AI for Gaming Project Description

100748418 - Farhan Irani
100765502 - Ayi Pranayanda
100754729 - Anita Lim

INFR4320U Artificial Intelligence for Gaming

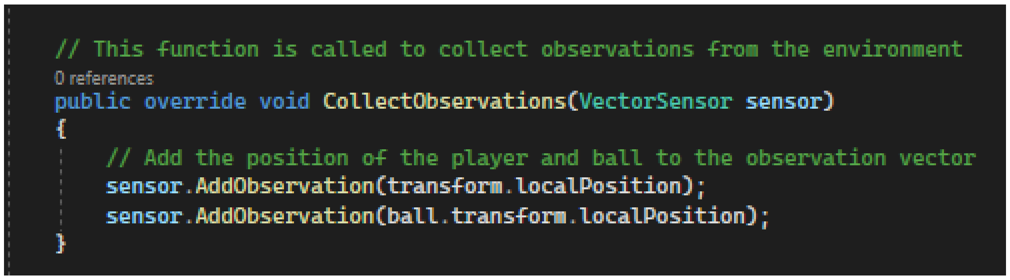
Winter 2023

1. Game Choice

Our team has made a decision to embark on the development of a short and simple Pong game that incorporates artificial intelligence (AI) techniques. The game will involve a simple and concise set of rules, with the objective of scoring points by hitting a ball back and forth between two paddles, one controlled by the player and the other by the AI. Points can be scored when the ball passes through the opposing player/AI. The game will be modeled to look like a soccer game with the field and the ball. We also added background music and sound effects to make the game more 'juicy' and appealing.

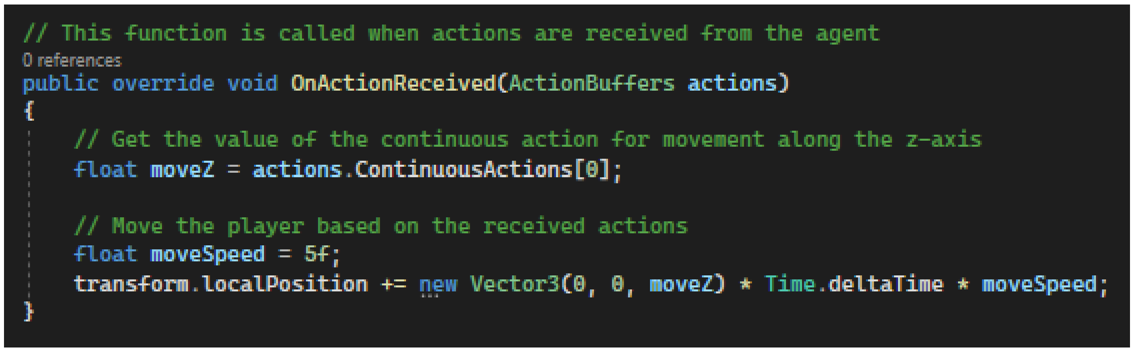
2. Which AI technique has been used and how it has been used in the game?

We used the AI technique reinforcement learning to create this game. The AI solution we used is MLAgents by unity. When training with this we add observations which are inputs for the AI agent to be aware of.



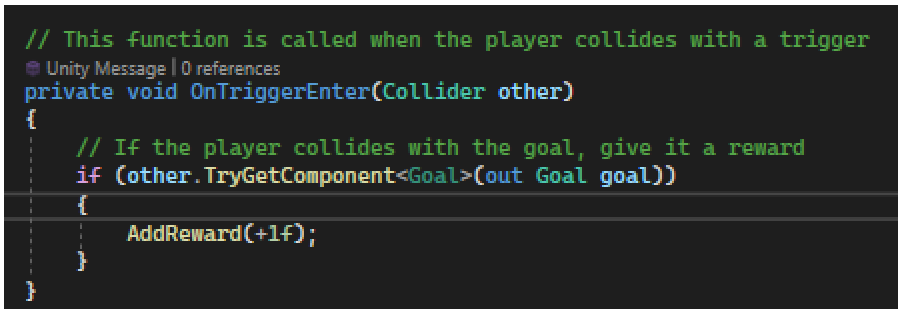
```
// This function is called to collect observations from the environment
0 references
public override void CollectObservations(VectorSensor sensor)
{
    // Add the position of the player and ball to the observation vector
    sensor.AddObservation(transform.localPosition);
    sensor.AddObservation(ball.transform.localPosition);
}
```

The two observations we have is the AI's own position, and the position of the ball. We also have a function that is called when actions are received from the agent. Right now we have a c continuous action for movement along the z-axis since in pong you only need to move in one axis. This gives the AI the ability to move in that axis without us explicitly telling it what to do.

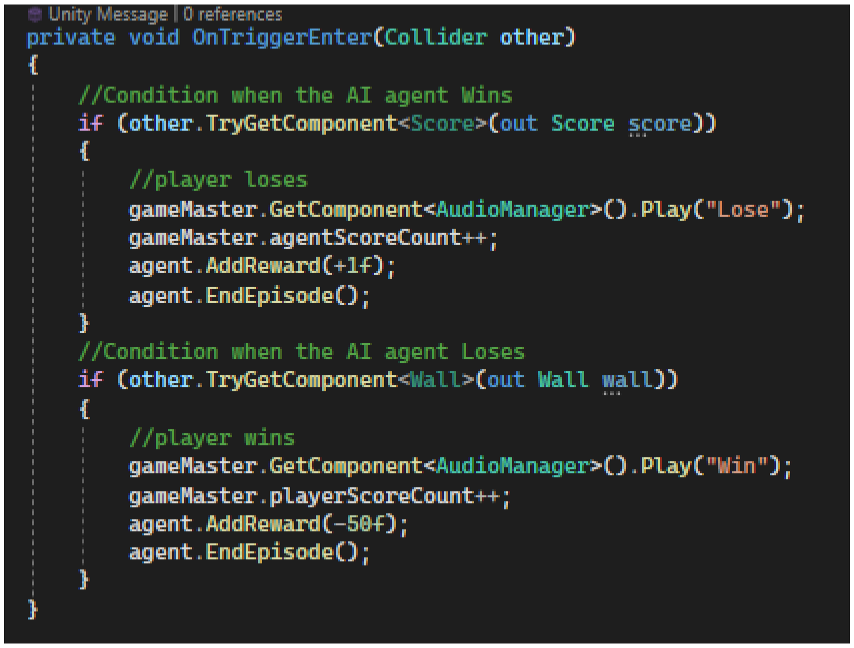


```
// This function is called when actions are received from the agent
0 references
public override void OnActionReceived(ActionBuffers actions)
{
    // Get the value of the continuous action for movement along the z-axis
    float moveZ = actions.ContinuousActions[0];

    // Move the player based on the received actions
    float moveSpeed = 5f;
    transform.localPosition += new Vector3(0, 0, moveZ) * Time.deltaTime * moveSpeed;
}
```

