Proyecto SimRacing

Desarrollo de Aplicaciones Multiplataforma

Alumno: Blas Fernández González

Tutor: Jorge López Fernández

ÍNDICE

1 INTRODUCCIÓN

1.1 MOTIVO DE LA ELECCIÓN

He elegido este proyecto ya que me llama bastante la atención el **diseño de videojuegos**, y ya que no hemos visto nada relacionado con las **IA** en PMDM he decidido investigar sobre ello.

1.2 DESCRIPCIÓN Y OBJETIVOS GENERALES

Este proyecto se basará principalmente en la **investigación e implementación** de una IA usando Machine Learning en diferentes escenarios **en Unity 3D** donde dicha IA aprenderá a moverse por sí sola.

Ya que el proyecto consta de 40 horas nos enfocaremos principalmente en el funcionamiento de la IA, y posteriormente, en otros detalles de nuestro juego como pueden ser menús para seleccionar el entorno, opciones y sonidos entre otros en función del tiempo disponible.

Si diese tiempo, se implementaría un sistema de conducción para que el jugador pueda competir contra esta IA.

1.3 POSIBLES APLICACIONES PRÁCTICAS

La principal aplicación práctica de este proyecto es observar e investigar sobre el funcionamiento del Machine Learning, y posteriormente se podría implementar como un simulador de carreras para publicar en plataformas como Steam o Itch.io.

2 INVESTIGACIÓN SOBRE LA IA EN VIDEOJUEGOS

2.1 DEFINICIÓN DE LA IA EN VIDEOJUEGOS

La **Inteligencia Artificial** (IA) en videojuegos son todas las **técnicas usadas para diseñar** el **comportamiento** de los **personajes** controlados por el juego (NPCs), o incluso el **diseño de mapas y niveles**.

2.2 TIPOS DE IA

Existen varios tipos de IA, que se diferencian en cómo procesan la información y toman decisiones. Los **tipos de IA más comunes** en videojuegos son:

- **IA basada en reglas**: Reglas predefinidas y condiciones para determinar la conducta de los NPCs.
- **Máquinas de estados finitos**: Conductas de NPCs con patrones predefinidos. [Video]
- IA de búsqueda de rutas: Determina la ruta óptima para los NPCs de navegar por los mapeados del juego.
- I**A con aprendizaje automático (Machine Learning)**: Adapta la conducta y creación de decisiones basándose en las experiencias anteriores, entrenamiento y exposición a datos.
- **Árboles de comportamiento**: Ofrece flexibilidad y permite realizar conductas complejas a la IA adaptándose a las condiciones cambiantes. [Video]

- **Aprendizaje por refuerzo**: Recibe un aprendizaje basándose en penalizaciones y recompensas según las acciones del jugador y ajusta su conducta para maximizar las recompensas. [Video]

2.3 EJEMPLOS DE IMPLEMENTACIÓN EN VIDEOJUEGOS

La implementación de IA en juegos es un ejemplo común de cómo la IA puede mejorar la experiencia de juego. Algunos ejemplos pueden ser:

- **Alien Isolation**: El principal enemigo (**Xenomorfo**) tiene una IA que aprende y 'mejora' en base a las acciones del jugador. Conforme el juego avanza su IA desbloquea ciertos patrones que antes estaban limitados para que dé la apariencia de que está mejorando para no caer en los mismos errores.

- **Forza Horizon**: La IA que usa Forza se basa en los '**Drivatars**'. Este tipo de IA desarrollada especialmente para esta serie de juegos se basa en el aprendizaje de nuestra conducción, manías y errores (entre otras) para replicar nuestro comportamiento en partidas ajenas de la manera más parecida posible teniendo en cuenta algunas 'restricciones'.

- **Gran Turismo**: La IA usada en la saga Gran Turismo es denominada '**GT Sophy**' y fue desarrollada por Sony. Su principal objetivo es darle un reto al jugador ya que su mayor dificultad está entrenada para ponerle difícil por no decir casi imposible la partida al jugador.

