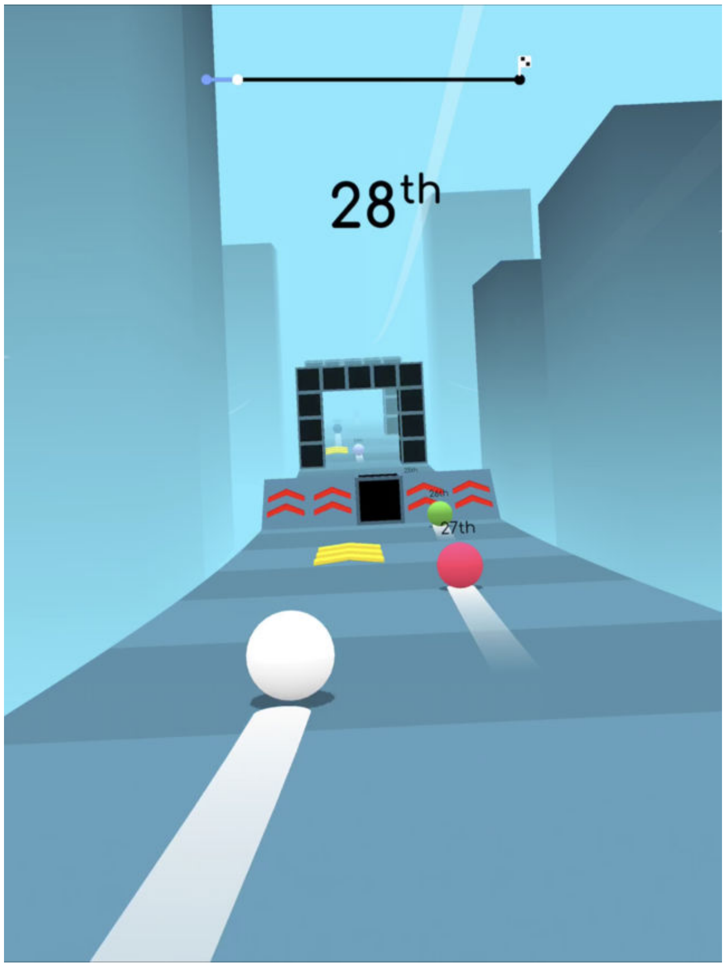
3D Graphics Project: Balls Race Game (2018)

3D Graphics Course - Technical University Of Crete

Panagiotis Drakopoulos - Teaching Assistant



1 Introduction

The task is to create a 3D Balls Racing game, based on the "Balls Race" game for Android and iOS devices (`https://www.youtube.com/watch?v=mC4jPyL2w3Y`).

The game should be developed in **Unity3D** Game Engine, by applying the techniques demonstrated in class. The player's **objective** is to constantly maneuver around obstacles, while trying to obtain a high score racing AI opponents. A fully completed game (100% of marks) should consist of the following sections and components:

1. **Graphics**: scene design, 3d objects, textures, materials, lights
2. **Physics**: colliders, physics materials, rigidbodies
3. **Scripting**: game logic and mechanics, AI opponents, controls, camera angles
4. **Audio**: sound effects
5. **UI**: On-screen UI for menus and score.

The more sophisticated the implementation of each of the above sections, the more marks your project will gain.

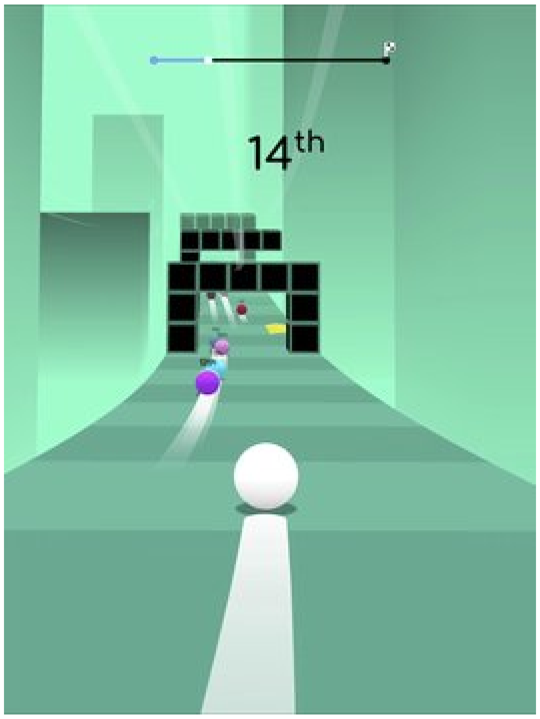
The percentages below indicate an approximate division of marks for each section of the practical work. The division of marks should also guide the development effort:

- Game design (Graphics, UI, Audio) : **35%**
- Game logic and functionality (Physics, Scripting): **50%**
- Personal touch and Extra functionality: **15%**

