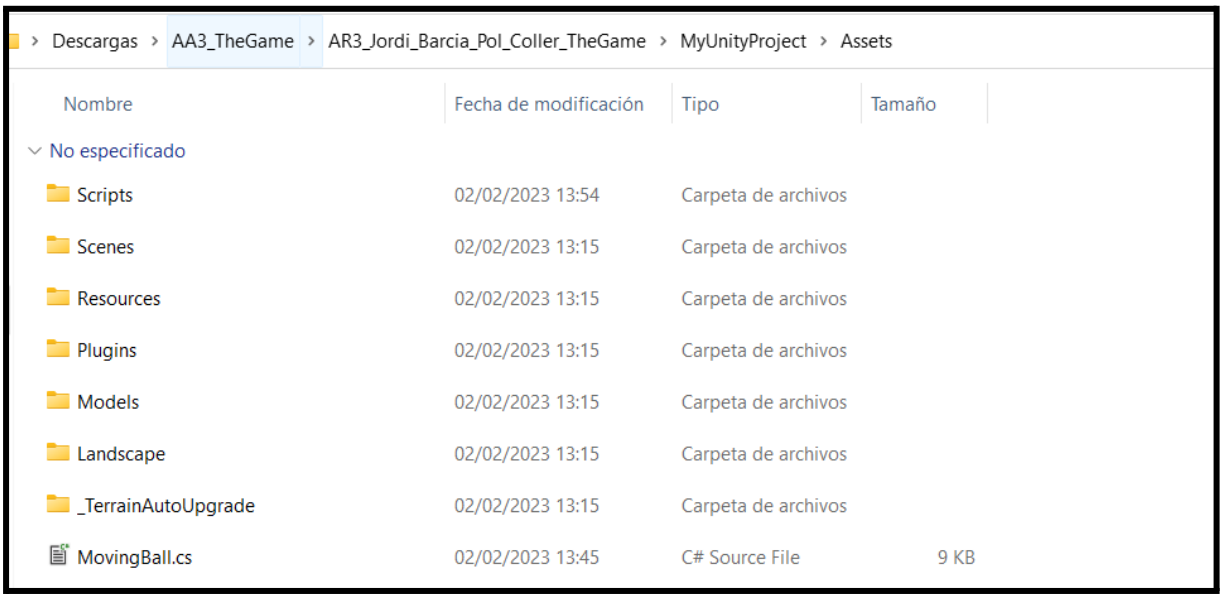
Jordi Barcia Peñaranda
Pol Coller Borja

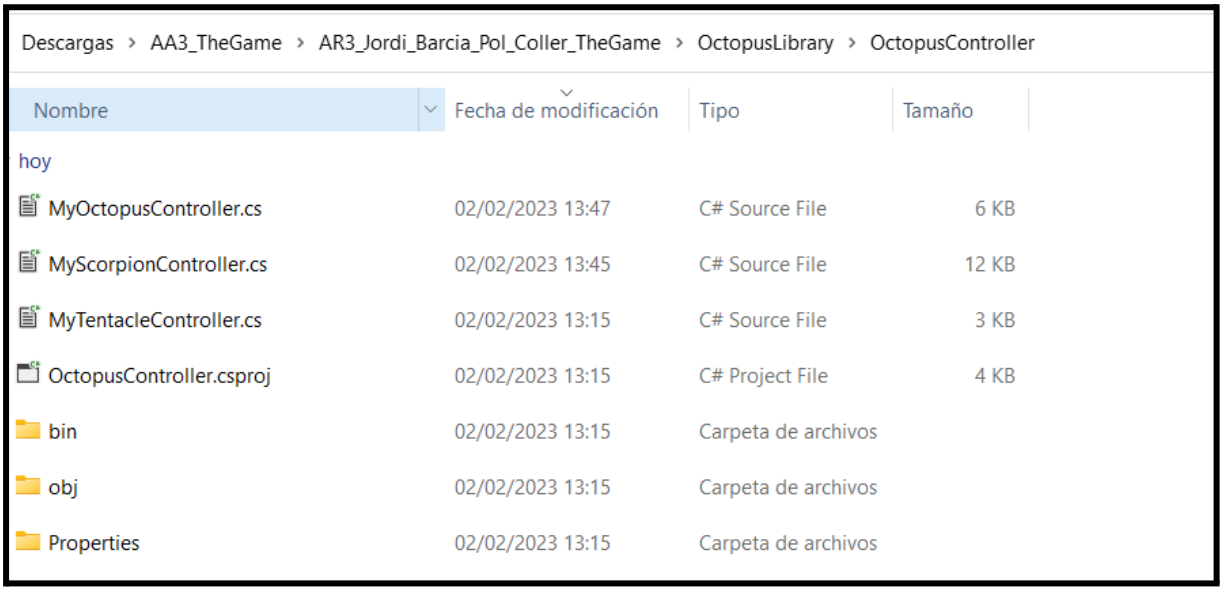
Ejercicio 0.