- **Hello Neighbor**: En el primer juego, el principal enemigo aprende de tus fallos poniendo trampas allá donde te eliminó en la partida anterior, además si el jugados comete o repite los mismos patrones varias veces la IA rondará más por esa 'ruta'. En el segundo juego, la IA te persigue de manera 'discreta' durante todo el juego aprendiendo sobre todo lo que haces, lo cual significa que si usas un objeto en concreto para realizar alguna acción la IA podrá usar ese objeto en tu contra.

Todas estas IAs hacen **implementan varios tipos** diferentes **de IA** como pueden ser Machine Learning, Pathfinding u otros, dependiendo de cada juego y el resultado que se desea obtener.

Video interesante: [AI and Game Design | The History of Artificial Intelligence In Video Games]

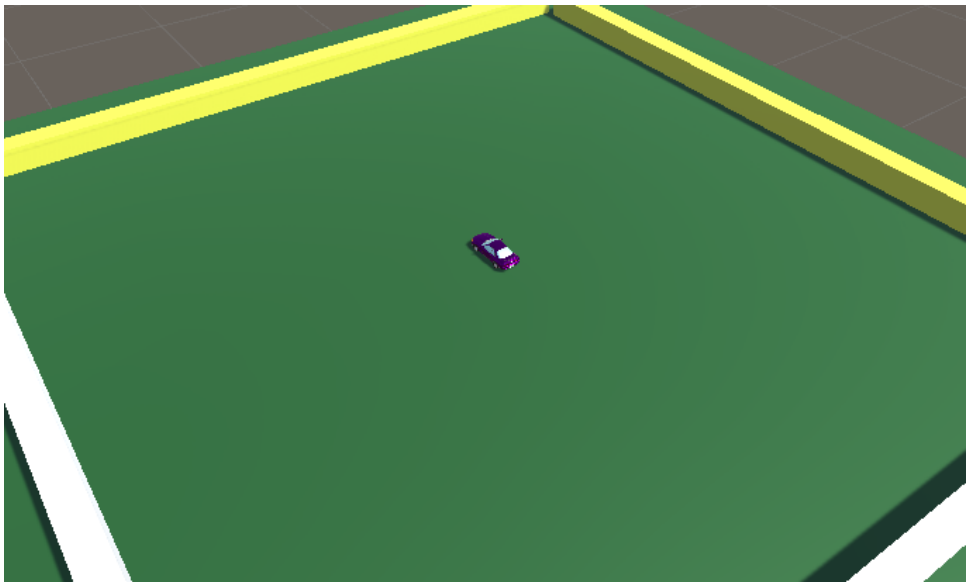
3 IMPLEMENTACIÓN DE LA IA

3.1 PREPARACIÓN

Para comenzar abriremos Unity Hub y crearemos un nuevo proyecto 3D el cuál llamaremos ‘SimRacing’.

Una vez que se haya creado nuestro proyecto procederemos a implementar los paquetes que necesitaremos, en nuestro caso ‘ML Agents’.

A continuación, crearemos una escena la cuál llamaremos ‘Area1’ en la cual crearemos un entorno sencillo con 4 paredes para que nuestra IA comience a aprender a moverse.



