GAME 28567 - Programming for games 3: Motion

Assignment 2: Angry Birbs

*25% of final grade*

*Journal & Assignment: Due of week 10*

# Overview

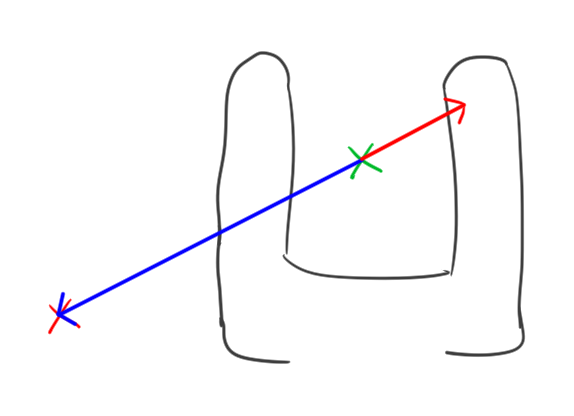
This project aims to create a simple physics-based game loosely using the same mechanics as Angry Birds. The game is played by loading a bird into a slingshot and firing at targets to break them. Some targets are static while others move vertically up and down. The more the slingshot is stretched, the harder a bird is thrown. When a bird hits a target with enough force, it breaks. Each bird has its own special ability that can be used when it is in the air as long as it has not collided with any other object yet.

Throughout this assignment, you will be expected to use the starter project and Unity’s 2D physics engine to implement the launching, special abilities, collision handling, and target movement. The scene and prefabs included do not need any further modification and should not be edited. All implementation should be done in code. **Do not use GameObject.Find or tags**. All queries should be done using the methods provided by the physics engine in Unity.

# Implementation Instructions

## Launcher

The code has already been implemented such that pressing the 1, 2, or 3 keys on the keyboard will load the red, blue and yellow bird into the slingshot, respectively. By clicking and holding near the bird, you can use the mouse to pull back the elastics at your desired angle. Releasing the mouse button will launch the bird by applying an impulse force to the rigidbody of the bird.



(Figure 1. The green cross marking the centre of the slingshot, the red cross marking the point position of the bird at mouse button release, the blue ***offset*** vector passed to the launch method and the red vector indicating the intended direction of the launch)

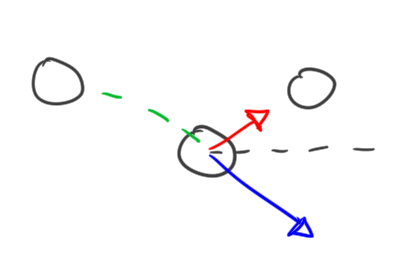
Your task will be to implement the Launch(Vector3 offset, float maximumStretch, Rigidbody birdRigidbody) method in the Birb.cs script. The ***offset*** variable is the vector from the stationary point on the slingshot to the point the mouse released the button (*see figure 1*). The ***rigidbody*** is the rigidbody of the currently loaded bird. It should use the ***LaunchForce*** variable to apply a variable force to the bird in the opposite direction. The launch impulse force should scale linearly with the amount of stretch of the offset. Use the argument **maximumStretch** argument to solve for the appropriate amount of force to apply.

## Birds

After launching a bird, clicking anywhere on the screen will make it use it’s special ability. A bird will only call its special ability once. Each ability is unique and will have its own implementation. The birds that have been launched and hit the ground can be ignored, there is no need to destroy them. Each script (noted in the header for each bird) will have its ExecuteAirSpecial method called when necessary.

#### Red Birb, AirSpecialBounce.cs

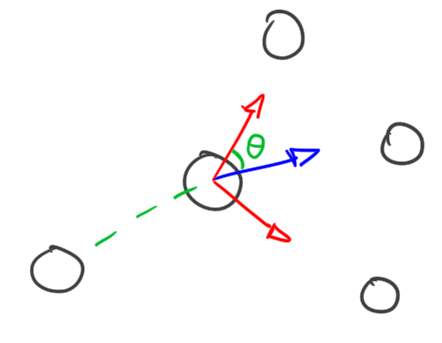
When the red bird is in the air, activating its special will cause it to flap and bounce. This should take the birds current velocity and reflect its direction vertically (*see figure 2*), and slow down it’s speed by the *percentage* stored in ***SlowDownFactor***. The provided public variable. ***SlowDownFactor*** will always be between 0 and 1.