…\…\AR3_Jordi_Barcia_Pol_Coller_TheGame\MyUnityProject\Assets



En Assets encontrarás el script de MovingBall.cs.
Dentro de la carpeta Scripts encontrarás el script de IK_Scorpion.cs.

…\...\AR3_Jordi_Barcia_Pol_Coller_TheGame\OctopusLibrary\OctopusController



En la carpeta OctopusLibrary encontrarás el .sln para abrir la solución del proyecto.
En la carpeta OctopusController encontrarás los distintos archivos.

…\…\AR3_Jordi_Barcia_Pol_Coller_TheGame\Build
El ejecutable (Delivery3.exe) del proyecto está dentro de una carpeta llamada Build.

Ejercicio 1:
Del ejercicio 1 los apartados que están hechos son el 2, 3, 4, 5.
Los apartados 2,3,4 están hechos en el script MovingBall.cs
El readme del apartado 5 lo tienes más abajo en este documento.

Ejercicio 2:
Del ejercicio 2 los apartados que están hechos son el 1, 2, 3, 4, 5, 6.

En el apartado 1 tienes en el script de MovingBall.cs el código para aumentar o disminuir el slider llamado “Effect strength” que está en la UI. La posición de la cola se ajusta en función de un target que está al lado de la pelota, el movimiento del target está dentro del script MovingBall.cs, mientras que el movimiento de la cola se actualiza en el script MyScorpionController.cs, que encontrarás en la carpeta de la dll proporcionada.

Para el apartado 2 todo está contenido en el script de MovingBall.cs

Para el apartado 3 tienes la trayectoria definida dentro del script de MovingBall.cs al igual que la configuración de la tecla para mostrar la información adicional, pero los line renderers de cada trayectoria los encontrarás en la escena del proyecto.

Para el apartado 4 tienes las posiciones de las flechas definidas en el código, pero el Gameobject como tal de las flechas están puestos dentro del GameObject de la pelota.

Los readmes del apartado 5 y 6 los tienes más abajo en este documento.

Ejercicio 3:
Del ejercicio 3 los apartados que están hechos son el 1, 2, 4, 6.

Para el apartado 1 tienes todo el código en script MyScorpionController.cs que puedes encontrar en la carpeta de la dll proporcionada. Concretamente en la función updateLegPos().

Para el apartado 2 puedes cambiar de cámaras utilizando las teclas F1 y F2 para así poder ver el circuito de obstáculos. El código para el cambio de cámaras está en el script IK_Scorpion.cs al igual que el camino predefinido que seguirá. Los obstáculos están puestos en la escena.

Para el apartado 4 tienes el código dentro del script MyScorpionController.cs, que encontrarás en la carpeta de la dll proporcionada. Concretamente dentro de la función UpdateBodyPosition().

Para el apartado 6 como he comentado anteriormente puedes alternar entre las dos cámaras, si pones la cámara donde se ven los obstáculos presionando la tecla “P” puedes iniciar el movimiento del escorpión a través del recorrido. El código para la tecla esta en el script de IK_Scorpion.cs.

Ejercicio 4:
Del ejercicio 4 los apartados que están hechos son el 1.
Para el apartado 1 el movimiento de la cola, va en función de un target que se actualiza en el script de MovingBall.cs y está puesto como GameObject dentro de la escena, mientras que la posición de la cola se adapta dentro del script MyScorpionController.cs dentro de la carpeta de la dll proporcionada.

Ejercicio 5:
Del ejercicio 5 los apartados que están hechos son el 1, 3.
Para el apartado 1 lo encontrarás dentro del script MyOctopusController.cs, que se encuentra dentro de la carpeta de la dll proporcionada. Concretamente en la función update_ccd().

Para el apartado 3 hemos implementado una función dentro del mismo script donde bloqueamos los tentáculos para que siempre miren a cámara, esta función se llama LimitRotationOnXAxis().

Ejercicio 1.

