds the currently selected Clip.
5. Delete - Click or Click-and-drag to delete Clips.
6. Mute - The mute tool mutes individual Clips.
7. Slip Edit - Left-click on the content of Clips to slide them left or right relative to the time-line while retaining the start/end points of the Clip.
8. Slice - Click and drag vertically to make a slice through the clip OR use (Right-Shift) to Slice without the need to drag vertically or switch to the Slice Tool.
9. Select - Either Left-click on Clips or Left-click and drag to make group selections.
10. Play selected - Click the clips you want to play. Click position will set the start location.

User interface and project explanations (Q/A)

Who is the creator of 'Pongo'?

Hello! I am Ahmed Karim, a high school student currently studying in Year 12. If you have any questions please feel free to contact me at pongo@customerservice.com ((not real)). Hope you enjoy my game!

Why are the Menus set out this way?

The menus were created to be simple but convenient at the same time. The use of only 2 large buttons on each menu makes it easy for the user to decide what to do. Convenience can be seen in the mute button at the top right corner on the main menu so that the people who do not like the music can get rid of it straight away.

Why have you picked this colour scheme?

Green and black are some of my favourite classic colour combinations due to the animated show called "Ben 10". Since this game is one of my all-time favourite old school video games, I have combined it with my old colour taste.

Why did you use this specific font?

I used 'UNISPACE' as my font because it gives the game's user interface a modern look.

Where did you get inspiration for the theme?

I took inspiration from the looks of a game called Rocket League, a vehicular soccer video game developed and published by Psyonix. It has a futuristic look.

What software did you use to design the game?

I mostly used 'Paint 3D' and converted the images to PNGs with transparent backgrounds so I could use them in my game. I found 'Paint 3D' to be extremely simple to use.

How is this any different from other 'pong' games?

Other games, whether it be remastered or arcade versions of 'Pong', all shared the same issues- a lack of modernism and a bland black and white colour palette. I have tackled this issue in my version of 'Pong' using modern post-processing effects, giving the game a 'glowing' effect and also using a vibrant colour palette.

What is the purpose of this project?

The goal of this project is to revive a forgotten classic game known as 'Pong' and make a game suitable for casual play which relieves the brain from stress.

Are you happy with your creation?

Yes! This game came out exactly how I envisioned and I am proud to be the creator of it.

Why is this game so simple?

Modern games are extremely complex. They have missions, side quests, achievements, pointing systems, in-game currency, time limits and so much more. My goal was to create a simple game that would ease the player's brain from all this unnecessary data processing.

What age group is this game aimed at?

Any! This game can be enjoyed by anyone and everyone. Although if I had to put a number on it, I would say from 3-60 year olds since there are bright colours which could hurt older people's eyes.

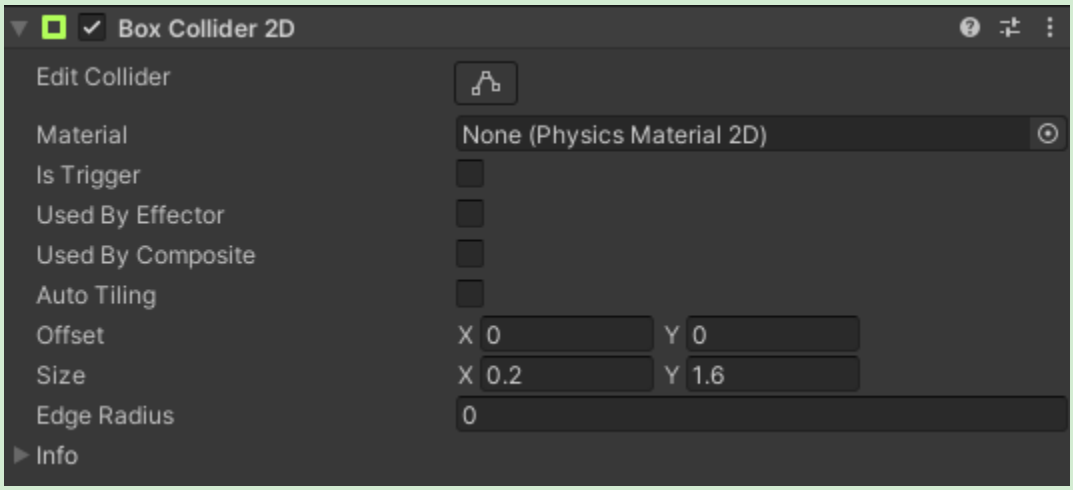
Why does the code seem minimal?

I utilised the Unity Physics engine to make my game. In simple terms, I had to interact with the Unity UI more than I had to code in Visual Studio in order to save time and make my game less complicated which fits perfectly with my usage of the RAD approach. This method is known as front-end development.

Technical specification - Components

Box Collider 2D:

Collider that interacts with the 2D physics system of Unity. It is a rectangle in shape with a defined position, width and height in the local coordinate space of a Sprite. Note that the rectangle is axis-aligned, with its edges parallel to the X or Y axes of local space.

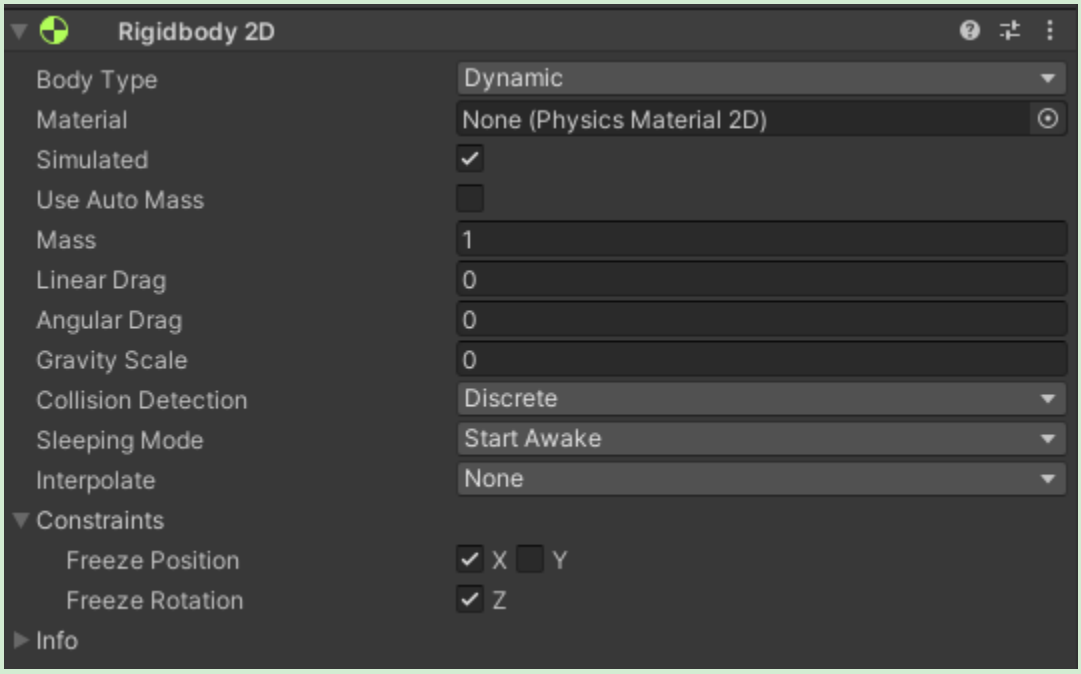


Property	Function
Material	A physics Material that determines properties of **collisions**, such as friction and bounce.
Is Trigger	Check this box if you want the **Box Collider** 2D to behave as a trigger.
Used by Effector	Check this box if you want the Box Collider 2D to be used by an attached Effector 2D component.
Used by Composite	Tick this checkbox if you want this Collider to be used by an attached Composite Collider 2D. When you enable **Used by Composite**, other properties disappear from the Box Collider 2D component, because they are now controlled by the attached Composite Collider 2D. The properties that disappear from the Box Collider 2D are **Material**, **Is Trigger**, **Used By Effector**, and **Edge Radius**.
Auto Tiling	Tick this checkbox if the Sprite Renderer component for the selected Sprite has the **Draw Mode** set to **Tiled**. This enables automatic updates to the shape of the Collider 2D, meaning that the shape is automatically readjusted when the Sprite's dimensions change. If you don't enable **Auto Tiling**, the Collider 2D geometry doesn't automatically repeat.
Offset	Set the local offset of the Collider 2D geometry.
Size	Set the size of the box in local space units.
Edge Radius	Controls a radius around edges, so that vertices are circular. This results in a larger Collider 2D with rounded convex corners. The default value for this setting is **0** (no radius).

This component was required in the paddles and walls.

Rigidbody 2D:

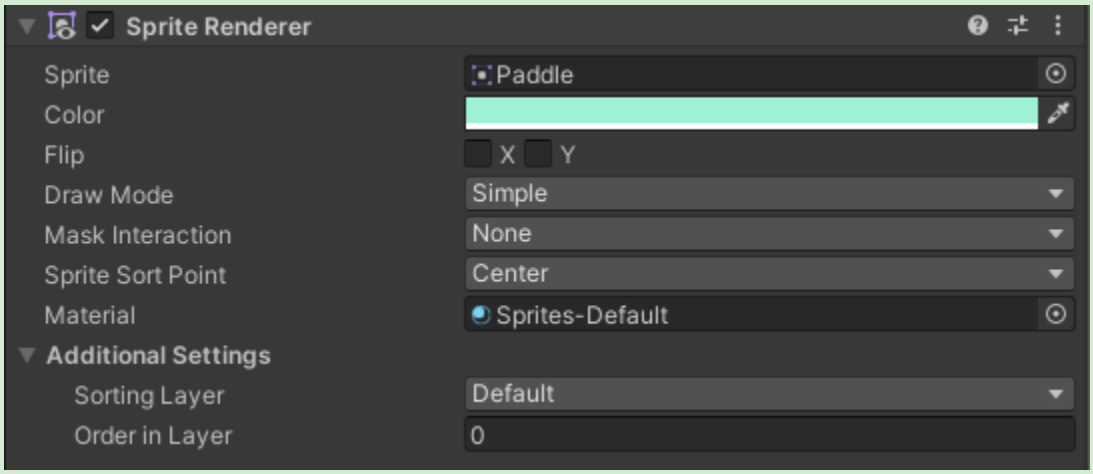
A Rigidbody 2D component places an object under the control of the physics engine. Many concepts familiar from the standard Rigidbody component carry over to Rigidbody 2D; the differences are that in 2D, objects can only move in the XY plane and can only rotate on an axis perpendicular to that plane. Adding a Rigidbody 2D allows a sprite to move in a physically convincing way by applying forces from the scripting API. When the appropriate collider component is also attached to the sprite GameObject, it is affected by collisions with other moving GameObjects. Using physics simplifies many common gameplay mechanics and allows for realistic behaviour with minimal coding.