When training the AI we simply give it a positive reward for every time it can hit the ball with the paddle, and a negative reward if they let the ball pass them and hit the wall. In Unity we use the function OnTriggerEnter to add rewards;

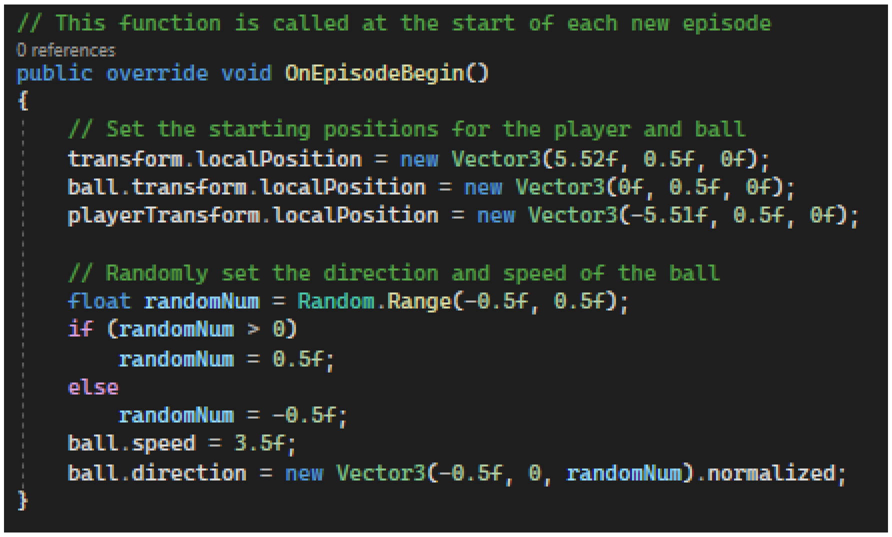


```
// This function is called when the player collides with a trigger
Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    // If the player collides with the goal, give it a reward
    if (other.TryGetComponent<Goal>(out Goal goal))
    {
        AddReward(+1f);
    }
}
```

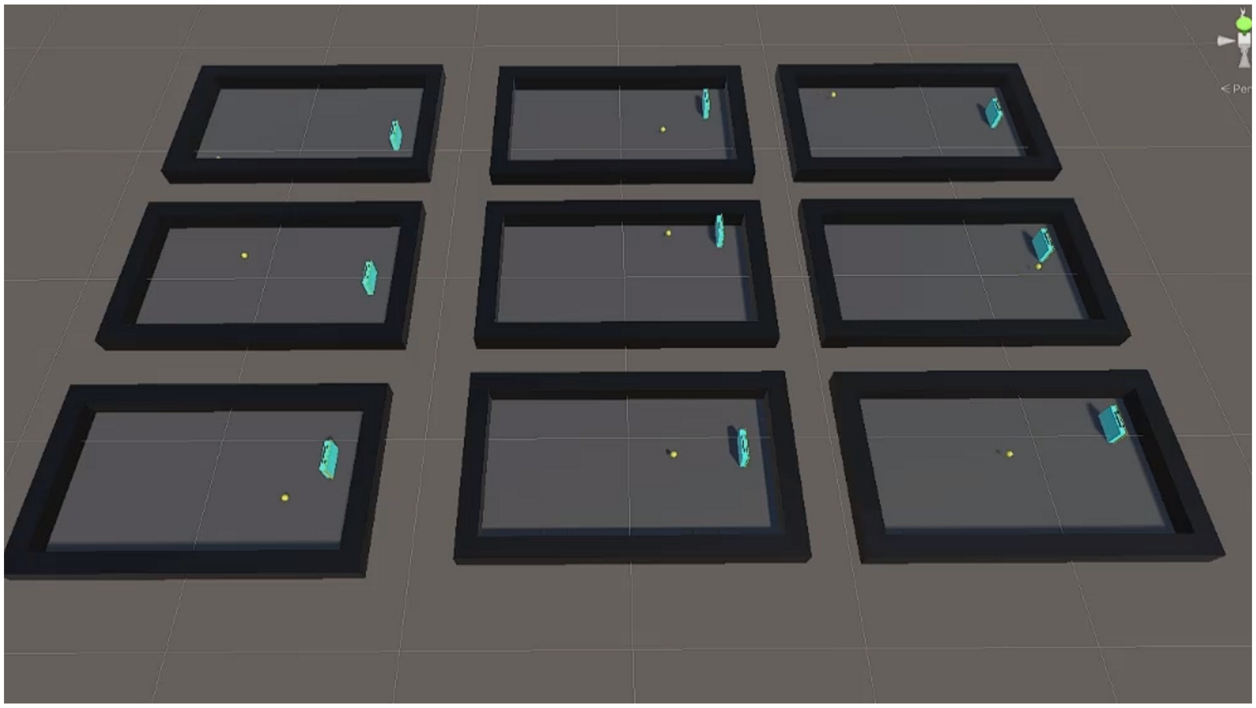


```
Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    //Condition when the AI agent Wins
    if (other.TryGetComponent<Score>(out Score score))
    {
        //player loses
        gameMaster.GetComponent<AudioManager>().Play("Lose");
        gameMaster.agentScoreCount++;
        agent.AddReward(+1f);
        agent.EndEpisode();
    }
    //Condition when the AI agent Loses
    if (other.TryGetComponent<Wall>(out Wall wall))
    {
        //player wins
        gameMaster.GetComponent<AudioManager>().Play("Win");
        gameMaster.playerScoreCount++;
        agent.AddReward(-50f);
        agent.EndEpisode();
    }
}
```

