Ellen's Drone Solution

Introduction

This document describes the solution for the coding assignment: Ellen's Drone.

Score

The score system consists of two scripts: *Scorer* and *ScoreUI*.
The *Scorer* script is a singleton that stores the score for the player. Being a singleton it can be accessed globally without the need to have a reference to Ellen. This script is attached to Ellen (copy) and it notifies when the score changes to update the score HUD (see Figure 1). Note that it was added to a copy of Ellen in order to be inside the delivered folder[1].
The *ScoreUI*, attached to *ScoreCanvas* prefab (see Figure 2), takes care of initialising the score HUD by instantiating a *ScoreHUD* prefab that contains a simple *Text* inside a *RectTransform*. The *ScoreUI* gets notified when the score changes in order to update the HUD text[2].

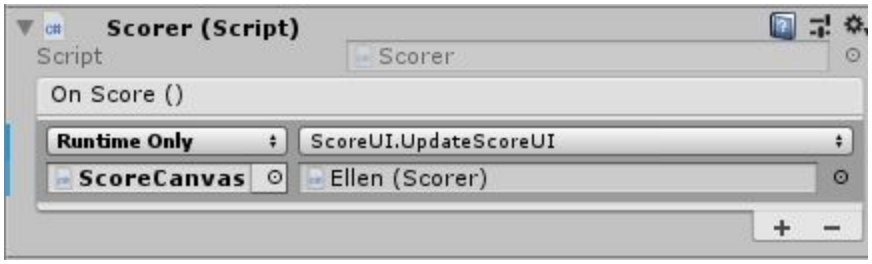

Figure 1. Scorer singleton on Ellen prefab.

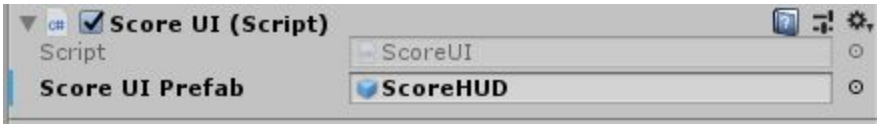

Figure 2. ScoreUI on ScoreCanvas prefab.

[1] The Scorer singleton could have been attached to an independent GameObject to keep Ellen unchanged. The decision of attaching it to Ellen was taken based on the idea that Ellen is the *scorer.*
[2] The ScoreUI could have been simpler. The decision of implementing it this was to keep the same architecture as HealthUI.

Pickups

The pickup system consists of two scripts: *Pickup* and *PickupsZone*.

The *Pickup* script, assigned to the *Pickup* prefab, represents the pickup item and it holds the *scoreAmount* to be assigned to the player (see Figure 3). This script handles the collision with *Ellen* and the *Drone,* adding the *scoreAmount* to the *Scorer* singleton when the collision happens. In order to collide with Ellen and the Drone, the *Pickup* prefab needs to be in the "*Collider*" layer[3] and it needs a collider configured as trigger. The *Pickup* script notifies the collision event to any observer.

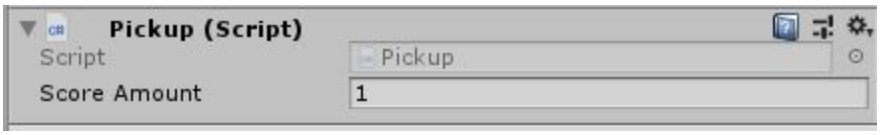

Figure 3. Pickup prefab configuration.

When the game starts, the *PickupsZone* script retrieves all the initialised pickups and then spawns more until there are *maxPickups* pickups in the scene[4], keeping them in a list. And when the player (or drone) collides with a pickup, the *PickupsZone* gets notified and it removes the pickup from the list. This script takes care of spawning the Pickups inside a *BoxCollider* and considering a minimum distance to other Pickups (minPickupSpace) and also a minimum distance to Ellen (minPickupToPlayerSpace).

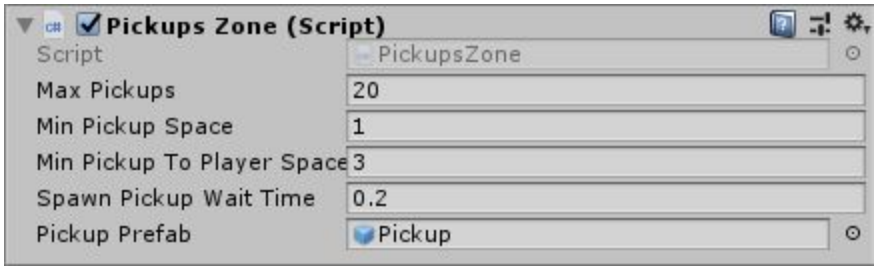

Figure 4. PickupsZone script attached to PickupsZone prefab.

[3] This is a constraint of the configuration of the project 3DGamekitLite (see Edit -> Project Settings -> Physics).

[4] The spawning occurs in a coroutine in order for the loading to be smooth (performance). The time between spawns can be configured in the PickupsZone prefab as *spawnPickupWaitTime*.

Ellen's Drone

The *DroneController* script contains all the logic that defines the Drone's behaviour based on five states: *Idle*, *Fetching*, *PickingUp*, *MovingUp* and *Returning*. The flow between these states is shown in Figure 5.