This component was required in the paddles.

Sprite renderer:

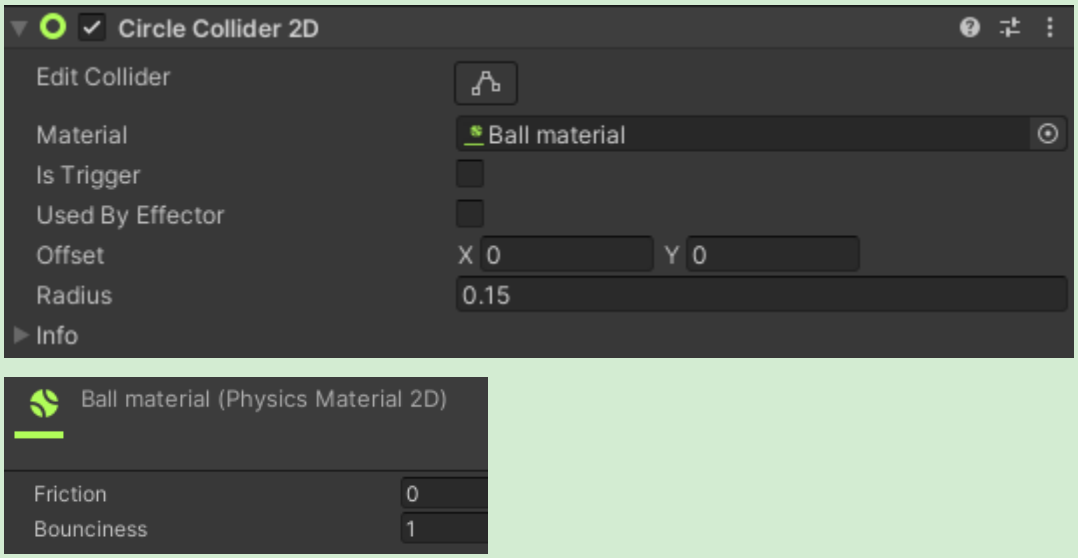
Renders a Sprite for 2D graphics.



This component was required for every sprite

Circle collider 2D:

Collider for 2D physics representing a circle. Basically makes the circular object gain a virtual 2D space so it can interact with in-game objects. 'Ball material' was part of the physics 2D materials which was automatically downloaded by unity upon the creation of my project.



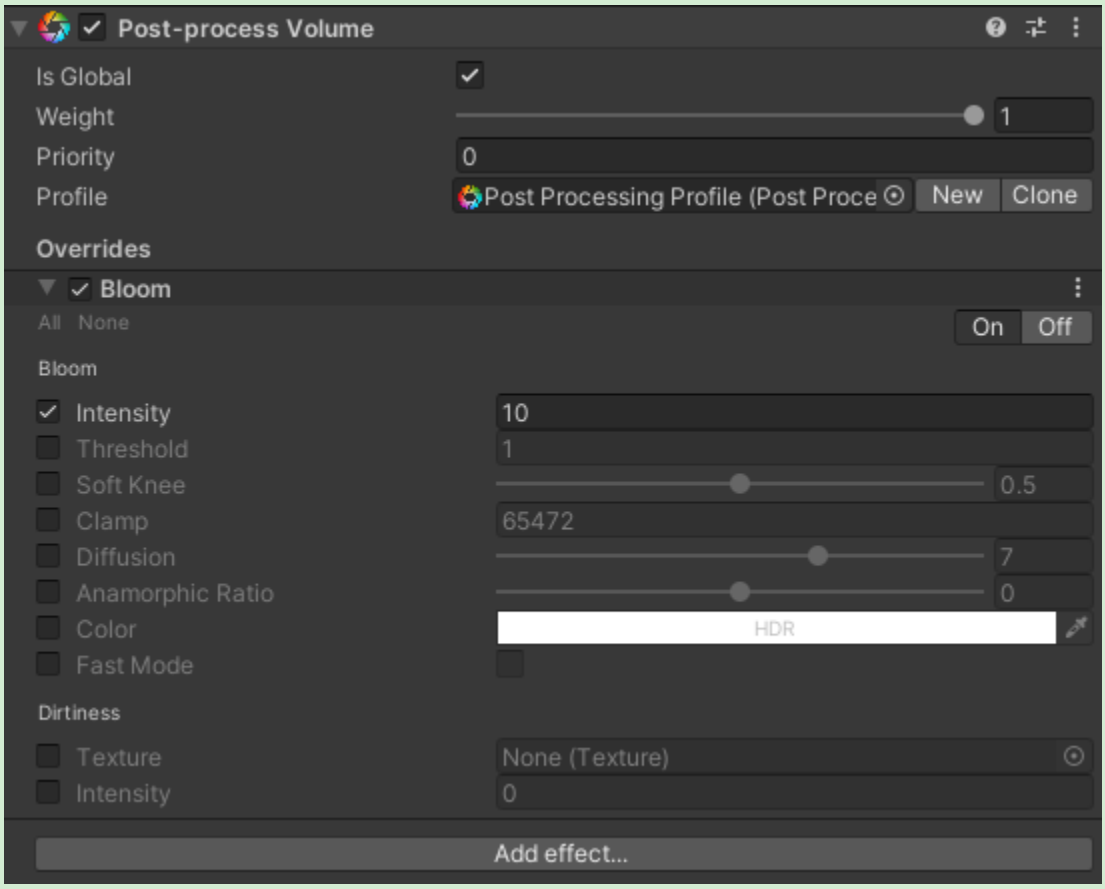
This component was required in the ball.

Audio Source:

This component can be used to play a single audio clip using Play, Pause and Stop functions. You can also adjust its volume while playing using the volume property, or seek using time. Required to give the ball a 'hit' sound effect and make the UI buttons have a sound effect.

Post Process Volume:

Screen-based image effects add specialised filters to the application camera which can be used to drastically increase the visual quality of a project. Unity's Post-Processing Stack replaces previous iterations of screen-based image effects and combines them all together in an easily configurable way.



This component was required to give the game's objects have a glowing effect.

Technical specifications-

Code with explanation

Note:

- All of the codes shown below have intrinsic documentation but were removed for this portion of the final documentation to help with the explanations.
- Proper grammar and additional in-depth explanations are present as opposed to the intrinsic documentation
- Please use provided numbers to read
- A larger screenshot including the intrinsic documentation is provided below the initial explanation

Player 1 Paddle

1. determines movement speed for player 1's paddle
2. abbreviation of the rigid body used in unity is called 'rb'
3. the x and y-axis of the paddle is named 'paddleDirection'
4. fetches the rigidbody2d from player 1 object in unity
5. required for getting input from player 1
6. determines the direction of the paddle in the y-axis
7. 'fixed update' is used in physics frames. It is necessary for my project since I have used the in-built unity physics features
8. speed of paddle is equal to the direction of travel times the paddle speed


```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player1 : MonoBehaviour
6  {
7      //1
8      public float paddleSpeed;
9
10     //2
11     private Rigidbody2D rb;
12     //3
13     private Vector2 paddleDirection;
14
15     private
16     // Start is called before the first frame update
17     void Start()
18     {
19         //4
20         rb = GetComponent<Rigidbody2D>();
21     }
22
23     // Update is called once per frame
24     void Update()
25     {
26         //5
27         float directionY = Input.GetAxisRaw("Vertical");
28
29         //6
30         paddleDirection = new Vector2(0, directionY).normalized;
31     }
32
33     //7
34
35     private void FixedUpdate()
36     {
37         //8
38         rb.velocity = paddleDirection * paddleSpeed;
39     }
40 }
```

What it does

- Required for movement of the paddle along the y-axis
- Assigns movement keys
- Makes the paddle move at a certain speed
- Makes the paddle interact with the ball

Difficulties

- Making the paddle speed fast enough to hit the ball but not too fast
- Getting Unity physics engine to cooperate with my code
- Figuring out how to fetch data from unity
- Changing the code to test the speed every single time

Player1.cs | Player2.cs | PlayerAI.cs | BallMovement.cs | BallBounce.cs | ScoreManager.cs | Char