(Figure 2. The green dotted curve indicating the motion of the bird, the blue vector indicating the instantaneous velocity of the bird on special activation, the dotted grey line indicating the reflection line and the red vector indicating the direction of velocity after the special)

#### Blue Birb, AirSpecialSplit.cs

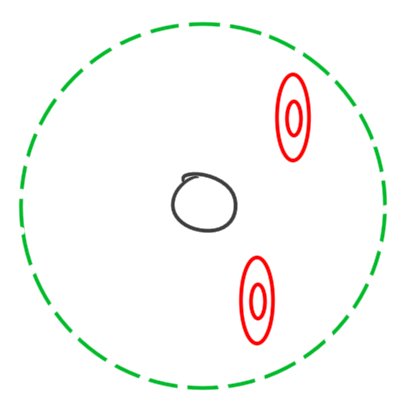
When the blue bird is in the air, activating its special will cause it to split into three birds (*see figure 3*). One bird continues on the same trajectory, one bird maintains the same speed but with a direction rotated by ***SplitAngleInDegrees*** degrees above the original heading, and one bird maintains the same speed but with a direction rotated by ***SplitAngleInDegrees*** degrees below the original heading. Please use the provided Birb.MakeBirbCopy( GameObject original ) to make copies of birds.



(Figure 3. The green dotted curve indicating the motion of the bird, the blue vector indicating instantaneous velocity of the bird at time of special activation, the red vectors indicating the two velocity vectors belonging to each of the newly spawned birds, and the green angle indicating the direction of the new birds relative to the original velocity direction)

#### Yellow Birb, AirSpecialExplode.cs

The yellow bird is in the air, activating its special will cause it to explode. All targets fully or partially within a radius of ***BlastRadius*** circle at the bird’s position at the moment of activation should be destroyed.



(Figure 4. the green dotted circle representing the circle of radius ***BlastRadius*** around the yellow bird during activation.)

Note: targets have been given their own layer, Targets. You can also assume that any object in the Targets layer will have a *Target.cs* script on the same gameobject. Use the ***Target.DestroyTarget*** method to destroy the targets when necessary (*see below for more Target details*).

## Targets

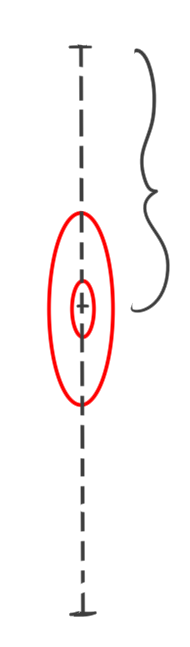
#### Destruction, Target.cs

When a target is hit, it may or may not break. The targets need to be hit with a minimum amount of speed for it to break. This speed is held in the **MinimumBreakSpeed** variable that has been provided.

To accomplish this, you should implement the ***OnCollisionEnter2D*** built-in method in Unity to detect the collision and determine whether or not the target should be destroyed. You should use the provided ***DestroyTarget*** method to destroy the target in the scene.

#### Moving Target, MovingTarget.cs

In the case of the further target, the target should move up and down with a consistent velocity. It should start by moving up by the distance stored in the ***HalfPathDistance*** variable at a speed based on the value stored in the ***MovementSpeed*** variable, in units per second. It should then move down at the same speed back to its original position, then continue traveling down until it has travelled another ***HalfPathDistance*** units. It should then move up, again at the same speed, to its original position and repeat this process continuously until destroyed (*see figure 5*). This should be done by using the rigidbody and not by editing the transform.



(Figure 5. the moving target and the path it moves vertically along. The indicated distance should match the length specified in ***HalfPathDistance***)

# 

# Submission

Please hand in your implementation as a unitypackage. Do not include any scenes or prefabs. Please submit the implementation by itself in the appropriate assignment folder.

**Do not compress or archive your solutions.**

**Do not edit the starter code outside of what is asked of you. This includes changing method signatures and changing accessibility of variables.**

Failure to follow submission guidelines will result in a deduction of **5% each** from the final grade of each part.