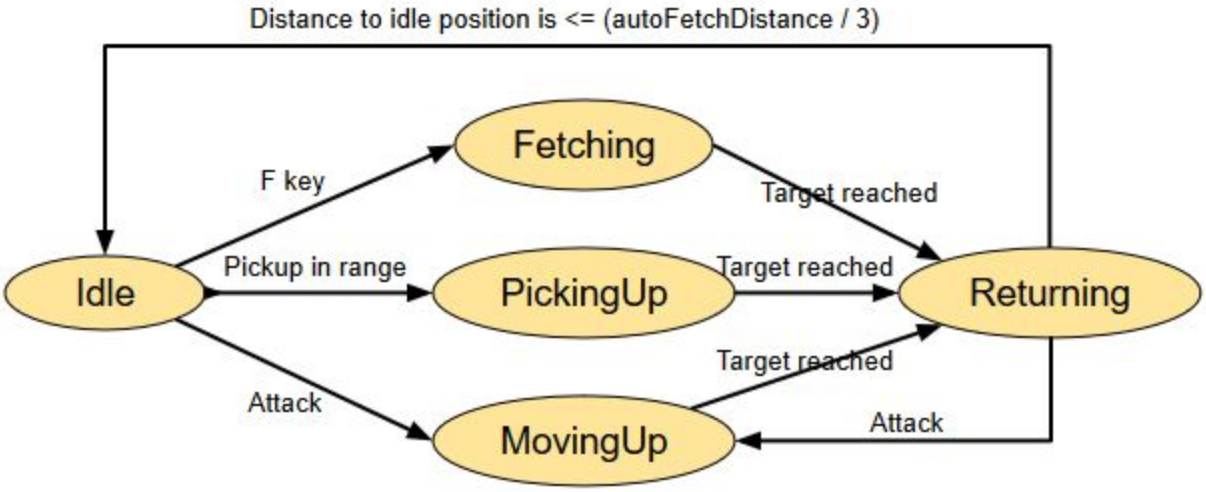

Figure 5. Drone's state machine.

The flow from one state to another is controlled based on the parameters shown in Figure 6.

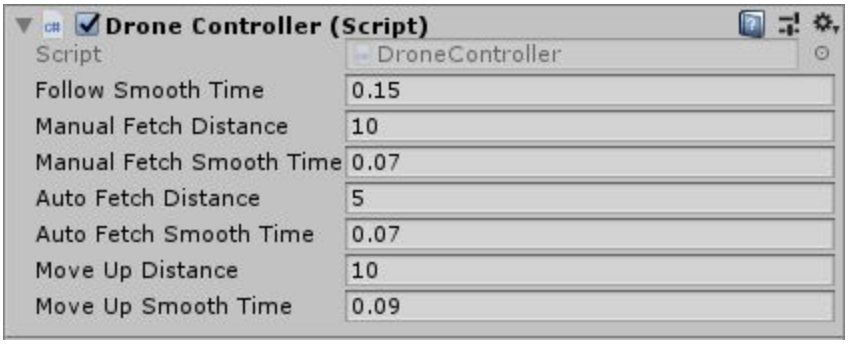

Figure 6. Drone's parameters to control state machine[5].

When the drone is in *Idle* state, it checks for the following events (in order):

1. F key press to fetch forward a distance of *manualFetchDistance*. The drone pass to *Fetching* state.
2. Ellen's melee attack to move up a distance of *moveUpDistance*. The drone pass to *MovingUp* state.
3. Pickup closer than *autoFetchDistance* to move to its position and pick it up. The drone pass to *PickingUp* state.

[5] All smooth time parameters are used for the speed of the drone: the lower the value, the greater the speed.

All previous states (*Fetching*, *PickingUp* and *MovingUp*) end when the drone reaches the target position, and then it passes to *Returning* state. While in *Returning* state, the drone listens for the following events:

1. Ellen's melee attack to move up a distance of *moveUpDistance* from the idle position. The drone pass to *MovingUp* state[6].
2. The distance to idle position is at least a third of *autoFetchDistance*[7]. The drone pass to *Idle* state.

The idle position is defined by a transform added as a child of Ellen's *HeadTarget* and it is referenced by the script *PlayerDroneTarget* attached to Ellen. The *DroneController* has a reference to the *PlayerDroneScript* to ask for the idle position every frame.

Figure 7. PlayerDoneTarget script that references the idle position.

The movement of the drone is made by using the utility function *Vector3.SmoothDamp*[8], executed every *FixedUpdate*[9] in order to be smooth.

Environment

The project was created on Windows10, using Unity3D 2019.1.9f1 and Visual Studio Community 2019.

[6] This is a design decision based on the fact that it looks better in gameplay, especially when the player spam the attack.

[7] Eventually, the drone never reaches the idle position, especially when Ellen runs all the time. This is why the threshold to pass to *Idle* cannot be too small (or the drone is always returning). The value of (*autoFetchDistance / 3*) works pretty well in practice and it comes from playing around with auto fetching feature.

[8] See https://docs.unity3d.com/ScriptReference/Vector3.SmoothDamp.html.

[9] Any other Update function causes the movement to be jittered.