Assembly-CSharp | Player1

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Player1 : MonoBehaviour
6   {
7       //determines movement speed for player 1's paddle
8       public float paddleSpeed;
9
10      //abbreviation of rigidbody is called rb
11      private Rigidbody2D rb;
12      //the x and y axis of the paddle is named paddledirection
13      private Vector2 paddleDirection;
14
15      private
16      // Start is called before the first frame update
17      void Start()
18      {
19          //fetches the rigidbody2d from player 1 object in unity
20          rb = GetComponent<Rigidbody2D>();
21      }
22
23      // Update is called once per frame
24      void Update()
25      {
26          //required for getting input from player 1
27          float directionY = Input.GetAxisRaw("Vertical");
28
29          //determines the direction of paddle in the y-axis
30          paddleDirection = new Vector2(0, directionY).normalized;
31      }
32
33      //fixed update is used in physics frames
34      //it is necessary for my project since i have used the in built unity physics feautures
35
36      private void FixedUpdate()
37      {
38          //speed of paddle is equal to the direction of travel times the paddle speed
39          rb.velocity = paddleDirection * paddleSpeed;
40      }
41  }
```

Player AI Paddle:

1. the rigid-body component in the ball is abbreviated as 'ball'
2. sets the ball's speed
3. another rigid body is abbreviated
4. makes it so the rigid body in the ball is called upon
5. physics-related logic requires this part
6. if ball position is greater than the ai paddle position
7. then the ai will move its paddle up
8. if ball position is less than the ai paddle position
9. then the ai will its paddle down
10. if the ball is moving away from the ai
11. ai paddle will idle in middle (makes ai smarter)
12. if ai paddle position is greater than the centre point
13. above centre point
14. move paddle down
15. moves paddle up

PlayerAI.cs* | Player1.cs | BallMovement.cs | BallBounce.cs | Play

Assembly-CSharp

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerAI : MonoBehaviour
{
    //1
    public Rigidbody2D ball;
    //2
    public float speed = 5.0f;
    //3
    protected Rigidbody2D _rigidbody;

    private void Awake()
    {
        //4
        _rigidbody = GetComponent<Rigidbody2D>();
    }

    //5
    private void FixedUpdate()
    {

        if (this.ball.velocity.x > 0.0f)
        {
            //6
            if (this.ball.position.y > this.transform.position.y)
            {
                //7
                _rigidbody.AddForce(Vector2.up * this.speed);
            }
            //8
            else if (this.ball.position.y < this.transform.position.y)
            {
                //9
                _rigidbody.AddForce(Vector2.down * this.speed);
            }
            //10
            //11
            else
            {
                //12
                //13
                if (this.transform.position.y > 0.0f)
                {
                    //14
                    _rigidbody.AddForce(Vector2.down * this.speed);
                }
                else if (this.transform.position.y < 0.0f)
                {
                    //15
                    _rigidbody.AddForce(Vector2.up * this.speed);
                }


            }
        }
    }
}
```

What it does

- Calls upon the paddle from unity
- Makes the opponent paddle move in the y-axis according to the ball in the singleplayer mode
- The paddle moves at a certain speed

Difficulties

- **Making the ai beatable by player**
  - This was one of the **hardest** challenges I had to overcome
  - The ai had to be good enough to not be deemed ‘easy to beat’ and had to be ‘human’ enough to make errors
  - Used various calculations and peer testing to fix
  - Took me a
- Correcting the velocities
  - The ai was overshooting or undershooting the ball
  - Sometimes the ai would miss the ball at the start of the game
  - There was also an infinite loop of the ai losing due to missing the ball
  - Had to make it slow enough to sometimes miss the ball

PlayerAI.cs | Player1.cs | Player2.cs | BallMovement.cs | BallBounce.cs | Score

Assembly-CSharp | PlayerAI

```
13
14      private void Awake()
15      {
16          //makes it so the rigidbody in the ball is called upon
17          _rigidbody = GetComponent<Rigidbody2D>();
18      }
19
20      //physics related logic requires this:
21      private void FixedUpdate()
22      {
23
24          if (this.ball.velocity.x > 0.0f)
25          {
26              //if ball position is greater than the ai paddle position
27              if (this.ball.position.y > this.transform.position.y)
28              {
29                  //then the ai will move its paddle up
30                  _rigidbody.AddForce(Vector2.up * this.speed);
31              }
32              //if ball position is less than the ai paddle position
33              else if (this.ball.position.y < this.transform.position.y)
34              {
35                  //then the ai will its paddle down
36                  _rigidbody.AddForce(Vector2.down * this.speed);
37              }
38              //if ball is moving away from the ai
39              //ai paddle will idle in middle (makes ai smarter)
40              else
41              {
42                  //if ai paddle position is greater than the centre point
43                  //above centre point
44                  if (this.transform.position.y > 0.0f)
45                  {
46                      //move paddle down
47                      _rigidbody.AddForce(Vector2.down * this.speed);
48                  }
49                  else if (this.transform.position.y < 0.0f)
50                  {
51                      //moves paddle up
52                      _rigidbody.AddForce(Vector2.up * this.speed);
53                  }
```

Player 2 Paddle:

1. determines movement speed for player 2's paddle
2. abbreviation of the rigid body is called ‘rb’
3. the x and y-axis of the paddle is named ‘paddleDirection’
4. fetches the rigidbody2d from player 2 object in unity
5. required for getting input from player 2
6. determines the direction of the paddle in the y-axis
7. ‘fixed update’ is used in physics frames
8. it is necessary for my project since I have used the inbuilt unity physics features
9. speed of paddle is equal to the direction of travel times the paddle speed

Player2.cs* | PlayerAI.cs | Player1.cs | BallMovement.cs | BallBounce.cs | SoundManager.c

Assembly-CSharp | Player2

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Player2 : MonoBehaviour
6   {
7       //1
8       public float paddleSpeed;
9
10      //2
11      private Rigidbody2D rb;
12      //3
13      private Vector2 paddleDirection;
14
15      private
16      // Start is called before the first frame update
17      void Start()
18      {
19          //4
20          rb = GetComponent<Rigidbody2D>();
21      }
22
23      // Update is called once per frame
24      void Update()
25      {
26          //5
27          float directionY = Input.GetAxisRaw("Vertical2");
28
29          //6
30          paddleDirection = new Vector2(0, directionY).normalized;
31      }
32
33      //7
34      //8
35      private void FixedUpdate()
36      {
37          //9
38          rb.velocity = paddleDirection * paddleSpeed;
39      }
40  }
```

What it does

- Required for movement of the paddle along the y-axis
- Assigns movement keys
- Makes the paddle move at a certain speed
- Makes the paddle interact with the ball

Difficulties

- Making the paddle speed fast enough to hit the ball but not too fast
- Getting Unity physics engine to cooperate with my code
- Figuring out how to fetch data from unity
- Changing the code to test the speed every single time

Player2.cs | PlayerAI.cs | Player1.cs | BallMovement.cs | BallBounce.cs | ScoreManager.cs | Ch

Assembly-CSharp | Player2

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player2 : MonoBehaviour
6  {
7      //determines movement speed for player 2's paddle
8      public float paddleSpeed;
9
10     //abbreviation of rigidbody is called rb
11     private Rigidbody2D rb;
12     //the x and y axis of the paddle is named paddledirection
13     private Vector2 paddleDirection;
14
15     private
16     // Start is called before the first frame update
17     void Start()
18     {
19         //fetches the rigidbody2d from player 2 object in unity
20         rb = GetComponent<Rigidbody2D>();
21     }
22
23     // Update is called once per frame
24     void Update()
25     {
26         //required for getting input from player 2
27         float directionY = Input.GetAxisRaw("Vertical2");
28
29         //determines the direction of paddle in the y-axis
30         paddleDirection = new Vector2(0, directionY).normalized;
31     }
32
33     //fixed update is used in physics frames
34     //it is necessary for my project since i have used the in built unity physics features
35     private void FixedUpdate()
36     {
37         //speed of paddle is equal to the direction of travel times the paddle speed
38         rb.velocity = paddleDirection * paddleSpeed;
39     }
40 }
```

Ball Movement 1st part:

1. determines starting speed of the ball
2. determines how much speed the ball is going to increase upon contact with the paddle
3. determines the maximum extra speed for the ball
4. whenever this statement is true p1 starts off with the ball and when it is false p2 starts instead
5. to keep track of how many times the ball has been hit by the paddle
6. we need this because we use simple physics to move the ball
7. to make the ball stop moving
8. makes the ball move back to the centre
9. the next 2 lines were implemented in order for the 'ienumerator' below it to work (c# generated)

BallMovement.cs* | BallBounce.cs | SoundManager.cs | PauseMenu.c

Assembly-CSharp

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallMovement : MonoBehaviour
{
    //1
    public float startSpeed;
    //2
    public float extraSpeed;
    //3
    public float maxExtraSpeed;

    //4
    public bool player1Start = true;

    //5
    private int hitCounter = 0;

    //6
    private Rigidbody2D rb;

    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();

        StartCoroutine(Launch());
    }

    private void RestartBall()
    {
        //7
        rb.velocity = new Vector2(0, 0);
        //8
        transform.position = new Vector2(0, 0);
    }