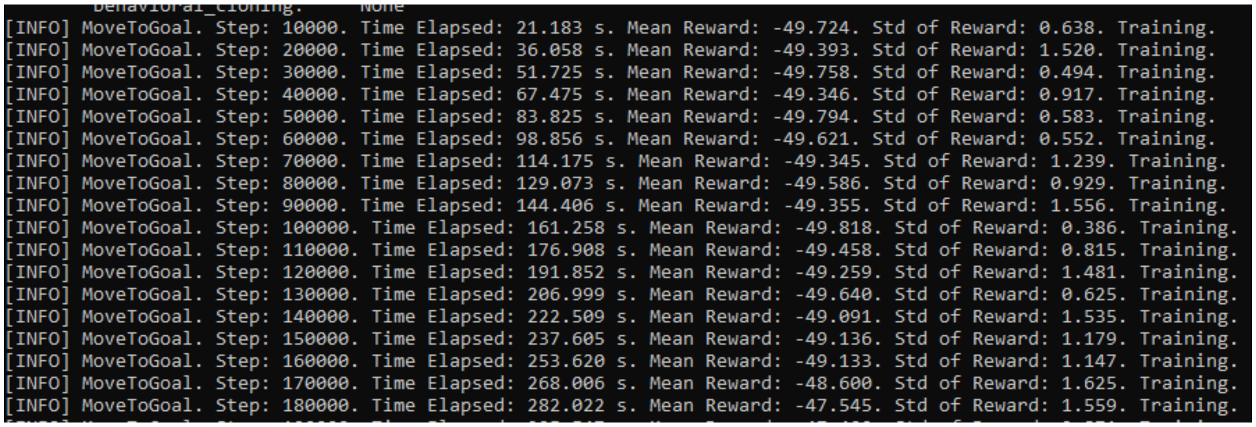
An episode is one training session for the AI to either fail or receive a reward, In our case an episode is when the AI lets the ball pass itself. As long as they are able to hit the ball back they can continue the episode until the max number of steps in an episode is reached and thus ending the episode. When a new episode starts, we randomly move the ball so that the AI can understand the ball will not be in the same spot every time. This is shown in the code below.


```
// This function is called at the start of each new episode
0 references
public override void OnEpisodeBegin()
{
    // Set the starting positions for the player and ball
    transform.localPosition = new Vector3(5.52f, 0.5f, 0f);
    ball.transform.localPosition = new Vector3(0f, 0.5f, 0f);
    playerTransform.localPosition = new Vector3(-5.51f, 0.5f, 0f);

    // Randomly set the direction and speed of the ball
    float randomNum = Random.Range(-0.5f, 0.5f);
    if (randomNum > 0)
        randomNum = 0.5f;
    else
        randomNum = -0.5f;
    ball.speed = 3.5f;
    ball.direction = new Vector3(-0.5f, 0, randomNum).normalized;
}
```

The training was done in parallel to train the AI faster and instead of playing against another AI or player we made the AI play against itself with a wall on the other side.



Beginning of Training

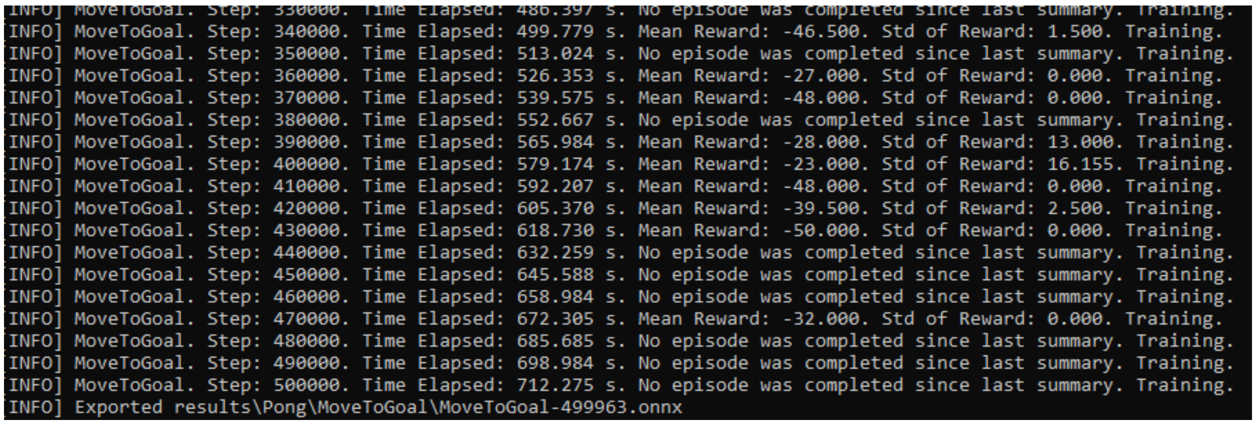


```
      Behavioral_cloning:     None
[INFO] MoveToGoal. Step: 10000. Time Elapsed: 21.183 s. Mean Reward: -49.724. Std of Reward: 0.638. Training.
[INFO] MoveToGoal. Step: 20000. Time Elapsed: 36.058 s. Mean Reward: -49.393. Std of Reward: 1.520. Training.
[INFO] MoveToGoal. Step: 30000. Time Elapsed: 51.725 s. Mean Reward: -49.758. Std of Reward: 0.494. Training.
[INFO] MoveToGoal. Step: 40000. Time Elapsed: 67.475 s. Mean Reward: -49.346. Std of Reward: 0.917. Training.
[INFO] MoveToGoal. Step: 50000. Time Elapsed: 83.825 s. Mean Reward: -49.794. Std of Reward: 0.583. Training.
[INFO] MoveToGoal. Step: 60000. Time Elapsed: 98.856 s. Mean Reward: -49.621. Std of Reward: 0.552. Training.
[INFO] MoveToGoal. Step: 70000. Time Elapsed: 114.175 s. Mean Reward: -49.345. Std of Reward: 1.239. Training.
[INFO] MoveToGoal. Step: 80000. Time Elapsed: 129.073 s. Mean Reward: -49.586. Std of Reward: 0.929. Training.
[INFO] MoveToGoal. Step: 90000. Time Elapsed: 144.406 s. Mean Reward: -49.355. Std of Reward: 1.556. Training.
[INFO] MoveToGoal. Step: 100000. Time Elapsed: 161.258 s. Mean Reward: -49.818. Std of Reward: 0.386. Training.
[INFO] MoveToGoal. Step: 110000. Time Elapsed: 176.908 s. Mean Reward: -49.458. Std of Reward: 0.815. Training.
[INFO] MoveToGoal. Step: 120000. Time Elapsed: 191.852 s. Mean Reward: -49.259. Std of Reward: 1.481. Training.
[INFO] MoveToGoal. Step: 130000. Time Elapsed: 206.999 s. Mean Reward: -49.640. Std of Reward: 0.625. Training.
[INFO] MoveToGoal. Step: 140000. Time Elapsed: 222.509 s. Mean Reward: -49.091. Std of Reward: 1.535. Training.
[INFO] MoveToGoal. Step: 150000. Time Elapsed: 237.605 s. Mean Reward: -49.136. Std of Reward: 1.179. Training.
[INFO] MoveToGoal. Step: 160000. Time Elapsed: 253.620 s. Mean Reward: -49.133. Std of Reward: 1.147. Training.
[INFO] MoveToGoal. Step: 170000. Time Elapsed: 268.006 s. Mean Reward: -48.600. Std of Reward: 1.625. Training.
[INFO] MoveToGoal. Step: 180000. Time Elapsed: 282.022 s. Mean Reward: -47.545. Std of Reward: 1.559. Training.
```

In the beginning of training, we started the ball at a lower speed than what would happen in a normal game to allow faster training for the AI. The plan was to gradually increase the speed of the ball and resume training after they got better at the previous speed. In the picture above you can slowly see an increase in the average reward the agent receives per 10000 steps.