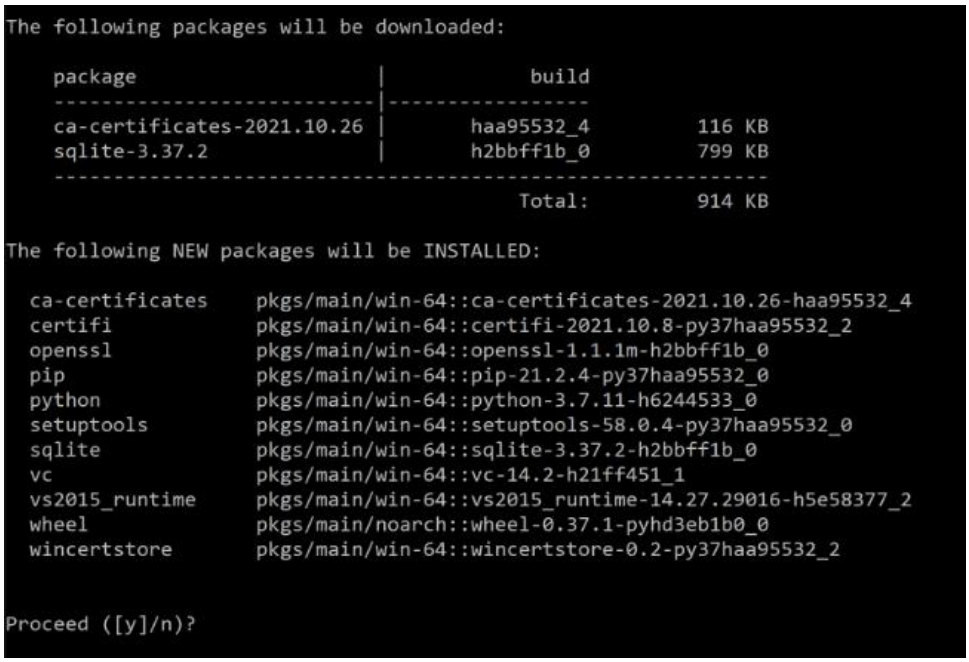
A continuación, instalaremos un programa parecido a Docker llamado Anaconda el cual nos ayudará a entrenar nuestra IA.

Para ello iremos a su página web y descargaremos el instalador que necesitemos, en mi caso será para Windows 10.

A continuación, iremos a la página de GitHub de Unity y descargaremos el repositorio en formato ‘.zip’, en mi caso usaré la versión 20.

Lo siguiente que haremos será abrir una línea de comandos de Anaconda y pondremos el siguiente comando y le daremos a la tecla ‘y’:

- conda create -n mlagents20 python=3.7: Este comando creará un entorno llamado ‘mlagents20’ usando la versión 3.7 de python.



Una vez se haya creado el entorno, lo activaremos usando el siguiente comando:

- conda activate mlagents20

Y acto seguido instalaremos ‘PyTorch’ con el siguiente comando, el cuál tardará un poco en terminar:

- pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html

Una vez haya terminado de instalar ‘PyTorch’ procederemos a instalar la versión de ml-agents con el siguiente comando:

- pip3 install mlagents==0.26.0

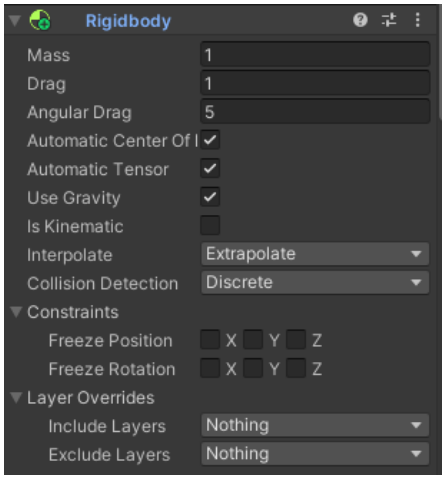
Con esto hecho procederemos a implementar la IA.

3.2 Diseño e Implementación de la IA

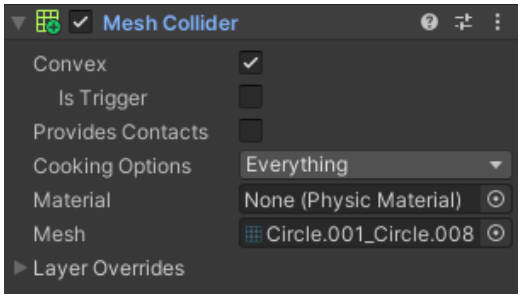
Añadiremos a nuestro vehículo un script llamado ‘NNArea’ el cual no es un script “corriente” ya que hace uso del paquete ML Agents instalado en Unity en vez de ‘MonoBehavior’ como de costumbre. Este script añadirá los siguientes componentes ya que le indicamos que serán requeridos:

```
[RequireComponent(typeof(Rigidbody))]
[RequireComponent(typeof(MeshCollider))]
[RequireComponent(typeof(DecisionRequester))]
Script de Unity (11 referencias de recurso) | 0 referencias
public class NNArea : Agent
```