    //9
    private void StartCouroutine(IEnumerator enumerator)
    {
        throw new NotImplementedException();
    }
```

What it does

- Chooses the start speed of the ball
- Determines how much extra speed can be added to it.
- Makes the ball reset position when a point is gained
- Keeps track of how many times the ball has been hit by the racket
- Makes the ball wait for a few seconds to let the player get ready
- Makes the ball go towards the direction of the player who lost last round

Difficulties

- Ball was too fast or too slow
- Ball did not reset in the centre perfectly
- Extra speed gained too slowly
- Ball did not wait long enough for the player to get ready
- Ball stayed at centre

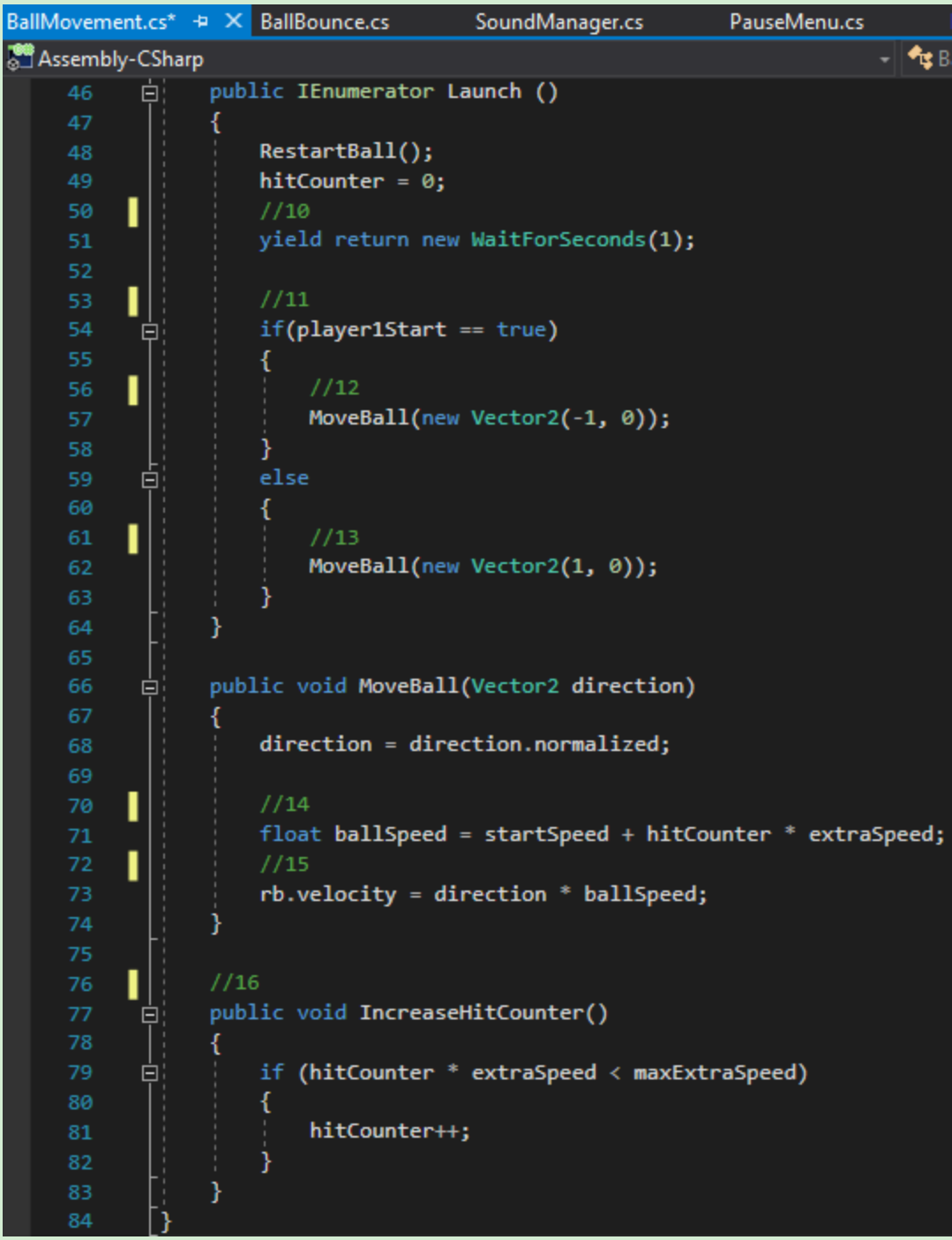
BallMovement.cs | Player2.cs | PlayerAI.cs | Player1.cs | BallBounce.cs | ScoreManager.cs | ChangeScene.cs

Assembly-CSharp | BallMovement

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class BallMovement : MonoBehaviour
7  {
8      //determines starting speed of ball
9      public float startSpeed;
10     //determines how much speed ball is going to increase upon contact with paddle
11     public float extraSpeed;
12     //determines the maximum extra speed for the ball
13     public float maxExtraSpeed;
14
15     //whenever this statement is true p1 starts off with the ball and when it is false p2 starts instead
16     public bool player1Start = true;
17
18     //to keep track of how many times the ball has been hit by the racket
19     private int hitCounter = 0;
20
21     //we need this because we use simple physics to move the ball
22     private Rigidbody2D rb;
23
24     // Start is called before the first frame update
25     void Start()
26     {
27         rb = GetComponent<Rigidbody2D>();
28
29         StartCoroutine(Launch());
30     }
31
32     private void RestartBall()
33     {
34         //to make ball stop moving
35         rb.velocity = new Vector2(0, 0);
36         //makes ball move back to the centre
37         transform.position = new Vector2(0, 0);
38     }
39
40     //the next 2 lines were implemented in order for the ienumerator below it to work (c# generated)
41     private void StartCouroutine(IEnumerator enumerator)
42     {
43         throw new NotImplementedException();
44     }
```

Ball Movement 2nd part:

10. makes the game wait for 1 second before the ball starts moving so players can get ready
11. if statement to check if p1 or p2 starts with the ball
12. this line of code results in the ball moving to the left towards p1
13. the ball moves right towards p2
14. determines ball speed according to how many times the paddle has made contact with the ball
15. applies previous code to the physics aspect of the ball
16. code required for ball speed to increase upon hit




```
46      public IEnumerator Launch ()
47      {
48          RestartBall();
49          hitCounter = 0;
50          //10
51          yield return new WaitForSeconds(1);
52
53          //11
54          if(player1Start == true)
55          {
56              //12
57              MoveBall(new Vector2(-1, 0));
58          }
59          else
60          {
61              //13
62              MoveBall(new Vector2(1, 0));
63          }
64      }
65
66      public void MoveBall(Vector2 direction)
67      {
68          direction = direction.normalized;
69
70          //14
71          float ballSpeed = startSpeed + hitCounter * extraSpeed;
72          //15
73          rb.velocity = direction * ballSpeed;
74      }
75
76      //16
77      public void IncreaseHitCounter()
78      {
79          if (hitCounter * extraSpeed < maxExtraSpeed)
80          {
81              hitCounter++;
82          }
83      }
84  }
```

What it does

- Chooses the start speed of the ball
- Determines how much extra speed can be added to it.
- Makes the ball reset position when a point is gained
- Keeps track of how many times the ball has been hit by the racket
- Makes the ball wait for a few seconds to let the player get ready
- Makes the ball go towards the direction of the player who lost last round

Difficulties

- Ball did not wait long enough for the player to get ready
- Ball stayed at centre
- Ball was too fast or too slow
- Ball did not reset in the centre perfectly
- Extra speed gained too slowly

BallMovement.cs | Player2.cs | PlayerAI.cs | Player1.cs | BallBounce.cs | ScoreManager.cs | ChangeScene.cs

Assembly-CSharp | BallMovement

```
46      public IEnumerator Launch ()
47      {
48          RestartBall();
49          hitCounter = 0;
50          //makes game wait for 1 second before ball starts moving so players can get ready
51          yield return new WaitForSeconds(1);
52
53          //if statement to check if p1 or p2 starts with ball
54          if(player1Start == true)
55          {
56              //this line of code results in ball moving to the left towards p1
57              MoveBall(new Vector2(-1, 0));
58          }
59          else
60          {
61              //ball moves right towards p2
62              MoveBall(new Vector2(1, 0));
63          }
64      }
65
66      public void MoveBall(Vector2 direction)
67      {
68          direction = direction.normalized;
69
70          //determines ball speed according to how many times the paddle has made contact with the ball
71          float ballSpeed = startSpeed + hitCounter * extraSpeed;
72          //applies previous code to the physics aspect of the ball
73          rb.velocity = direction * ballSpeed;
74      }
75
76      //code required for ball speed to increase upon hit
77      public void IncreaseHitCounter()
78      {
79          if (hitCounter * extraSpeed < maxExtraSpeed)
80          {
81              hitCounter++;
82          }
83      }
84  }
```

Ball Bounce 1st part:

1. making variable for hit sound effect
2. this code is to utilise the 'IncreaseHitCounter' algorithm presented in another script
3. reference to 'ScoreManager' script for scoring functionality
4. getting the position of the object the ball had a collision with
5. to know which side of the paddle the ball made contact with
6. following code required to know if the ball hit p1 or p2 paddle
7. increases the number of hits
8. moves the ball along the axes

BallBounce.cs* | SoundManager.cs | PauseMenu.cs | MuteAudio.cs | DestroySoundEffect.cs

Assembly-CSharp | BallBounce

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallBounce : MonoBehaviour
{
    //1
    public GameObject hitSFX;
    //2
    public BallMovement ballMovement;
    //3
    public ScoreManager scoreManager;

    private void Bounce(Collision2D collision)
    {
        Vector3 ballPosition = transform.position;
        //4
        Vector3 paddlePosition = collision.transform.position;
        //5
        float paddleHeight = collision.collider.bounds.size.y;

        //6
        float positionX;
        if(collision.gameObject.name == "Player 1")
        {
            positionX = 1;
        }

        else
        {
            positionX = -1;
        }
        float positionY = (ballPosition.y - paddlePosition.y) / paddleHeight;

        //6
        ballMovement.IncreaseHitCounter();
        //7
        ballMovement.MoveBall(new Vector2(positionX, positionY));
    }
```

What it does

- Makes sound effects upon touching anything
- 'hit counter' is increased to increase speed
- If sides are touched then a point is given to the player
- Scoring functionality was key for this function
- Checks if sides or top was touched

Difficulties

- Deciding how many points was enough to have a enjoyable gameplay experience
- Deciding the speed which was not too fast or too slow
- The sound effect repeated upon contact, so I had to destroy it using other code

BallBounce.cs | BallMovement.cs | Player2.cs | PlayerAI.cs | Player1.cs | ScoreManager.cs | Ch

Assembly-CSharp | BallBounce

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class BallBounce : MonoBehaviour
6   {
7       //making variable for hit sound effect
8       public GameObject hitSFX;
9       //this code is to utilize  the increashitcounter algorithm presented in another script
10      public BallMovement ballMovement;
11      //reference to scoremanager script for scoring functionality
12      public ScoreManager scoreManager;
13
14      private void Bounce(Collision2D collision)
15      {
16          Vector3 ballPosition = transform.position;
17          //getting the position of the object the ball had collision with
18          Vector3 paddlePosition = collision.transform.position;
19          //to know which side of the paddle the ball made contact with
20          float paddleHeight = collision.collider.bounds.size.y;
21
22          //following code required to know if the ball hit p1 or p2 paddle
23          float positionX;
24          if(collision.gameObject.name == "Player 1")
25          {
26              positionX = 1;
27          }
28
29          else
30          {
31              positionX = -1;
32          }
33          float positionY = (ballPosition.y - paddlePosition.y) / paddleHeight;
34
35          //increases amount of hits
36          ballMovement.IncreaseHitCounter();
37          //moves the ball along the axes
38          ballMovement.MoveBall(new Vector2(positionX, positionY));
39      }
```