Middle of Training

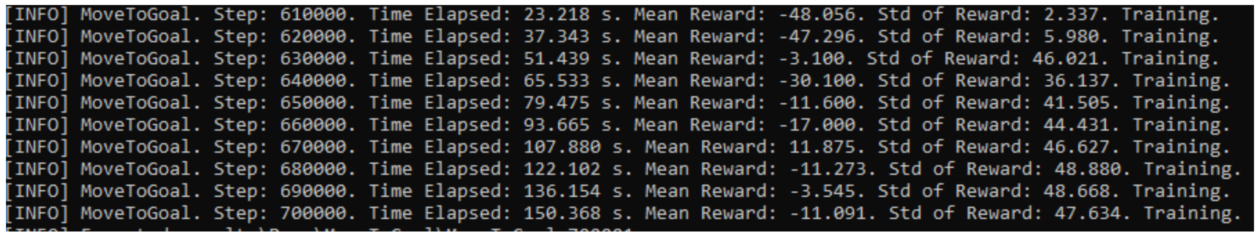
In the image below, is somewhere in the middle of training at 500000 steps. We increased the ball speed a little more and you can see that the AI was already learning much better. You can tell they are performing optimally because there was a lot of “No episode was completed since the last summary” which meant that the agent was hitting the ball back until the max episode step was reached. The AI agent did not let the ball pass them at this time.



```
[INFO] MoveToGoal. Step: 330000. Time Elapsed: 486.397 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 340000. Time Elapsed: 499.779 s. Mean Reward: -46.500. Std of Reward: 1.500. Training.
[INFO] MoveToGoal. Step: 350000. Time Elapsed: 513.024 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 360000. Time Elapsed: 526.353 s. Mean Reward: -27.000. Std of Reward: 0.000. Training.
[INFO] MoveToGoal. Step: 370000. Time Elapsed: 539.575 s. Mean Reward: -48.000. Std of Reward: 0.000. Training.
[INFO] MoveToGoal. Step: 380000. Time Elapsed: 552.667 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 390000. Time Elapsed: 565.984 s. Mean Reward: -28.000. Std of Reward: 13.000. Training.
[INFO] MoveToGoal. Step: 400000. Time Elapsed: 579.174 s. Mean Reward: -23.000. Std of Reward: 16.155. Training.
[INFO] MoveToGoal. Step: 410000. Time Elapsed: 592.207 s. Mean Reward: -48.000. Std of Reward: 0.000. Training.
[INFO] MoveToGoal. Step: 420000. Time Elapsed: 605.370 s. Mean Reward: -39.500. Std of Reward: 2.500. Training.
[INFO] MoveToGoal. Step: 430000. Time Elapsed: 618.730 s. Mean Reward: -50.000. Std of Reward: 0.000. Training.
[INFO] MoveToGoal. Step: 440000. Time Elapsed: 632.259 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 450000. Time Elapsed: 645.588 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 460000. Time Elapsed: 658.984 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 470000. Time Elapsed: 672.305 s. Mean Reward: -32.000. Std of Reward: 0.000. Training.
[INFO] MoveToGoal. Step: 480000. Time Elapsed: 685.685 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 490000. Time Elapsed: 698.984 s. No episode was completed since last summary. Training.
[INFO] MoveToGoal. Step: 500000. Time Elapsed: 712.275 s. No episode was completed since last summary. Training.
[INFO] Exported results\Pong\MoveToGoal\MoveToGoal-499963.onnx
```

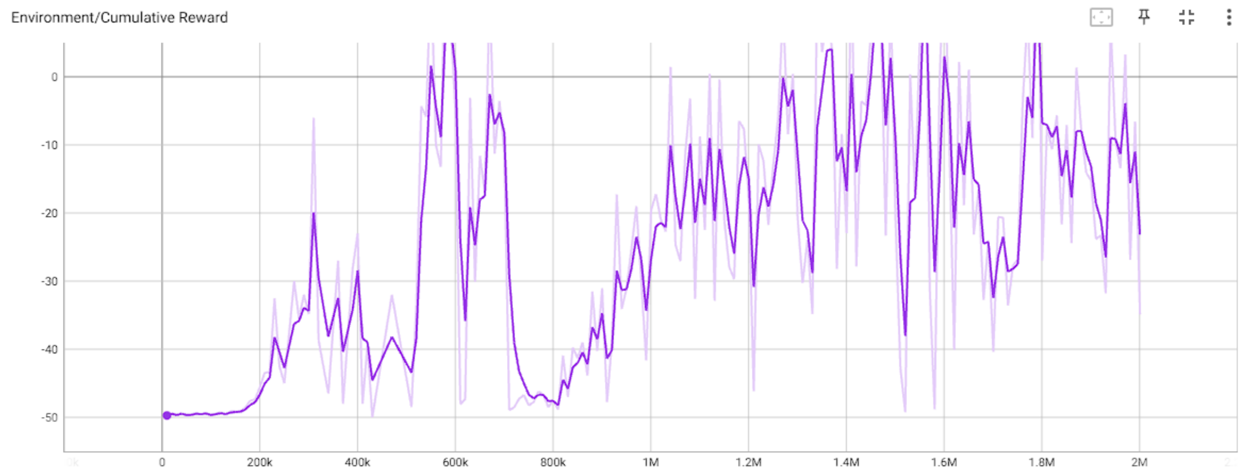
End of Training

This is the data closer to the end of training before we trained the AI at the actual game speed. It was a little harder for the AI to adjust itself for this increase in speed and at first the average reward dropped significantly, but in the end it was able to climb up the rewards ladder again.



