Pau Marín
Mariona Barranco
3 CDI

Animation Foundations. Delivery 3 "The game"

<u>Where to find stuff</u>

Exercise 1

1. IK_Scorpion.cs ➜ public void NotifyShoot()
   MovingBall.cs ➜ public void OnCollisionEnter()
2. MovingBall.cs ➜ void Update()
3. Slider_force (gameObject)
   ShootForce().cs
   IK_Scorpion.cs ➜ void Update() ➜ if(Input.GetKeyDown/Up)
   ShootingPhysics.cs
4. IK_Scorpion.cs ➜ public void NotifyStartWalk()
   MovingBall.cs ➜ OnStartWalk & ResetBall() & ResetTimer()

Exercise 2

1. Strength Slider
   MagnusEffectSlider.cs
   ShootingPhysics.cs ➜ Shoot()
2. ShootingPhysics.cs ➜ Shoot()
3. ShootingPhysics.cs ➜ DebugTrajectory() & DebugTrajectoryWithMagnus()
4. ShootingPhysics.cs & UnityEditor

Exercise 3

1. MyScorpionController.cs ➜ InitLegPos() & UpdateLegPos()
2. In Unity Scene
3. MyScorpionController.cs
4. IK_Scorpion.cs ➜ public void UpdateBodyPosition()
5. IK_Scorpion.cs ➜ public void UpdateBodyRotation()
6. Scorpion ➜ Player Controller.cs

Exercise 4

1. TailBodyTargetAniamtion.cs
2. Ik_Scorpion.cs ➜ Update()
   MyScorpionController.cs ➜ SetTargetNormal() & DistanceToTarget()
   ShootingPhysics.cs ➜ Update()

Exercise 5

1. MyOctopusController.cs ➜ update_cdd()
2. MyOctopusController.cs ➜ UpdateTentacles()
3. MyOctopusController.cs ➜ update_cdd()

Explanations

Exercise 1.5

As we need to use forces on the following exercises, we ended up using the Euler solver to compute the ball position each frame. We used the target - starting position vector as an initial velocity for the ball so it goes in that direction, and then add gravity force.

$$initialVelocity\ = target\ - initialBallPosition$$

$$\sum forces\ = \ gravity\ = \ (0, -\ 9.81, 0)$$

$$\begin{gathered} acceleration\ = \ forces\ /\ mass \\ velocity\ = currentVelocity\ +\ acceleration\ \cdot\ dt \\ position\ = currentPosition\ + velocity \cdot dt \end{gathered}$$

Exercise 2.5

To calculate the rotation velocity of the ball, we can assume various things that will make our lives way easier. First, we assume that it's a perfect sphere, so we can ignore things such as the inertia tensor.

In theory, $\omega\ =\ \tau/I$, but as we assume $I\ =\ 1$ then $\omega\ =\ \tau$

To calculate the torque τ:

$$\tau\ = point\ \times (direction \cdot force)$$

Where $point$ is the point on the ball where the force is applied.
$direction$ is the same direction as the initialVelocity vector (target - initialBallPosition)
$force$ is the force that the user selects in the UI

This is of course very approximated, but we don't care as we are in a videogame and not a faithful representation of real physics.

Exercise 2.6

The formula we ended up using for the magnus effect is the following:

$$magnusForce\ = \ S\ \cdot\ (\omega \times\ currentVelocity)$$

Where S is air resistance (we're using a hardcoded 2.0f value)
ω is angular velocity (as calculated in the previous exercise)

We chose this formula as it was the easiest to implement and it provided the visual effect that we wanted to achieve, without complicated calculations and cheap in performance. As we said previously, we don't need the most accurate physics as it's just for the visual effect in the game. At the same time, we apply a gravity force $gravity\ =\ (0, -\ 9.81, 0)$

Exercise 4.3

The rule that we've added is that the Sphere of the end effector of the scorpion must have it's forward axis aligned with the normal of the target point in the ball.

To do that, we take the dot product of the sphere forward vector and the normal of the target to compare it.

If the dot product returns 1, then it means that the tail is aligned with the target.
The problem is that the gradient descent minimizes the function, so if we added the dot result to the error function, what it would do is make the normal perpendicular to the forward vector.
To fix this, we add 1 to the dot result, and take the absolute value.