Ball Bounce 2nd part:

1. following code required to detect collision:
2. collision2d parameter is named collision
3. detects if the ball has made contact with a paddle.
4. contact with paddle results in the ball bouncing off
5. checks if the ball touched the right side
6. gives p1 a point
7. 'Player1Goal' variable was called from the 'ScoreManager' script
8. if p1 scored a goal then set the variable to false so that p2 gets the ball next round
9. calling launch coroutine
10. checks if the ball touched the left side
11. gives p2 a point
12. if p2 scored a goal then set the variable to true so that p1 gets the ball next round
13. calling launch coroutine
14. makes sound effects depending on the position the ball is in

BallBounce.cs* | SoundManager.cs | PauseMenu.cs | MuteAudio.cs | DestroySoundEffect.c

Assembly-CSharp | BallBounce

```
41      //1
42
43      //2
44      private void OnCollisionEnter2D(Collision2D collision)
45      {
46          //3
47          if (collision.gameObject.name == "Player 1" || collision.gameObject.name == "Player 2")
48          {
49              //4
50              Bounce(collision);
51          }
52
53          //5
54          else if(collision.gameObject.name == "Right Side")
55          {
56              //6
57              //7
58              scoreManager.Player1Goal();
59
60              //8
61              ballMovement.player1Start = false;
62              //9
63              StartCoroutine(ballMovement.Launch());
64          }
65          //10
66          else if (collision.gameObject.name == "Left Side")
67          {
68              //11
69              scoreManager.Player2Goal();
70
71              //12
72              ballMovement.player1Start = true;
73              //13
74              StartCoroutine(ballMovement.Launch());
75          }
76
77          //14
78          Instantiate(hitSFX, transform.position, transform.rotation);
79      }
80  }
```

What it does

- Makes sound effects upon touching anything
- ‘hit counter’ is increased to increase speed
- If sides are touched then a point is given to the player
- Scoring functionality was key for this function
- Checks if sides or top was touched

Difficulties

- Deciding how many points was enough to have a enjoyable gameplay experience
- Deciding the speed which was not too fast or too slow
- The sound effect repeated upon contact, so I had to destroy it using other code

BallBounce.cs | BallMovement.cs | Player2.cs | PlayerAI.cs | Player1.cs | ScoreManager.cs | ChangeScen

Assembly-CSharp | BallBounce

```
41      //following code required to detect collision
42
43      //collision2d parameter is named collision
44      private void OnCollisionEnter2D(Collision2D collision)
45      {
46          //detects if ball has made contact with a paddle
47          if (collision.gameObject.name == "Player 1" || collision.gameObject.name == "Player 2")
48          {
49              //contact with paddle results in the ball bouncing off
50              Bounce(collision);
51          }
52
53          //checks if ball touched right side
54          else if(collision.gameObject.name == "Right Side")
55          {
56              //gives p1 a point
57              //player1goal variable was called from scoremanager script
58              scoreManager.Player1Goal();
59
60              //if p1 scored a goal then set variable to false so that p2 gets the ball next round
61              ballMovement.player1Start = false;
62              //calling launch coroutine
63              StartCoroutine(ballMovement.Launch());
64          }
65          //checks if ball touched left side
66          else if (collision.gameObject.name == "Left Side")
67          {
68              //gives p2 a point
69              scoreManager.Player2Goal();
70
71              //if p2 scored a goal then set variable to true so that p1 gets the ball next round
72              ballMovement.player1Start = true;
73              //calling launch coroutine
74              StartCoroutine(ballMovement.Launch());
75          }
76
77          //makes sound effects depending on the position the ball is in
78          Instantiate(hitSFX, transform.position, transform.rotation);
79      }
80  }
```

By: Ahmed Karim

Score Manager:

1. to access unity UI elements
2. variable determines what score players need in order to win
3. sets player scores to zero
4. makes the pointing system use the texts at the bottom
5. the function is called whenever p1 scores a goal
6. adds 1 to p1 score
7. using to string function as p1 score is an integer
8. checks score to decide whether a side won yet or not
9. the function is called whenever p2 scores a goal
10. adds 1 to p2 score
11. using to string function as p2 score is an integer
12. checks score to decide whether a side won yet or not
13. change scene if certain points are reached
14. loads p1 winning screen if p1 wins
15. p1 winning screen
16. loads p2 winning screen if p2 wins
17. p2 winning screen

ScoreManager.cs* | SoundManager.cs | PauseMenu.cs | MuteA

Assembly-CSharp

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   //1
5   using UnityEngine.UI;
6   using UnityEngine.SceneManagement;
7
8   public class ScoreManager : MonoBehaviour
9   {
10      //2
11      public int scoreToReach;
12      //3
13      private int player1Score = 0;
14      private int player2Score = 0;
15
16      //4
17      public Text player1ScoreText;
18      public Text player2ScoreText;
19
20      //5
21      public void Player1Goal()
22      {
23          //6
24          player1Score++;
25          //7
26          player1ScoreText.text = player1Score.ToString();
27          //8
28          CheckScore();
29      }
30
31      //9
32      public void Player2Goal()
33      {
34          //10
35          player2Score++;
36          //11
37          player2ScoreText.text = player2Score.ToString();
38          //12
39          CheckScore();
40      }
41
42      //13
43      private void CheckScore()
44      {
45          //14
46          if (player1Score == scoreToReach)
47          {
48              //15
49              SceneManager.LoadScene(2);
50          }
51          //16
52          if (player2Score == scoreToReach)
53          {
54              //17
55              SceneManager.LoadScene(7);
56          }
```

What it does

- Adds score to overall points displayed at the bottom of the screen
- Sets scores to zero at the start of the games
- Constantly checks the scores
- Uses integers to relay information
- When a player reaches 5 points, it shows their respective winning screen

Difficulties

- After the game ended, it did not go to game over screen
- Deciding how many points was enough to have a enjoyable gameplay experience

ScoreManager.cs | BallBounce.cs | BallMovement.cs | Player2.cs

Assembly-CSharp

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//to acess unity ui elements
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class ScoreManager : MonoBehaviour
{
    //variable determines what score players need in order to win
    public int scoreToReach;
    //sets player scores to zero
    private int player1Score = 0;
    private int player2Score = 0;

    //makes the pointing system use the texts at the bottom
    public Text player1ScoreText;
    public Text player2ScoreText;

    //function is called whenever p1 scores a goal
    public void Player1Goal()
    {
        //adds 1 to p1 score
        player1Score++;
        //using to string function as p1 score is an interger
        player1ScoreText.text = player1Score.ToString();
        //checks score to decide whether a side won yet or not
        CheckScore();
    }

    //function is called whenever p2 scores a goal
    public void Player2Goal()
    {
        //adds 1 to p2 score
        player2Score++;
        //using to string function as p2 score is an interger
        player2ScoreText.text = player2Score.ToString();
        //checks score to decide whether a side won yet or not
        CheckScore();
    }

    //change scene if certain points is reached
    private void CheckScore()
    {
        //loads p1 winning screen if p1 wins
        if (player1Score == scoreToReach)
        {
            //p1 winning screen
            SceneManager.LoadScene(2);
        }
        //loads p2 winning screen if p2 wins
        if (player2Score == scoreToReach)
        {
            //p2 winning screen
            SceneManager.LoadScene(7);
        }
```

Sound Manager:

1. to access the properties of UI elements via code
2. we use serialised view when we want the code to be private but also show up in the editor
3. volume slider is called by this code
4. if there is no saved volume level by player
5. the volume will be set to one by default
6. the volume of the game will be equal to the volume slider
7. saves the volume temporarily
8. loads player's preferred volume level
9. saves the volume for when the player relaunches