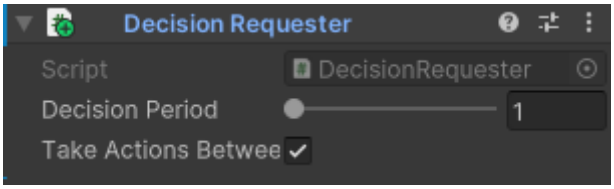
- **Rigidbody**: Estableceremos el ‘Drag’ a 1 y el ‘Angular Drag’ a 5.



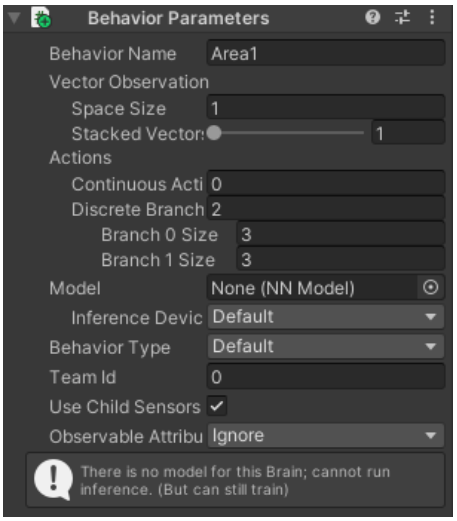
- **MeshCollider**: Activaremos el parámetro ‘Convex’.



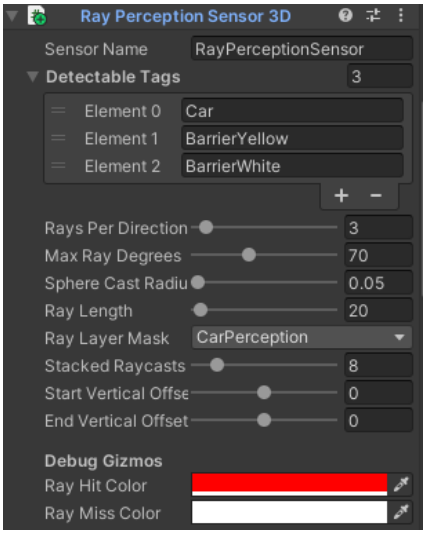
- **Decision Requester**: Estableceremos el ‘Decision Period’ a 1 (reducirá el “tiempo de reacción”).



- **Behaviour Parameters**: Estableceremos el ‘Behaviour Name’ a ‘Area 1’ (nombre de la ‘red’ a entrenar), añadiremos 2 acciones en ‘Discrete Branch’. La primera hará referencia al movimiento hacia delante, atrás o no hacer nada. La segunda será el movimiento hacia la izquierda, derecha o no hacer nada.



Y nosotros añadiremos un ‘Ray Perception Sensor 3D’ que añadirá los sensores necesarios a nuestro vehículo. Estableceremos 3 ‘Detectable Tags’ que serán las paredes amarillas, blancas y el coche, además de crear un layer llamado ‘CarPerception’ que incluirá a estos 3:



3.2.1 Script

Lo primero que haremos será definir una clase con las recompensas y penalizaciones que queramos:

```
public class RewardInfo
{
    public float mult_forward = 0.001f;
    public float barrier = -0.001f;
    public float car = -0.001f;
}
```