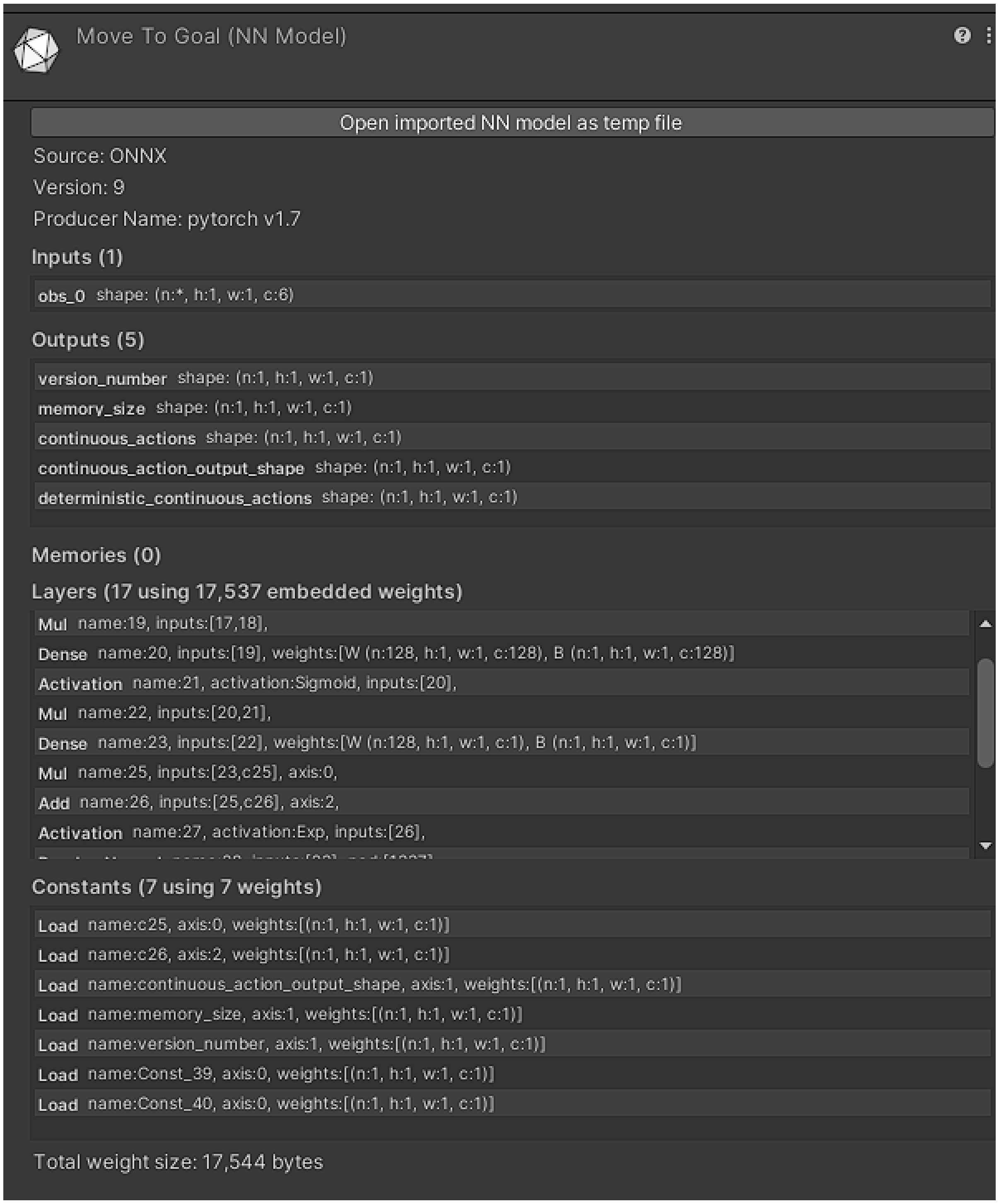
```
[INFO] MoveToGoal. Step: 610000. Time Elapsed: 23.218 s. Mean Reward: -48.056. Std of Reward: 2.337. Training.
[INFO] MoveToGoal. Step: 620000. Time Elapsed: 37.343 s. Mean Reward: -47.296. Std of Reward: 5.980. Training.
[INFO] MoveToGoal. Step: 630000. Time Elapsed: 51.439 s. Mean Reward: -3.100. Std of Reward: 46.021. Training.
[INFO] MoveToGoal. Step: 640000. Time Elapsed: 65.533 s. Mean Reward: -30.100. Std of Reward: 36.137. Training.
[INFO] MoveToGoal. Step: 650000. Time Elapsed: 79.475 s. Mean Reward: -11.600. Std of Reward: 41.505. Training.
[INFO] MoveToGoal. Step: 660000. Time Elapsed: 93.665 s. Mean Reward: -17.000. Std of Reward: 44.431. Training.
[INFO] MoveToGoal. Step: 670000. Time Elapsed: 107.880 s. Mean Reward: 11.875. Std of Reward: 46.627. Training.
[INFO] MoveToGoal. Step: 680000. Time Elapsed: 122.102 s. Mean Reward: -11.273. Std of Reward: 48.880. Training.
[INFO] MoveToGoal. Step: 690000. Time Elapsed: 136.154 s. Mean Reward: -3.545. Std of Reward: 48.668. Training.
[INFO] MoveToGoal. Step: 700000. Time Elapsed: 150.368 s. Mean Reward: -11.091. Std of Reward: 47.634. Training.
```

Rewards Graph



- To 500k was lowest speed
- To 600k was the next speed
- To 700k was next speed
- To 1M was next speed
- To the end was the fastest speed