SoundManager.cs* | PauseMenu.cs* | MuteAudio.cs* | DestroySoundEffec

Assembly-CSharp

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   //1
5   using UnityEngine.UI;
6
7   public class SoundManager : MonoBehaviour
8   {
9       //2
10      //3
11      [SerializeField] Slider volumeSlider;
12      // Start is called before the first frame update
13      void Start()
14      {
15          //4
16          if(!PlayerPrefs.HasKey("musicVolume"))
17          {
18              //5
19              PlayerPrefs.SetFloat("musicVolume", 1);
20              Load();
21          }
22
23      }
24
25      public void ChangeVolume()
26      {
27          //6
28          AudioListener.volume = volumeSlider.value;
29          //7
30          Save();
31      }
32
33      public void Load()
34      {
35          //8
36          volumeSlider.value = PlayerPrefs.GetFloat("musicVolume");
37      }
38
39      public void Save()
40      {
41          //9
42          PlayerPrefs.SetFloat("musicVolume", volumeSlider.value);
43      }
44  }
```

What it does

- Changes volume level
- Saves and loads player volume preferences
- Makes the default volume level 100%
- Allowed option to mute

Difficulties

- Sound was sometimes too loud
- Had to make the volume a suitable amount

SoundManager.cs | ScoreManager.cs | BallBounce.cs | BallMovement.cs

Assembly-CSharp

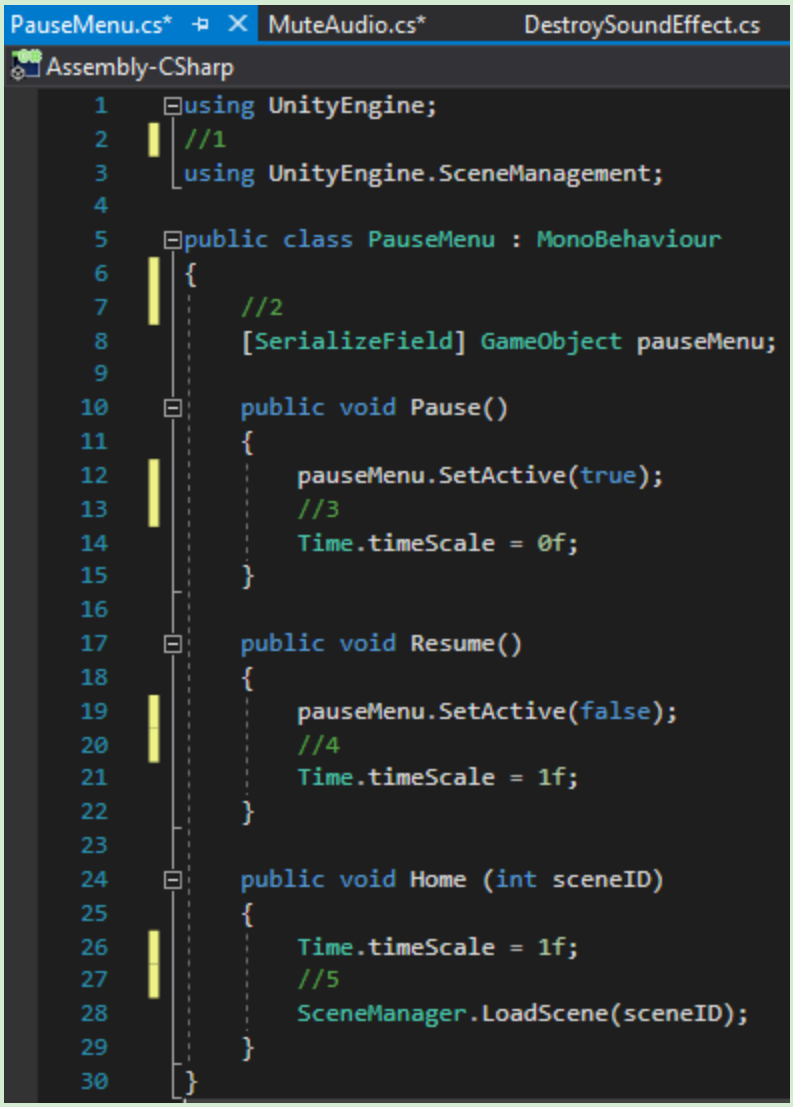
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//to access the properties of ui elements via code
using UnityEngine.UI;

public class SoundManager : MonoBehaviour
{
    //we use serialized view when we want the code to be private
    //but also show up in the editor
    //volume slider is called by this code
    [SerializeField] Slider volumeSlider;
    // Start is called before the first frame update
    void Start()
    {
        //if there is no saved volume level by player
        if(!PlayerPrefs.HasKey("musicVolume"))
        {
            //volume will be set to one by default
            PlayerPrefs.SetFloat("musicVolume", 1);
            Load();
        }

    }

    public void ChangeVolume()
    {
        //volume of game will be equal to the volume slider
        AudioListener.volume = volumeSlider.value;
        //saves the volume temporarily
        Save();
    }

    public void Load()
    {
        //loads player's preffered volume level
        volumeSlider.value = PlayerPrefs.GetFloat("musicVolume");
    }

    public void Save()
    {
        //saves the volume for when the player relaunches
        PlayerPrefs.SetFloat("musicVolume", volumeSlider.value);
    }
}
```

Pause Menu:

1. This namespace needed to be added in order to make sure the code works
2. the pause button is called upon
3. pauses the game by making the time passing in-game zero
4. resumes the game by making the time passing in-game one
5. loads the main menu when the home button is clicked

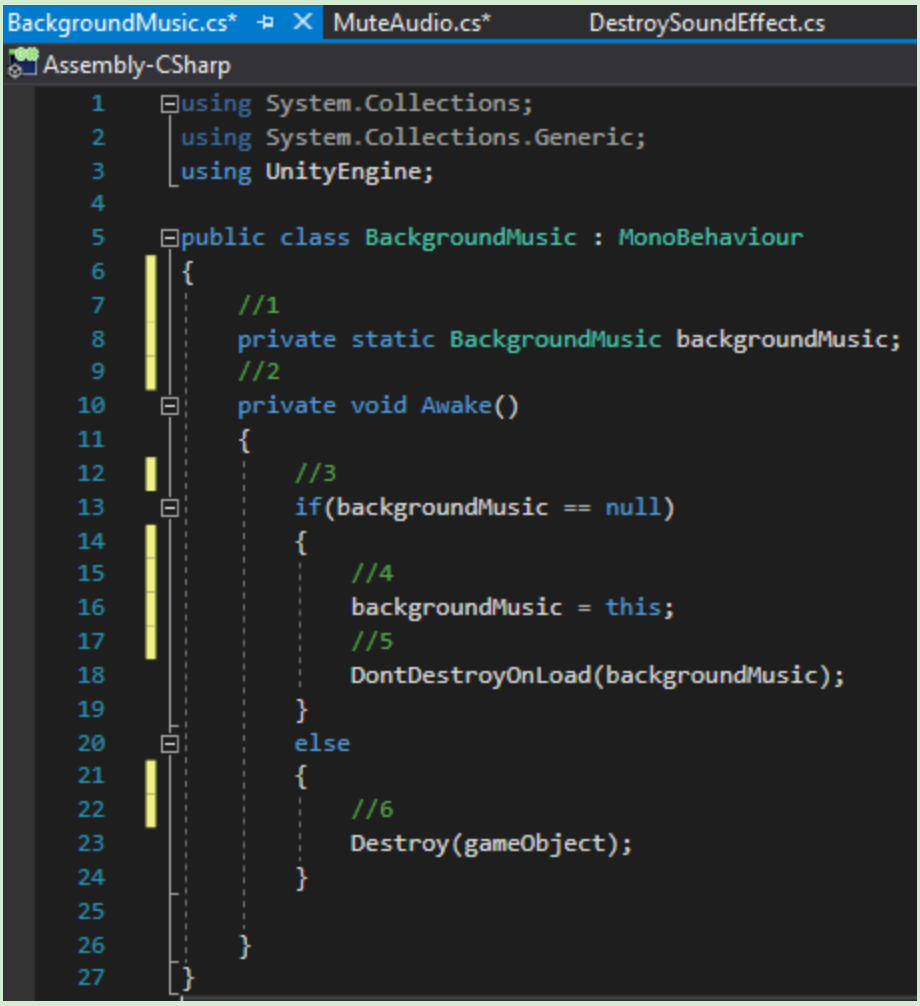
What it does:

- When the pause button is pressed, the game freezes
- When the resume button is pressed, the game starts again
- When the home button is pressed, the game goes to the main menu



Background Music:

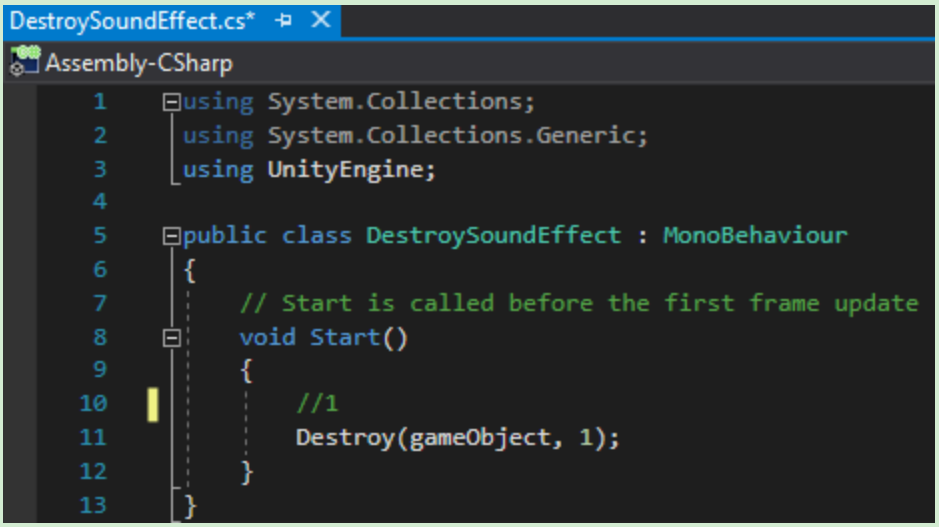
1. calls upon the background music
2. 'awake' keeps this code alive
3. if there is no background music then this line of code starts playing music
4. since I only used one background music for simplicity, I don't need to specify which music I want my game to play
5. makes it so the background music is not destroyed/stopped
6. makes it so the background music is not constantly being created



What it does:

- Background music starts when the game starts
- Checks whether background music is already playing
- Makes it so the game processing is not flooded with background music creation

Destroy Sound Effect:

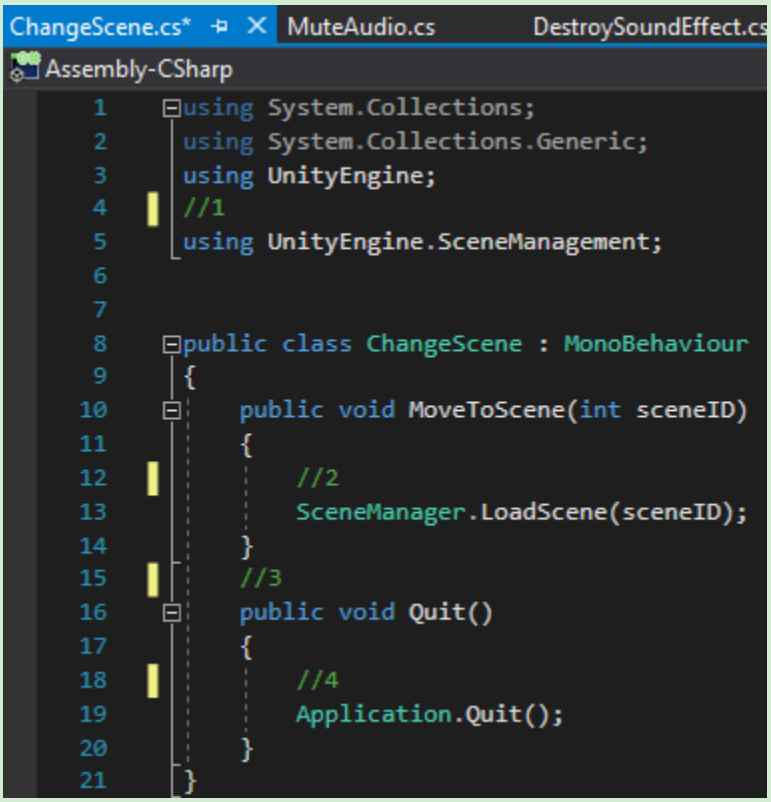


1. 'gameobject' gets destroyed after 1 second

What it does:

- When the ball collides with anything, it makes a noise. In order to get rid of the noise staying and piling up in the background processing of the game, it is necessary to destroy it

Change Scene:



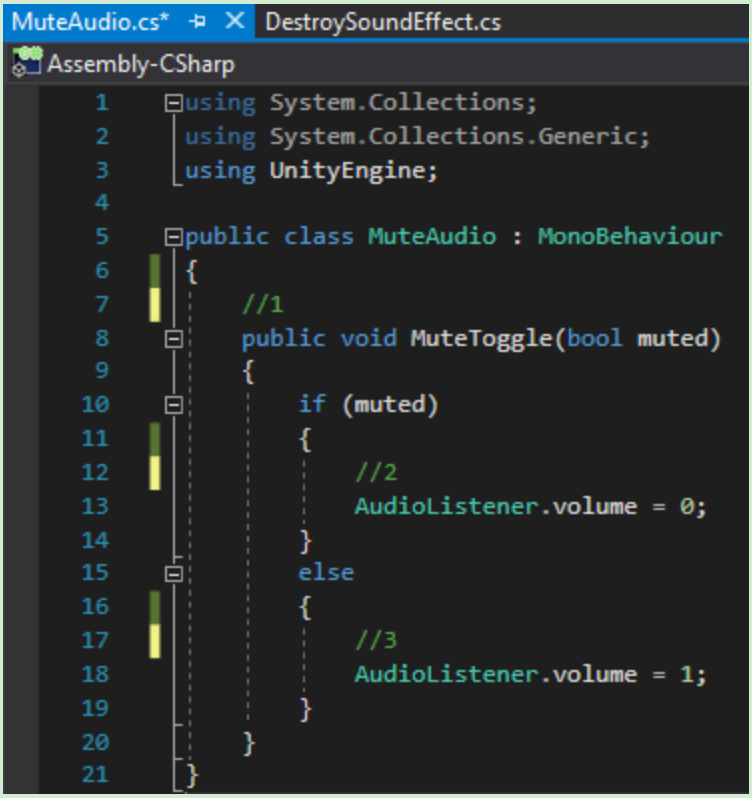
1. This namespace is needed to use a function called LoadScene from the SceneManager class
2. every scene created will have a unique ID assigned to it
3. this will be called whenever the player presses the quit button
4. the application will quit due to this script

What it does:

- Assigns unique IDs to different scenes
- When the player presses quit, the application will be closed

Mute Audio:

1. boolean is used to sense interaction with the toggle
2. when the muted box is checked the volume becomes zero
3. when the muted box is unchecked the volume becomes a hundred



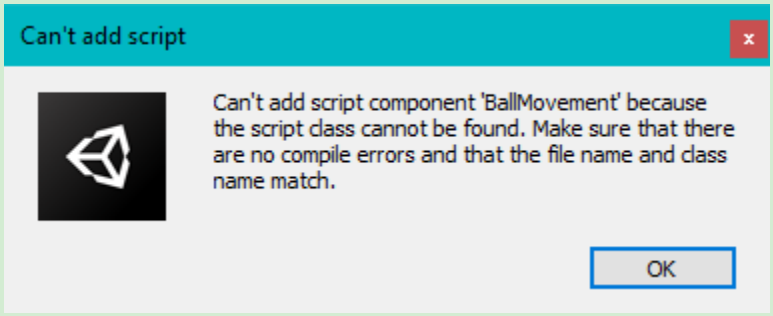
What it does:

- When the mute toggle is pressed, the game gets muted
- When the mute toggle is unpressed, the game regains its volume

Evaluation

Problems and solutions:

- Could not add the script for ball movement

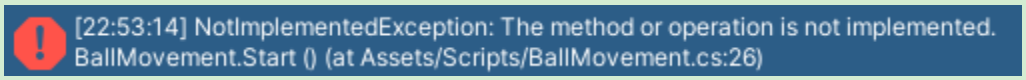


  - The solution was to change the name of the class at the start of the script

```
public class NewBehaviourScript : MonoBehaviour
```

```
public class BallMovement : MonoBehaviour
```

- Ball movement script was not working upon the first test



  - Spelt "Coroutine" wrong

```
StartCouroutine(Launch());
```

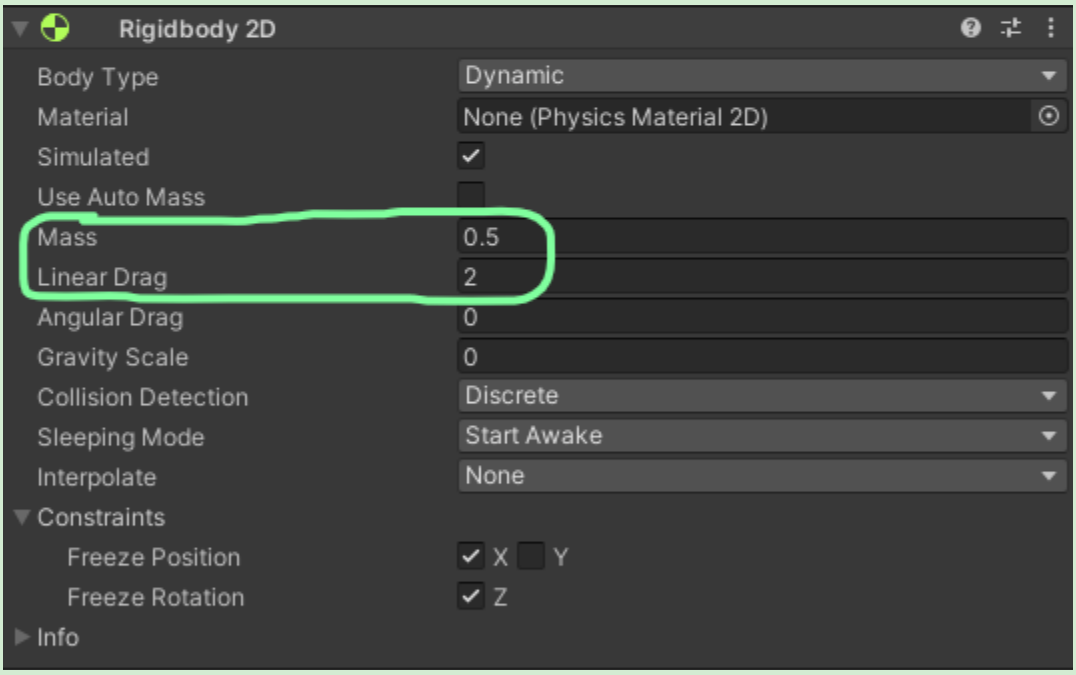
- Couldn't end game
  - Had to add 'CheckScore'. These 2 lines of code ensured the game kept the scores of both players between the number of points required to win, if a side reaches 5 goals this code is called upon.

```
public void Player1Goal()
{
    //player score goes up by 1
    player1Score++;
    //player score is an interger, so it is necessary to convert it to a string in order for it to be functional
    player1ScoreText.text = player1Score.ToString();
    CheckScore();
}

public void Player2Goal()
{
    //player score goes up by 1
    player2Score++;
    //player score is an interger, so it is necessary to convert it to a string in order for it to be functional
    player2ScoreText.text = player2Score.ToString();
    CheckScore();
}
```