A continuación, inicializaremos los componentes necesarios a un valor por defecto:

```
public override void Initialize()
{
    rb = this.GetComponent<Rigidbody>();
    rb.drag = 1;
    rb.angularDrag = 5;
    rb.interpolation = RigidbodyInterpolation.Extrapolate;

    this.GetComponent<MeshCollider>().convex = true;

    this.GetComponent<DecisionRequester>().DecisionPeriod = 1;

    bnd = this.GetComponent<MeshRenderer>().bounds;

    recall_position = new Vector3(this.transform.position.x, this.transform.position.y, this.transform.position.z);
    recall_rotation = new Quaternion(this.transform.rotation.x, this.transform.rotation.y, this.transform.rotation.z, this.transform.rotation.w);
}
```

Lo siguiente que haremos será añadir el método 'OnEpisodeBegin()' que establecerá la posición, rotación y velocidad de los vehículos a su estado inicial cada vez que se inicie un episodio de aprendizaje:

```
public override void OnEpisodeBegin()
{
    rb.velocity = Vector3.zero;
    this.transform.position = recall_position;
    this.transform.rotation = recall_rotation;
}
```

A continuación, implementaremos el método 'OnActionReceived()' en el que indicaremos qué hacer en función de las acciones recibidas:

```
public override void OnActionReceived(ActionBuffers actions)
{
    if (isWheelsDown() == false)
        return;

    float mag = rb.velocity.sqrMagnitude;

    switch (actions.DiscreteActions.Array[0])
    {
        case 0:
            break;
        case 1:
            rb.AddRelativeForce(Vector3.back * Movespeed * Time.deltaTime, ForceMode.VelocityChange);
            break;
        case 2:
            rb.AddRelativeForce(Vector3.forward * Movespeed * Time.deltaTime, ForceMode.VelocityChange);
            AddReward(mag * rwd.mult_forward);
            break;
    }

    switch (actions.DiscreteActions.Array[1])
    {
        case 0:
            break;
        case 1:
            this.transform.Rotate(Vector3.up, -Turnspeed * Time.deltaTime);
            break;
        case 2:
            this.transform.Rotate(Vector3.up, Turnspeed * Time.deltaTime);
            break;
    }
}
```

- Si el vehículo no está tocando el terreno no se realizará ninguna acción.
- El primer switch indica las acciones de movimiento hacia adelante, atrás o no hacer nada que puede realizar el vehículo. En caso de moverse hacia delante se le recompensará con una pequeña cifra que ayudará a su aprendizaje.
- El segundo switch permitirá que el vehículo se mueva hacia los laterales o que por el contrario vaya recto.

Con el siguiente método podremos comprobar de que nuestro script funciona correctamente, es decir, nos permitirá movernos por el terreno haciendo uso de las teclas WASD:

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    actionsOut.DiscreteActions.Array[0] = 0;
    actionsOut.DiscreteActions.Array[1] = 0;

    float move = Input.GetAxis("Vertical");
    float turn = Input.GetAxis("Horizontal");

    if (move < 0)
        actionsOut.DiscreteActions.Array[0] = 1;
    else if (move > 0)
        actionsOut.DiscreteActions.Array[0] = 2;

    if (turn < 0)
        actionsOut.DiscreteActions.Array[1] = 1;
    else if (turn > 0)
        actionsOut.DiscreteActions.Array[1] = 2;
}
```

A continuación, definiremos las penalizaciones que se aplicarán cuando el vehículo choque contra algún muro o contra otro vehículo:

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("BarrierWhite") == true
        || collision.gameObject.CompareTag("BarrierYellow") == true)
    {
        AddReward(rwd.barrier);
    }
    else if (collision.gameObject.CompareTag("Car") == true)
    {
        AddReward(rwd.car);
    }
}
```