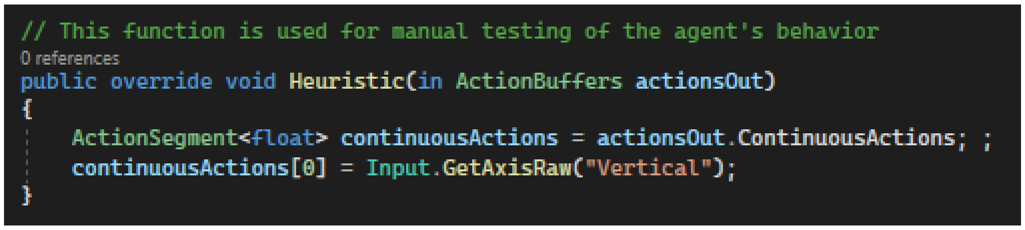
In this graph you can see what I described above, in the lowest speed the AI was able to do what it needed to do fairly quickly and you can see the proof in the graph because that is where it shot up the fastest. The next speed increase led to a big drop but quickly the AI learned to adjust and went up again in rewards. This is when we understood better how it learned and adjusted the speed accordingly which is why you do not see any more dramatic drops like you did after the first training session. Overall the graph is gradually increasing in the amount of rewards and you can see that the AI is improving. After completing all the training we receive a neural network model which is essentially the brain of the AI agent. The brain has inputs, outputs, layers and constants.



```
Move To Goal (NN Model)

Open imported NN model as temp file
Source: ONNX
Version: 9
Producer Name: pytorch v1.7
Inputs (1)
obs_0  shape: (n:*, h:1, w:1, c:6)
Outputs (5)
version_number  shape: (n:1, h:1, w:1, c:1)
memory_size  shape: (n:1, h:1, w:1, c:1)
continuous_actions  shape: (n:1, h:1, w:1, c:1)
continuous_action_output_shape  shape: (n:1, h:1, w:1, c:1)
deterministic_continuous_actions  shape: (n:1, h:1, w:1, c:1)
Memories (0)
Layers (17 using 17,537 embedded weights)
Mul  name:19, inputs:[17,18],
Dense  name:20, inputs:[19], weights:[W (n:128, h:1, w:1, c:128), B (n:1, h:1, w:1, c:128)]
Activation  name:21, activation:Sigmoid, inputs:[20],
Mul  name:22, inputs:[20,21],
Dense  name:23, inputs:[22], weights:[W (n:128, h:1, w:1, c:1), B (n:1, h:1, w:1, c:1)]
Mul  name:25, inputs:[23,c25], axis:0,
Add  name:26, inputs:[25,c26], axis:2,
Activation  name:27, activation:Exp, inputs:[26],
Constants (7 using 7 weights)
Load  name:c25, axis:0, weights:[(n:1, h:1, w:1, c:1)]
Load  name:c26, axis:2, weights:[(n:1, h:1, w:1, c:1)]
Load  name:continuous_action_output_shape, axis:1, weights:[(n:1, h:1, w:1, c:1)]
Load  name:memory_size, axis:1, weights:[(n:1, h:1, w:1, c:1)]
Load  name:version_number, axis:1, weights:[(n:1, h:1, w:1, c:1)]
Load  name:Const_39, axis:0, weights:[(n:1, h:1, w:1, c:1)]
Load  name:Const_40, axis:0, weights:[(n:1, h:1, w:1, c:1)]
Total weight size: 17,544 bytes
```

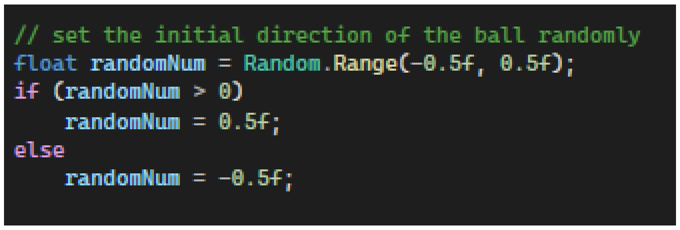
3. **Game Execution**

Other than the AI portion we had to execute a game for the AI to train and play in. We already know how the AI moves, and the way the player moves is very similar. The player moves using “W’ and ‘S’ keys to move their paddle up and down and is done using this code below.



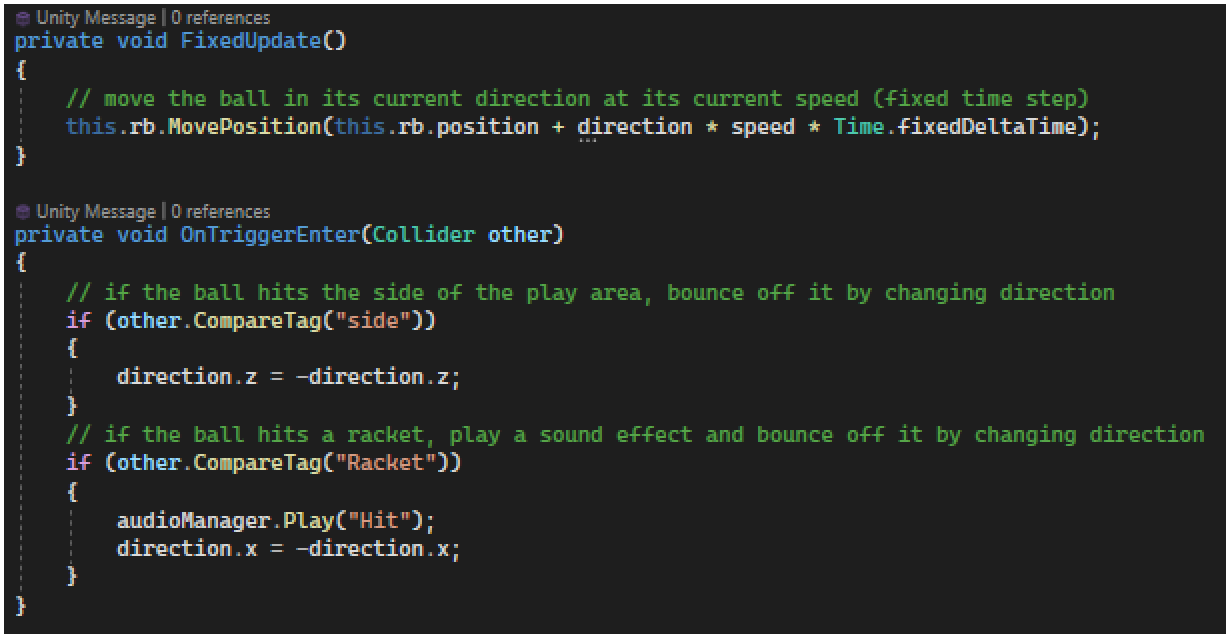
```
// This function is used for manual testing of the agent's behavior
0 references
public override void Heuristic(in ActionBuffers actionsOut)
{
    ActionSegment<float> continuousActions = actionsOut.ContinuousActions; ;
    continuousActions[0] = Input.GetAxisRaw("Vertical");
}
```

The ball movement is the next thing we handle and is very simple. The ball starts with a random direction towards the player and does so using the Random class in Unity.



```
// set the initial direction of the ball randomly
float randomNum = Random.Range(-0.5f, 0.5f);
if (randomNum > 0)
    randomNum = 0.5f;
else
    randomNum = -0.5f;
```

Then the only thing left to do is move the ball using its Rigidbody component to move and to change the direction when it hits a paddle or a wall. It is done so with the code below.