El cálculo de la velocidad está implementado en la función CalculateVelocity()
Primero calculamos la distancia entre el target de la portería y el rigidbody de la pelota.

```
distBtRb = blueTarget.position - rb.position;
```

Seguidamente calculamos la velocidad (Vel). Al ser MRU la velocidad en los ejes X y Z queda constante.

```
vel = distBtRb.normalized * (Mathf.Sqrt((new Vector3(0, 0, shootForce).magnitude) * distBtRb.magnitude * 2));
```

Posteriormente calculamos el tiempo y la velocidad en el eje Y -> Vy = time * g/2 y le sumamos el valor ya calculado en el paso anterior de la velocidad en el eje Y a este resultado.

```
time = distBtRb.magnitude / (Mathf.Sqrt((new Vector3(0, 0, shootForce).magnitude) * distBtRb.magnitude * 2));

velY = Mathf.Abs(time * gravityF / 2) + vel.y;
vel = new Vector3(vel.x, velY, vel.z);
```

Finalmente en OnCollisionEnter() igualamos la velocidad del rigidbody de la pelota a esta velocidad calculada. Esta velocidad se actualiza constantemente en el Update().

Ejercicio 2

5.

Para poder calcular la velocidad angular lo primero que hacemos es calcular la distancia (distBtRb) que hay del target azul que está en la portería a la pelota. Una vez tenemos esta distancia calculamos la velocidad (vel), utilizando la distancia normalizada la multiplicamos por la raíz cuadrada de la magnitud de un vector la cual está siendo multiplicado a la vez por la magnitud de la distancia calculada anteriormente. Lo siguiente es calcular el tiempo, por lo tanto, lo que hacemos es una operacion similar a la anterior pero en vez de multiplicar dividimos.

```
distBtRb = blueTarget.position - rb.position;
vel = distBtRb.normalized * (Mathf.Sqrt((new Vector3(0, 0, shootForce).magnitude) * distBtRb.magnitude * 2));
time = distBtRb.magnitude / (Mathf.Sqrt((new Vector3(0, 0, shootForce).magnitude) * distBtRb.magnitude * 2));
```

Ahora calculamos el componente Y de la velocidad aplicando la siguiente fórmula:

```
velY = Mathf.Abs(time * gravityF / 2) + vel.y;
```

Con esto actualizamos los componentes de la velocidad sustituyendo el componente Y por el que hemos calculado.

```
vel = new Vector3(vel.x, velY, vel.z);
```

Teniendo ya la velocidad podemos empezar a hacer los cálculos para obtener la velocidad de rotación de la pelota.
Lo primero que haremos será calcular la distancia (distMagnus) que hay de la pelota a un YellowTarget puesto en la escena.

```
distMagnus = new Vector3(rb.position.x - yellowTarget.transform.position.x,
    rb.position.y - yellowTarget.transform.position.y, rb.position.z - yellowTarget.transform.position.z);
```

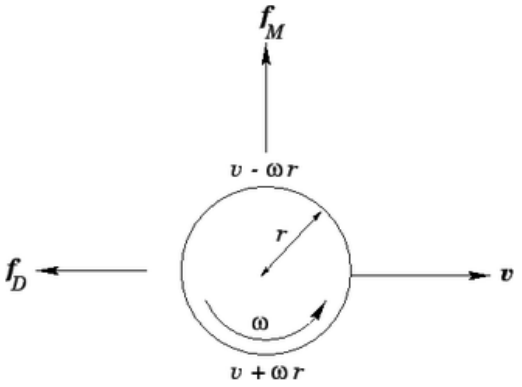
Con esta distancia calcularemos el radio utilizando el un Cross Product entre la distancia y la velocidad que hemos calculado anteriormente.

```
radius = Vector3.Cross(distMagnus, vel);
```

Este radio (radius) que hemos calculado nos servirá para finalmente poder obtener la velocidad angular (w) haciendo el Cross Product del radio con la vel calculada anteriormente.

```
w = Vector3.Cross(radius, vel);
```

6.



Magnus Force Diagram

$$\mathbf{f}_M = S(v)\,\boldsymbol{\omega} \times \mathbf{v}$$

Fórmula utilizada del Magnus Effect

Para aplicar el magnus lo primero que hacemos es calcular la distancia en función del target amarillo que esta puesto en la escena. Una vez hecho esto lo que hacemos es calcular el radio (radius), haciendo el Cross Product entre la distancia (distMagnus) y la velocidad (vel) calculada anteriormente. Una vez hecho esto pasamos al siguiente punto que es calcular la velocidad angular (w), haciendo el Cross Product del radio y la velocidad.

Una vez hecho todo esto aplicamos la función del Magnus effect donde effectMagnus es la cantidad de efecto que le daremos a la pelota dentro de la escena:

```
forceMagnus = effectMagnus * (Vector3.Cross(w, vel));
```

Que nos dará como resultado una fuerza que sumaremos a la velocidad que hemos calculado y nos dará la velocidad final del Magnus effect.

```
initVelBlueLine = vel + forceMagnus;
```

A pesar de no ser la fórmula que más se ajusta a la realidad, es buena para poder representar el concepto del Magnus, además es más fácil de calcular con lo cual genera menos problemas y nos ofrece un resultado prácticamente similar en el caso de esta práctica.