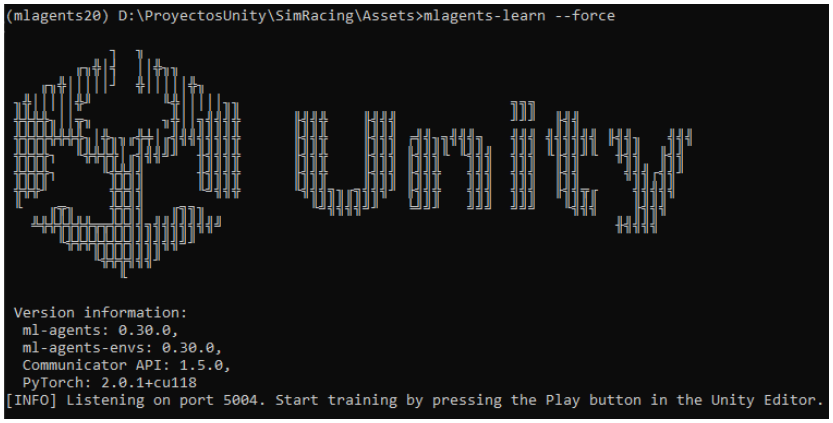
Por último, crearemos un método llamado ‘isWheelDown()’ que comprobará si el vehículo está o no tocando el suelo:

```
private bool isWheelsDown()
{
    return Physics.Raycast(this.transform.position, -this.transform.up, bnd.size.y * 0.55f);
}
```

3.3 ENTRENAMIENTO

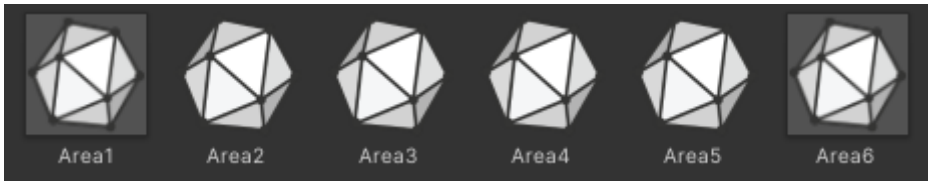
Para entrenar nuestra IA tendremos que usar el terminal de Anaconda y acceder a la ruta de nuestro proyecto, en concreto a la carpeta 'Assets', y ejecutamos el siguiente código y le daremos al botón play de Unity cuando nos aparezca la imagen de abajo:

- mlagents-learn –force

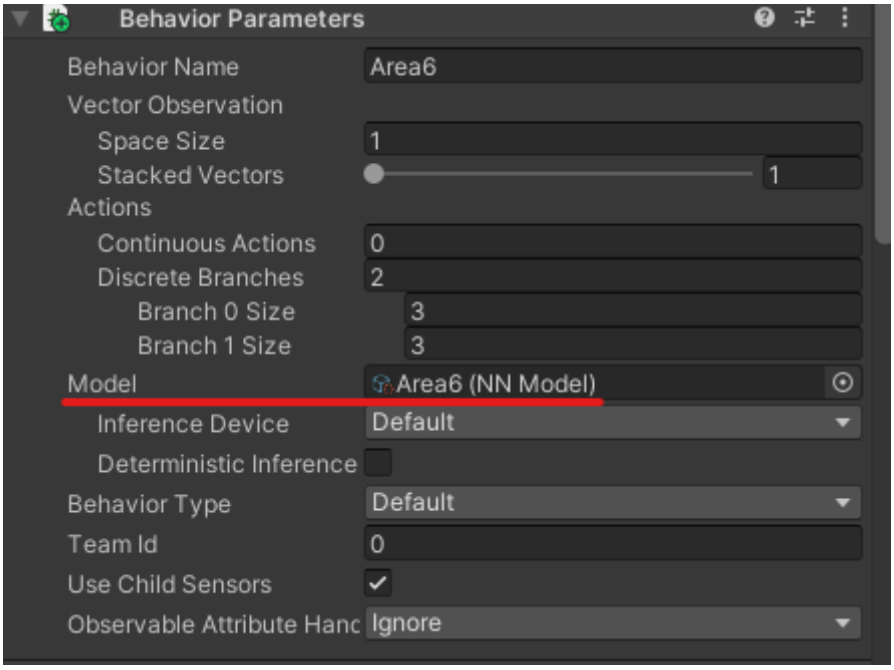


Por defecto esto realizará un entrenamiento de aproximadamente 15 minutos.

