Technical Report

Kick The Cone

Table of contents

Introduction

I started this game without a concrete project idea, at the beginning I just wanted to implement all that I have learned. At certain point I just wanted to release the game for the PC, but then I noticed that making it for Android would be a greater challenge and that it would make it more available for people to test. And that lead to many unused scripts, now documented in here and available for any other project.

Script list

Editor:

To use any of these scripts remember to put it in a folder in Assets named Editor. Or, add an assembly that controls which code comes first, always letting the editor scripts come first.

1. Coroutine extensions

In this script that I found online, one can control the coroutine more extensively being able to cancel and restart the coroutine by code. This is used in both GameController scripts. To use it remember to make a coroutine reference at the beginning of the code (private/public/protected/etc Coroutine <coroutine name>), and then when you want to restart it, do it as: this.RestartCoroutine(<IEnumerator name()>, ref <the coroutine reference>).

2. Folder finder

In this script there is some commented code that reflects the past use of it, both do almost the same thing, find a prefab in a selected folder. In the past code I did not need to use the Resources class, but I found out that this is needed if one wants to make a game for Android. Because the AssetDatabase does not work in this platform. The first function of this class finds the game obstacles in Assets/Resources/Prefabs/Obstacle Prefabs and place them in a list of GameObjects, returning this list to the class that called it. The second function does almost the same, as it searches for tracks in Assets/Resources/Prefabs/Track Prefabs and returns a list with the tracks. The third function does as the second function, but instead of finding and returning the prefabs as GameObjects, it returns as Module (commented in the fourth section of this editor list).

3. Invoke with parameters

Another script found online (here: https://gist.github.com/cbaltzer/6196131) that allows me to use an Invoke method and send parameters to the function called. I added this script trying to solve a problem, but later I found another way to do it, and this script is not being used. In any way this might be useful in future projects.

4. Module

Script that was made by my e-tutor Pedro Silva, it is used to control which direction each track is going. To use it, one must place it in the root Game Object of the track and then add the info needed, being: Direction (an enum, that one can choose), Start (Transform) and End (Transform).

Engine:

1. PC Scripts:

a. Ground Creator

This script uses the Folder Finder to receive the tracks and initiate them two times (object pooling). The common distance for rendering set on the public variable is 35, but if you want to set more than 40 or if you want to set the tracks before player variable higher than 5 the script will initiate the tracks three times. Any more need for instancing tracks will have to change the script so that it initiates the tracks four/five/... times. This is needed because the initiated tracks need to be deactivated to be considered “usable” tracks for the next interaction, this deactivation is part of this script. Only allowing deactivated tracks to be used in the track placement is an obvious solution to the tracks already instanced in front of the player changing position in the middle of the game.

This script uses the Module class by catching the direction of each track and not allowing the left or right direction to be instanced repeatedly. And it also uses the module to find the beginning and end of each track to place it.

In this script there is a function called PlayerRayCheck, in this function we return which track is the player standing on by using a raycast.

Another function is RenderedTracksGetLast, this one returns the last rendered track.

b. Game Controller

In this script the target frame rate is defined as well as the player forward speed is increased due the time passed.

Here the obstacle spawn chance is defined in a function called ObstacleSpawnChance, that returns the float representing the chance in % that the obstacles should spawn.

There is a function called Falling that checks if the player has fallen a certain level lower than it should be. To do this we know that my tracks do not go up or down, so if the player’s Rigidbody center of mass is lower than a certain value, he is certainly falling.

Additionally there is a function called WrongWay that checks if the player has turned itself in the direction of the track’s beginning.

Each game over situation is controlled by this script, that then sends the calculated situation to: the player script so that it plays an animation and sound (being the sound defined in this Game Controller script) in addition to instancing a water particle in case she falls on water, to the GameMenuManager so that it opens the game over menu with a mocking sentence based on how you lost.

ATTENTION! This script uses the coroutine extensions to set which call was the last within 2 seconds. This is needed because the player may hit an obstacle and fall out of the platform, and we do not want the menu to open twice, so we wait 2 seconds for all physics animations to be finished.

c. *Unity Chan for Infinite dodge*

This script is a highly modified version of the original script that comes with Unity Chan. First, the character does not rotate on pressing left (“a”) or right (“d”), only when pressed the button to specifically rotate: “q” to rotate left, “e” to rotate right.

Second, the physics that set the maximum magnitude to the character’s speed are on FixedUpdate, since we use AddForce. Besides, the jump and slide functions are placed here too since the jump function uses AddForce and I thought that would be not much of a problem to check the slide at the same speed we check for the jump.

Third, the LateUpdate function checks if the player is falling horizontally to play the animation related to it. This check is made the same way I made the horizontal falling in GameController, since we know that the player does not walk normally much lower or higher than a certain level, we define that if this condition is met, she is surely falling.

Fourth, the Slide function makes the collider smaller as the animation plays, and not on the transition state. Since I use the transition state to reset the collider.

Fifth, the Jump function sends an impulse to the Rigidbody and sets the collider higher, if the animation of jumping is being played, and not in transition. When the animation is in transition again, this function resets the collider.