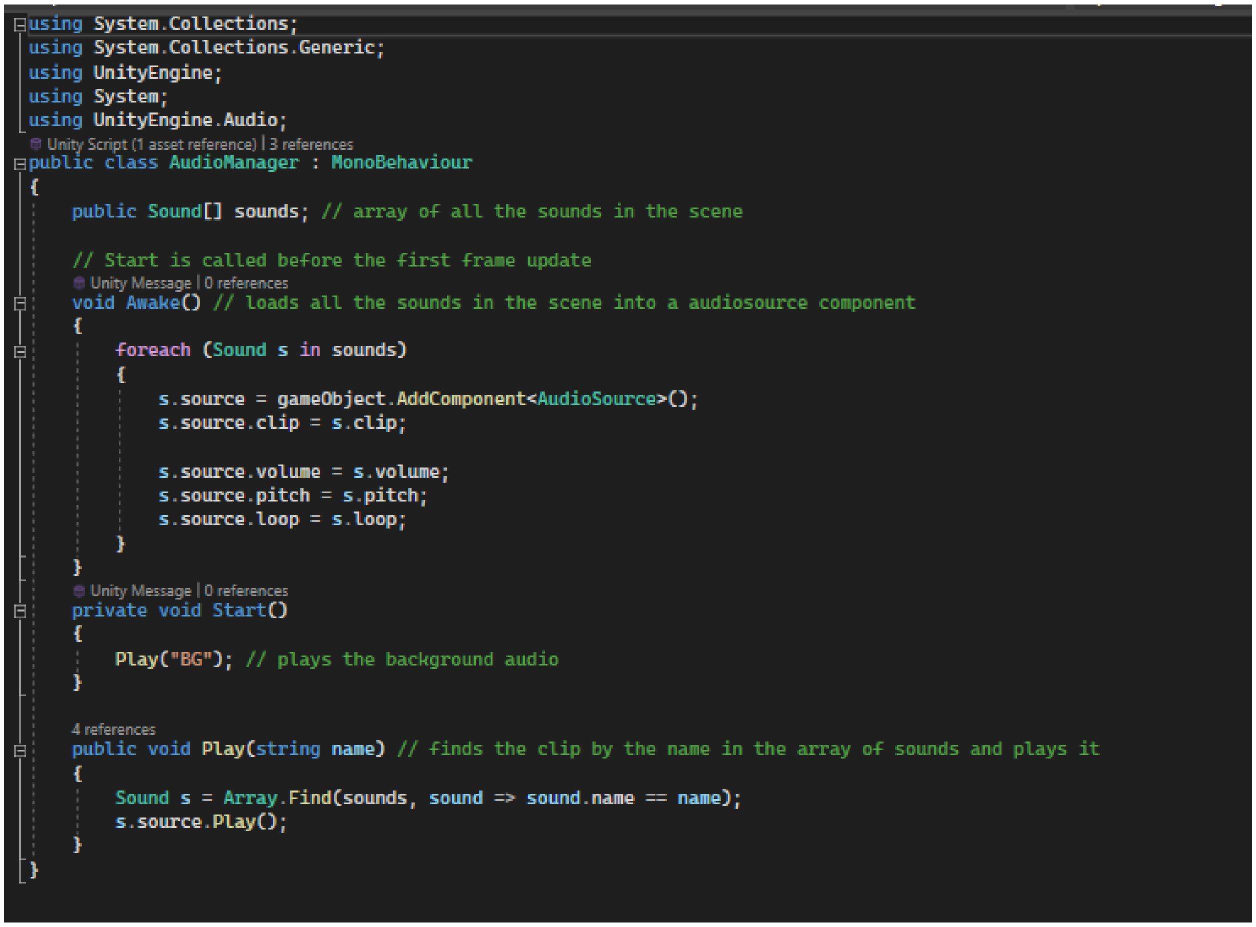
```
Unity Message | 0 references
private void FixedUpdate()
{
    // move the ball in its current direction at its current speed (fixed time step)
    this.rb.MovePosition(this.rb.position + direction * speed * Time.fixedDeltaTime);
}

Unity Message | 0 references
private void OnTriggerEnter(Collider other)
{
    // if the ball hits the side of the play area, bounce off it by changing direction
    if (other.CompareTag("side"))
    {
        direction.z = -direction.z;
    }
    // if the ball hits a racket, play a sound effect and bounce off it by changing direction
    if (other.CompareTag("Racket"))
    {
        audioManager.Play("Hit");
        direction.x = -direction.x;
    }
}
```

4. **What makes your game interesting to the player?**

The game is interesting to the player because it is known that a programmer did not have to tell the AI how to act. The AI learns what to do based on the reinforcements learning system with positive and negative rewards. The future plan is to have different brains trained at varied time lengths to create different difficulties for the player. The longer the AI trains, the harder it will be.

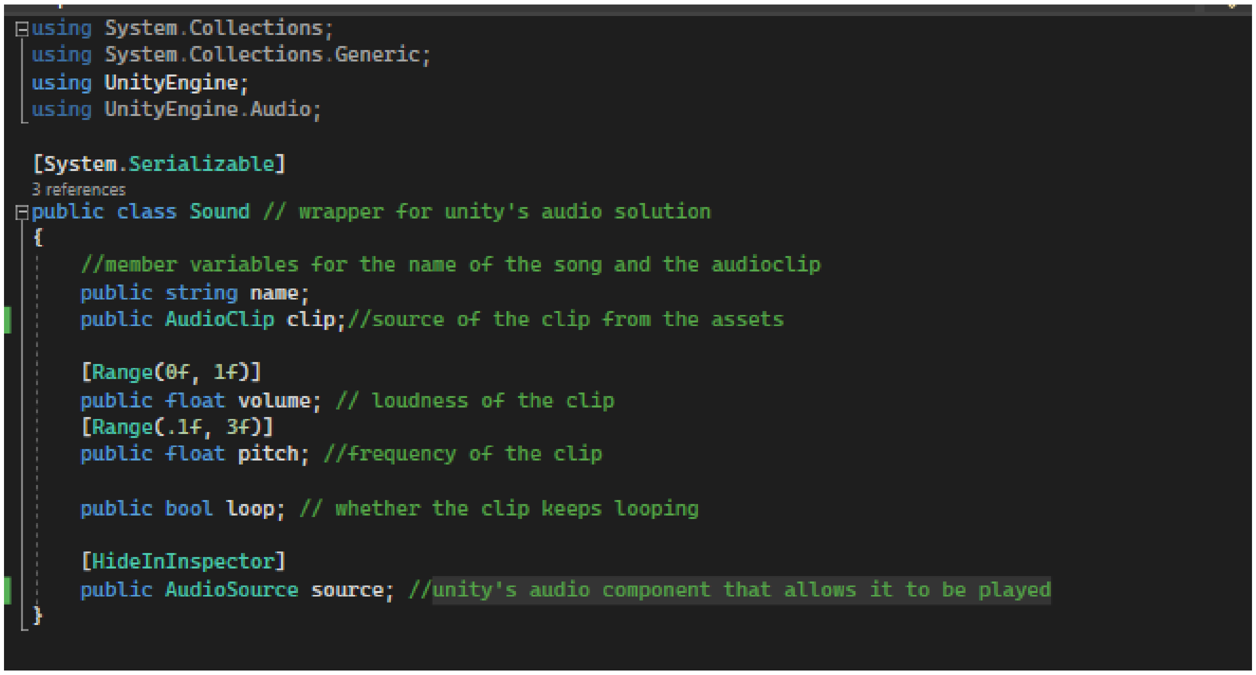
The game is also interesting because we have background music, sound effects, and textures that add juice to the game. When the player plays the game, there is a scoring system that determines who is the winner. In terms of the game mechanic, normal pong would be too boring which is why we made the pong ball move faster as time goes on to make it difficult for both the player and the AI.