Una vez que haya terminado nos generará un archivo '.onnx' con el contenido aprendido como si de un cerebro se tratase:



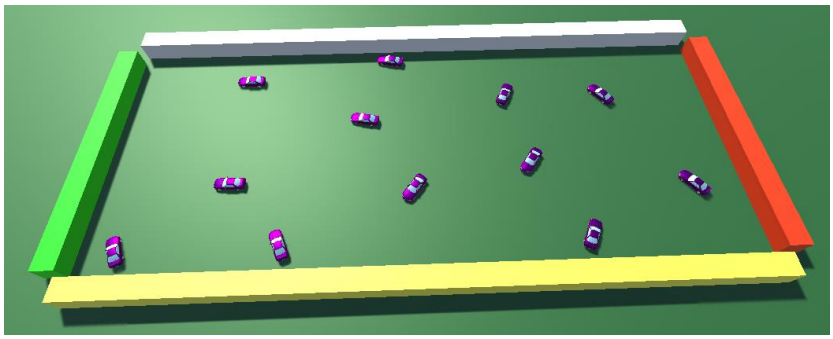
Si queremos comprobar qué es lo que ha aprendido nuestra IA tendremos que seleccionar el coche/coches y añadir este archivo '.onnx' en el siguiente campo:



Con esto hecho añadiremos 2 variantes de nuestro script. Una para que aprenda a moverse hacia adelante, y otra que aprenderá a conducir por un circuito.

Para que aprenda a moverse hacia adelante dejaremos el método 'OnCollisionEnter()' de la siguiente manera, lo cual añade una recompensa (pared verde) y una penalización (pared roja). De esta manera será capaz de aprender que debe dirigirse sólo hacia adelante:

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("BarrierWhite") == true
        || collision.gameObject.CompareTag("BarrierYellow") == true)
    {
        AddReward(rwd.barrier);
    }
    else if (collision.gameObject.CompareTag("Car") == true)
    {
        AddReward(rwd.car);
    }
    else if (collision.gameObject.CompareTag("Win") == true)
    {
        AddReward(rwd.win);
        if (doEpisodes == true)
            EndEpisode();
    }
    else if (collision.gameObject.CompareTag("Lose") == true)
    {
        AddReward(rwd.lose);
        if (doEpisodes == true)
            EndEpisode();
    }
}
```



Y, por último, para que los vehículos aprendan a moverse por un circuito modificaremos el mismo código de la siguiente manera:

```
private void OnCollisionEnter(Collision collision)
{
    float mag = collision.relativeVelocity.sqrMagnitude;

    if (collision.gameObject.CompareTag("BarrierWhite") == true
        || collision.gameObject.CompareTag("BarrierYellow") == true)
    {
        AddReward(mag * rwd.mult_barrier);
        if (doEpisodes == true)
            EndEpisode();
    }
    else if (collision.gameObject.CompareTag("Car") == true)
    {
        AddReward(mag * rwd.mult_car);
        if (doEpisodes == true)
            EndEpisode();
    }
}
```

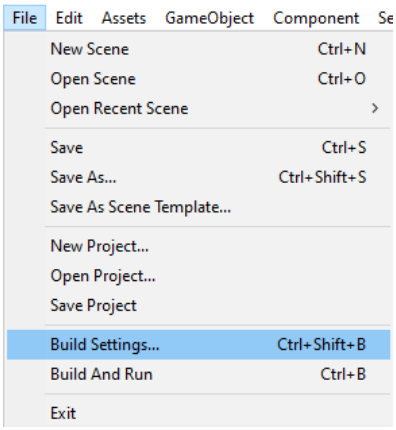
Para aumentar el tiempo de entrenamiento de nuestra IA tendremos que crear un archivo '.yaml' con la siguiente estructura en la carpeta 'Assets' de nuestro proyecto:

```yaml
behaviors:
  AreaEntrenamiento1Hora:
    trainer_type: ppo
    network_settings:
      hidden_units: 1024
      num_layers: 3
    reward_signals:
      extrinsic:
        network_settings:
          hidden_units: 1024
          num_layers: 3
    max_steps: 2000000
```

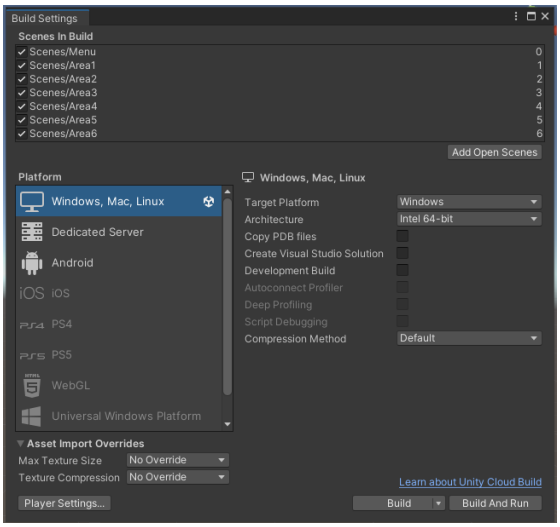
En este caso, 'max_steps' está configurado a 2000000, lo que es lo mismo, aproximadamente una hora de entrenamiento.

3.4 Exportar proyecto para Windows

Una vez tengamos el proyecto terminado procederemos a exportarlo para Windows. Para ello iremos al menú 'File → Build Settings...':



Esto nos abrirá la siguiente pestaña, en la cual tendremos que especificar qué escenas añadir a nuestro ‘juego’. Si fuese necesario podríamos modificar la configuración para exportarlo. Por último, le daremos al botón ‘Build’, donde tendremos que seleccionar la carpeta de salida:



Una vez terminado, podremos ejecutarlo haciendo doble clic sobre el ejecutable:

4 CONCLUSIONES

El objetivo principal de este proyecto era crear un simulador de carreras con IA contra la que el jugador pudiese competir. Pero dado a que he modificado todo el proyecto a última hora dado a algunos problemas técnicos y que no me terminaba de gustar el resultado, decidí centrarme en el entrenamiento de la IA con Machine Learning.

4.1 Resultados obtenidos

El resultado obtenido se queda corto para la idea de proyecto que tenía inicialmente, pero como ya he mencionado, no me terminó de gustar el resultado inicial que obtuve, con lo cual me centré en el desarrollo e investigación del Machine Learning ya que es un tipo de IA bastante curiosa.

4.2 Puntos pendientes y/o posibles mejoras

Este proyecto se ha basado principalmente en la investigación y desarrollo de una IA, pero me hubiese gustado implementar las funciones necesarias para que el jugador pudiese competir contra esta IA.

Algunos de esos puntos son:

- Implementar un menú de selección de vehículos (además de más modelos) dónde el jugador pudiese seleccionar qué coche usar.
- Implementar las físicas necesarias para que se pudiese conducir correctamente.
- Implementar animaciones de conducción (ruedas, volante, derrapes, etc.)
- Implementar una interfaz como la de los videojuegos de carreras con tacómetro, gasolina, estado de las ruedas, etc.

5 Recursos utilizados

5.1 Hardware

Para este proyecto he utilizado mi ordenador personal de sobremesa con las siguientes características:

- **CPU**: AMD Ryzen 5 3600 3.6 GHz
- **RAM**: 16 GB
- **GPU**: Nvidia GTX 1650 4GB GDDR6

5.2 Software

He utilizado **Unity 2022.2.3f1** para desarrollar este 'videojuego', además de los diferentes 'assets' utilizados como los modelos 3D de los circuitos y vehículo, la música utilizada, etc.

Además, he usado Anaconda, un programa parecido a Docker que nos permite instalar programas en un entorno virtual para ayudarnos con el desarrollo de nuestro proyecto.

También he usado el paquete 'ML Agents' del Package Manager de Unity.