There is a function called ResetCollider, that is the one called after the jump and the slide, that does what the name implies, resets the collider.

This script, as the original, sends animations to the animator, but some of this animations are received from the GameController script on TouchAnimation.

Additionally, this script is the one that receives the collision and sends the message with the collision settings to the GameController.

d. *Sunlight Control*

This script must be a component of the light used as sun. Here we rotate the sun creating a day/night cycle. And also, as I do not like the skybox used as default in the night sky, this script changes the skybox as the night falls.

Furthermore there is a function called LightSet, that returns if the street lights must be on or off.

IMPORTANT! In this script you must define the IsItOn variable as true if the game starts at day, and false if it starts at night.

e. *Mushroom Nav Mesh*

This script controls a mushroom trough a navmesh, this navmesh is made in runtime and it is constructed as the tracks are placed. The mushroom control is made in a way that it faces the last rendered track and moves towards that direction, playing an animation and sound as it goes.

If the mushroom collider hits an enemy or a cone it does play a specific animation and sound.

On top of that I have noticed sometimes the mushroom got stuck in a GameObject, so I did a check that it must be moving, if it gets stuck it will move automatically to the end of the track where it is standing.

2. Android Scripts:

Here we must pay attention because every force used must be much higher to have the same effect as in the PC. Even using FixedUpdate, the forces do not work the same.

a. Ground Creator

Works the same way as the PC version, but here I reference the Android Scripts instead of the PC ones.

b. Game Controller

Works almost the way as the PC version. In addition to what the PC version does, this script sets the timeout of the mobile screen as to never sleep. And it references Android Scripts.

c. Unity Chan for Infinite dodge

Works slightly different from the PC version, here we use the accelerometer with a low pass filter to move the character sideways.

This script has a AndroidTouchedInput function that checks the position of the touch and defines as the touch ends which direction it has moved the most. Subsequently setting the action the character must take: jump, slide, rotate left, rotate right.

d. Sunlight Control

Unfortunately it is too much for a phone hardware to handle real time directional lights moving. So this script in the Android version basically says that the lights are always off and that the sun does not move.

e. Tutorial Game Controller

This script is a Game Controller and Menu Manager for the Tutorial Scene. Basically a mix of both adding the fact that the player will be sent back to the last checkpoint in case it fails. These checkpoints are set by GameObjects spread around the scene. And by the time the player reaches the last checkpoint, the menu opens and the key “TutorialDone” will get a value.

This “TutorialDone” key is important so that the player does not play the tutorial over and over as the game initiates.

3. Menu Managers:

a. *Game Menu Manager PC*

This script controls every canvas text and activation in game, in addition to playing the in game music and adding the highscore key values.

There is a function called InGameSound that starts the game music as the scene loads and plays each song randomly at the end of another.

Another function is the Set Activity, that stops the music being played and enables the end game canvas menu. It does furthermore receive the score to save it if it is the higher archived. This function receives a number as a case from Game Controller, and sets the sentence to the game over based on what happened.

An additional function is the Score, this one controls the bottom part of the game helping the player know: how many points it got, if it did kick a cone or if it is going the wrong way. Using a coroutine to allow some time for the player to read what happened.

Some more functions: Play to load the game scene, BackToMenu to load the main menu scene, ThePause to allow the player to pause the game by pressing the pause on the top right, and an extra just for pc that allows the player to press p to pause and esc to go back to the main menu.

b. *Game Menu Manager Android*

Basically the same as its PC counterpart.

c. *Menu Manager PC*

This script controls the main menu scene. Here we have a control for the play and quit button, also the high score reset and view.

Only for the PC version there is a control to watch the mushroom run through the scenario.

d. *Menu Manager Android*

Same as the PC version but with a tutorial reset and without the mushroom.

e. *Set Volume*

Script I found online to control the volumes through a slider. This is a bit more complicated than it looks because the volume is controlled in decibels. This function controls not only the master volume but the Sound and FX volumes separately.

4. Cutscene:

a. *Cutscene Call Menu*

This script calls the main scene from a timeline asset placed in the same file.

b. *Unity Chan Control Scene*

This script controls the animations/physics Unity Chan must play while running on the cutscene.

c. Playable Control

This is a function that was supposed to restart the cutscene in case the player wanted to replay the tutorial. But that does not happen for reasons unknown.

5. Extras:

a. Tutorial Button Awake

The tutorial reset button on the Main menu must only appear if the player has already played the tutorial. This function does it.

b. Street Light Control

This script must be a component of every street light in the game. It checks if you are in the Android(day only) or PC(day/night), and sets the light accordingly, recieving the LightSet on each script.

c. Player Sounds

This script plays the player sounds accordingly to each action. These actions are messages sent by the animation. To set a message on a part of an animation check: https://www.youtube.com/watch?v=Bnm8mzxnwP8

ATTENTION! This script must be a component of the player and the player must have an Audio Source. Every sound here is public, so it can be set by the inspector.

d. Particle Starter

This script is made to start the particle system if a certain collider in the tree has been touched by the player. This particle system is just some leaves falling.