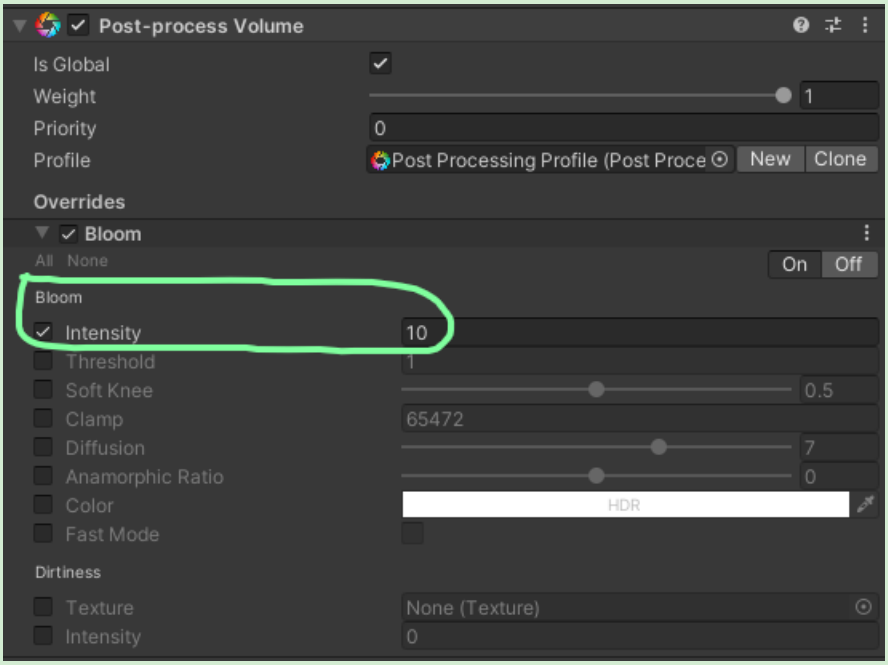
- Could not switch to different scenes when different players won
    - Had to separate the 'SceneManager.LoadScene' for each of the players

```
//change scene if certain points is reached
private void CheckScore()
{
    //loads p1 winning screen if p1 wins
    if (player1Score == scoreToReach)
    {
        //p1 winning screen
        SceneManager.LoadScene(2);
    }
    //loads p2 winning screen if p2 wins
    if (player2Score == scoreToReach)
    {
        //p2 winning screen
        SceneManager.LoadScene(7);
    }
}
```

- Ai Player kept missing the ball after 1 goal by either side was scored
    - Had to change the mass and linear drag

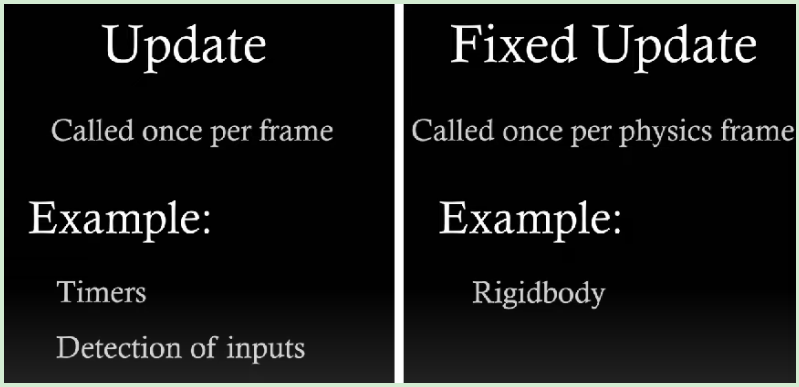


- Objects and texts were not glowing
    - Had to change the bloom of post-processing volume component



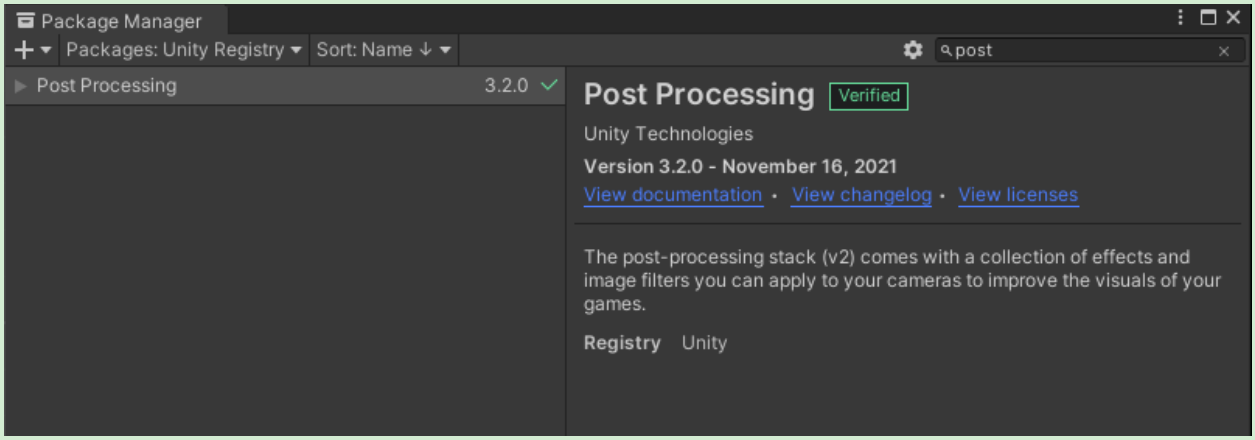
What did I learn?

- I learned the difference between update and fixed update
    - The player paddle needed 'fixed update' to function correctly: The difference between 'update' and 'fixed update' is that the 'update' function is called once per frame. So almost anything that needs to be adjusted regularly happens here. For example, timers and detection of inputs are usually done in the 'update' function. On the other hand 'fixed update' is used with objects which are a part of the physics engine so anything that needs to be applied to the rigid body should happen in 'fixed update'.

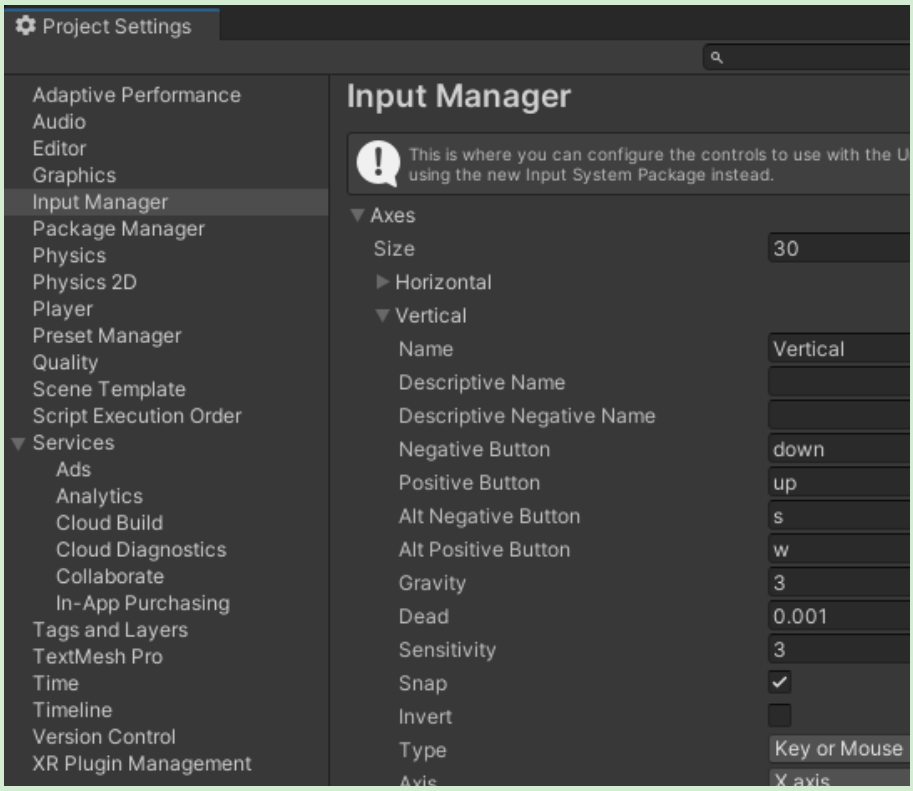



```
//fixed update is used in physics frames
//it is necessary for my project since i have used the in built unity physics feautures

private void FixedUpdate()
{
    rb.velocity = paddleDirection * paddleSpeed;
}
```

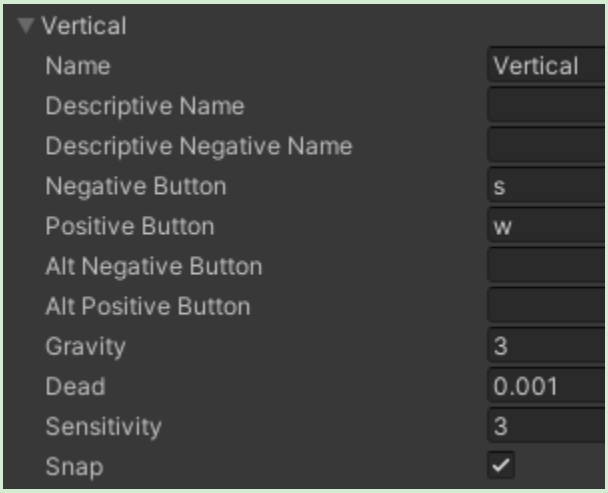
- I learned how to use the 'Post processing' (processing of the output of a computer program) package and access components required to put a glowing effect on my game



- My game required 2 different vertical axes for each paddle so I had to change 'size' by 1: From 29 to 30



- P1 paddle:



- I learned how to utilise the 'Unity Physics Engine' in order to become a more efficient developer.
- I learned how making simple games is not a simple job. The amount of work I needed to put into a basic game like 'Pong' was completely unexpected.

How I could have improved on my design?

- Add more animations
- Add effects on hitting the ball to make the game more lively
- Make a trail following the ball
- Add different difficulty levels for the ai in singleplayer
- Add more game modes
- Add online multiplayer along with local multiplayer

According to my peers I could have:

- Added different difficulty levels for the ai in singleplayer
- Added more levels
- Added a sensitivity slider
- Added more settings

Why I haven't done these:

- It would have made the game more complex, which opposes my purpose of creating a simple game which relieves the mind
- These were unnecessary settings/functions for the basic processes of the game

Summary:

The game I have created called ‘Pongo’ achieves everything I set as my goal. It is simple, straightforward, eases the brain and has a soothing gameplay experience. The game took me longer to make than I originally anticipated, but resulted in more hard work and time consumption which made the end result satisfying to call my own creation. I **intended** to modernise an unappreciated classic known as ‘Pong’ and I firmly believe I have **achieved my purpose**. The simplicity I wanted to provide in my game shines throughout the whole experience.

Overall, I am extremely happy with my creation and hope to achieve astonishing marks! Thank you for reading!