For music we added the audiomanager script along with the sound script to control the music more effectively. The audiomanager as its name suggests manages all the audio clips in the current scene as well as plays any given one based on the Play function that finds the audio clip from the array of all the clips in the scene and plays that one.



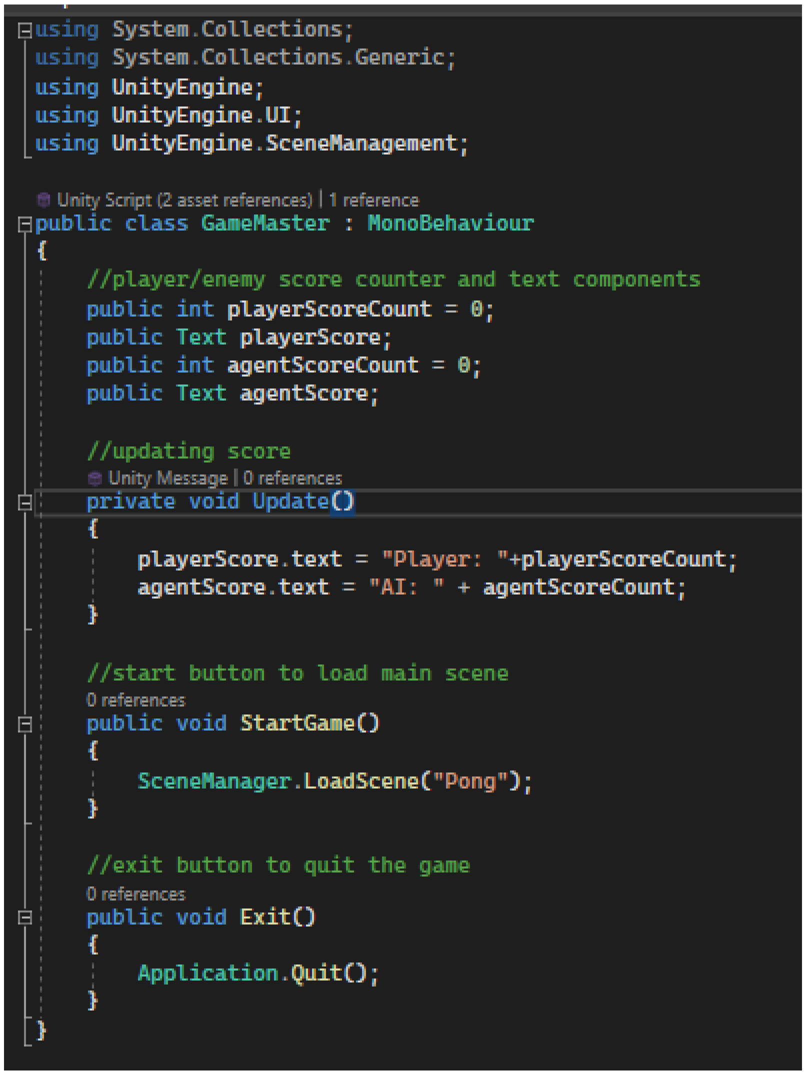
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using UnityEngine.Audio;
Unity Script (1 asset reference) | 3 references
public class AudioManager : MonoBehaviour
{
    public Sound[] sounds; // array of all the sounds in the scene

    // Start is called before the first frame update
    Unity Message | 0 references
    void Awake() // loads all the sounds in the scene into a audiosource component
    {
        foreach (Sound s in sounds)
        {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;

            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
        }
    }
    Unity Message | 0 references
    private void Start()
    {
        Play("BG"); // plays the background audio
    }

    4 references
    public void Play(string name) // finds the clip by the name in the array of sounds and plays it
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        s.source.Play();
    }
}
```

The sounds script is a wrapper created for unity’s audio solution. It helps us to simplify the code and is easy to implement in our game. The sound class has a name so that it can be searched by the audiomanager by it, a audioClip component that takes in the raw mp3 file to use from the assets folder, volume, pitch, and loop to set these values in the game, and lastly the audiosource which is the unity's audio component that allows it to be played.



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

[System.Serializable]
3 references
public class Sound // wrapper for unity's audio solution
{
    //member variables for the name of the song and the audioclip
    public string name;
    public AudioClip clip;//source of the clip from the assets

    [Range(0f, 1f)]
    public float volume; // loudness of the clip
    [Range(.1f, 3f)]
    public float pitch; //frequency of the clip

    public bool loop; // whether the clip keeps looping

    [HideInInspector]
    public AudioSource source; //unity's audio component that allows it to be played
}
```

The final script that we added for controlling the game was the gamemaster script. This script controlled everything from the scene switching from the main menu to the game scene, to handling the player and enemy's score value and text component as well. This script was constantly updating to keep the score correct at all times.



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

Unity Script (2 asset references) | 1 reference
public class GameMaster : MonoBehaviour
{
    //player/enemy score counter and text components
    public int playerScoreCount = 0;
    public Text playerScore;
    public int agentScoreCount = 0;
    public Text agentScore;

    //updating score
    Unity Message | 0 references
    private void Update()
    {
        playerScore.text = "Player: "+playerScoreCount;
        agentScore.text = "AI: " + agentScoreCount;
    }

    //start button to load main scene
    0 references
    public void StartGame()
    {
        SceneManager.LoadScene("Pong");
    }

    //exit button to quit the game
    0 references
    public void Exit()
    {
        Application.Quit();
    }
}
```

5. **Graphics, animation and sound effects should be added to the game**

No animations were required for this game. The game has a soccer theme with the arena looking like a soccer field and the ball as a soccer ball. For sound effects, there is background music, sounds for when the ball hits the paddle, and when a point is won or lost.

References

https://pixabay.com/ for audio clips

https://www.iconarchive.com/ for icons and images

https://github.com/Unity-Technologies/ml-agents for the setup of the ml agents
https://www.youtube.com/watch?v=zPFU30tbyKs&t=324s - help with the setup of the ml agents in unity
https://www.youtube.com/watch?v=supqT7kqpEI&t=213s - help with examples of setting up ml agents in unity