ATTENTION! This script must be a component of every tree you want this to work. Also the tree must have a trigger collider for this.

e. Obstacle Spawner

A code that was mostly made by my e-tutor. This places obstacles on top of each ground collider.

It is important to note that the chances of the obstacle to spawn are decided by each GameController, being PC or Android.

To place an obstacle, this code randomizes the obstacles on the list choosing one and placing in a random place on top of the ground. And when the the ground is disabled this code deletes the obstacle, so next time the Game Object ground is activated it randomizes another obstacle.

ATTENTION! This code must be placed on each Game Object Ground as a component, this Game Object must have a box collider. Also every obstacle has a layer to size them based on the local scale on the ground.

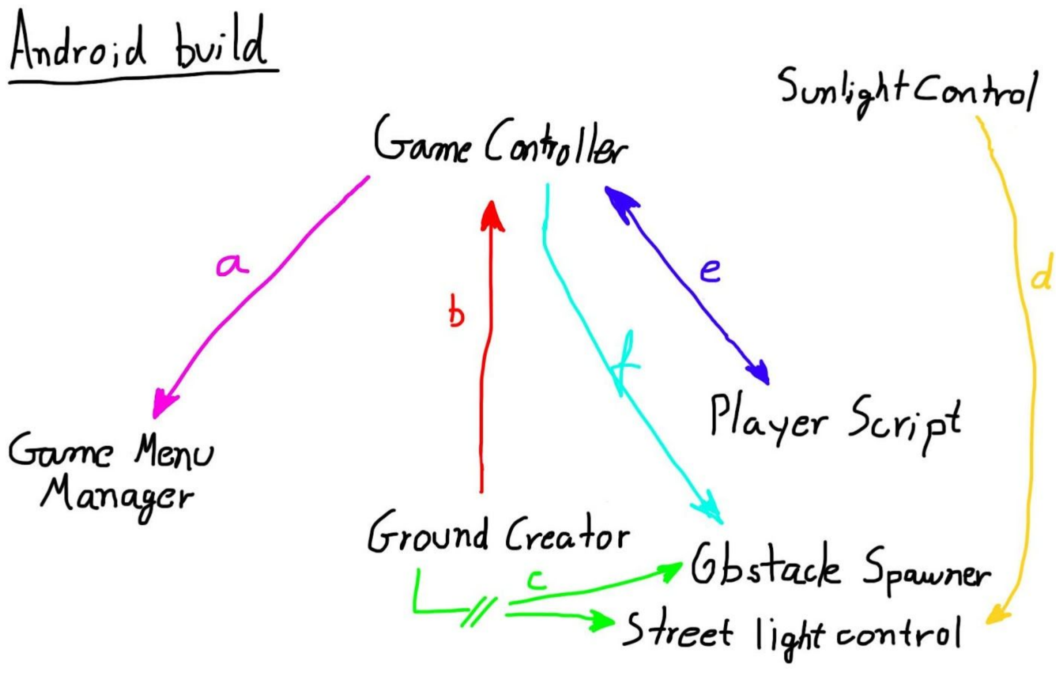
6. Unused

Many scripts were Unused on the final game, especially the ones that came with the assets.

Some of them were just used for testing, and some of them were used on the PC build that I gave up on doing because I just wanted to use some specific Shader for Android that does not look so good on pc.

7. Connection between scripts

As it is shown on the GDD, here it is the connection between the Android Scripts in game.



A) The Game Controller receives the collisions and sends to the Game Menu Manager so we can have context related game over sentences and also have the score on the screen's bottom.

B) The Game Controller receives the track where the player is standing on to check if the player is moving in the wrong way.

C) The Ground Creator starts two clones of each track and place them randomly in front of the player as he moves. At the time each ground of the track is activated the Obstacle Spawner instantiates an obstacle randomly, if deactivated it will delete the spawned obstacle and wait for another call. As for the Street light control, it will check each time his game object (light) gets activated if the sun is down. If it is, it will activate the lights, and the opposite too. Unfortunately, this last code is almost useless since a moving source of light would be very consuming on Android so I did not allow the light to move in the scene.

D) The Sunlight Control was a script made for moving the sunlight and making a day-night cycle. It also did change the skybox when it was night, so we would have a starry night. This script has a public function where Street Light control could receive the boolean representing if the light should be on.

E) Here the Game Controller receives the player’s center of mass, since the game does not have a walking upwards or downwards mechanic I can assume that if the player is under a certain level he is just falling out of the track. Also, the Game Controller receives the player’s public variable forward speed and manipulates it in a way that it increases as the time in-game does, both scripts have limits for the speed, being in the Game Controller the max 6 and in the Player Script the magnitude of the Rigidbody’s velocity will max at 8. Another connection between these two is that the Game Controller receives the player collisions and sends back: the respective animation that the player should do, disables the Player Script update in case of game over. It also checks if the player has hit the cone while sliding.

F) The Game Controller sends the probabilities of the obstacles to spawn to the Obstacle Spawner.