2 Minimum Requirements

1. **Level Design**: In terms of level design, you **must choose one** of the following options and develop your game based on it:
   - OPTION 1: One endless level with randomly generated obstacles, level parts, boost pick-ups.
   - OPTION 2: 2-3 pre-made levels with randomly generated obstacles and boost pick-ups.
2. **Graphics**: Scene(s) with the appropriate 3d objects (racing objects, obstacles, walls,..). All surfaces must have materials with at least one basic (albedo) texture attached to them.
3. **Physics**: Objects with appropriate components, such as colliders for collision detection. Some surfaces may require a physic material so that they have less friction.
4. **Game logic and mechanics**: As mentioned in the Introduction, the player's goal is to race other opponents while trying to avoid obstacles. Your player's object should be able to (at least) move forward (without user input) and steer left and right (by keyboard input). You can add more controls if you wish. (ex. jump). Finally, score updating is required.
5. **UI**: At least two UI GameObjects, such as *Text*, in the scene to display the current game score, position and state, by using at least one *Canvas*.The *Canvas* could be in Screen Space and/or World Space mode. The UI components need to be handled by script, so that they display the current status or score of the game.

3 Setting up the Scene



3.1 Basic Instructions

- Use your imagination and creativity to create an attractive game. Take a look at the real game for ideas.
- You can use Unity's **3D models** (cubes, spheres, ..) to create your scene by transforming them to give them he shape you wish.
- You can create your own 3d models in separate 3d-modeling software (such as 3D Studio Max, Blender, Sketchup) and import them in your Unity project. However, this is NOT recommended if you don't have plenty of time and haven't used such software before.
- You can use third-party 3d models found on the Internet to help you build your scene.
- Add appropriate **materials** to object surfaces. For example, use Unity's Material Editor to make a surface look shiny/smooth or rough

etc. `https://docs.unity3d.com/Manual/StandardShaderMaterialParameters.html`

- You can use extra **lights** to add style to your scene. Remember, you can customize the light color, position, intensity and more.

4 Physics

4.1 Collision between objects

Objects must have **Collider** components attached to them, otherwise they will pass through each other. It is important that the collider matches the 3D shape of an object as best as possible. For example, a box collider is ok for a box-shaped object, such as a wall or a smooth floor. More complicated 3D models require different types of colliders. For example, using a box or a sphere collider for a pin is wrong, as it will result in un-realistic behaviour.

`https://docs.unity3d.com/Manual/CollidersOverview.html`

4.2 Rigidbody

Rigidbodies enable your GameObjects to act under the control of physics. The **Rigidbody** can receive forces and torque to make your objects move in a realistic way. Any GameObject must contain a Rigidbody to be influenced by gravity, act under added forces via scripting, or interact with other objects through the NVIDIA PhysX physics engine.

A rigidBody is defined by many parameters, such as its mass, drag, angular drag and more.

`https://docs.unity3d.com/Manual/class-Rigidbody.html`

4.3 Raycast

Raycasting is a Physics function, often used to check for the presence of other objects (with colliders) at a specified direction. This works by casting an invisible ray, from a point of origin, towards a specific direction against all colliders in the scene. The function returns true if the ray hits any collider or false if it did not hit anything.

You can customize Raycast parameters such as: direction, point of origin, length, layer masks (used to ignore a specified layer of colliders).

`https://docs.unity3d.com/ScriptReference/Physics.Raycast.html`

4.4 Physic material

The Physic Material is used to adjust **friction** and bouncing effects of colliding objects.

To create a Physic Material select *Assets, Create , Physic Material* from the menu bar. Then drag the Physic Material from the Project View onto a Collider in the scene.

`https://docs.unity3d.com/Manual/class-PhysicMaterial.html`

5 Scripting: Game logic and functionality

The scripting part involves the entire functionality of your game. That is from handling user input, triggering events upon specific actions and more.

When the game starts, the user will be presented with a menu that gives him the option to Play or Exit the game. If the player clicks play, the level will start. The player's object will have to constantly move forward, without any input. The player should not be able to stop the forward movement or slow down. If a player collides with an obstacle, a 'Game Over' UI should appear, informing the player of his high score. The 'Game Over' screen should also give the player the option to Restart the level.

Your scripts will have to:

- **Initialize** variables, load game objects and/or other scripts that need to be accessed.
- Apply movement on player's and opponents **RigidBodies** (by using forces or position transformations).
- Produce/generate game content "randomly", such as obstacles, level parts, boost pick-ups.
- **Keep track of the game state** and act accordingly. For example, you should constantly check for:

  - keyboard input
  - collisions
  - player's current position

- **Reset game objects' state** (such as position) **when required**.
- Keep the **game score** and thus update the UI elements.
- Implement a very simple AI system for the opponents, so they can move and avoid obstacles. (hint: Raycast)

Important things to remember:

- A script will never run, unless it is attached to at least one game object in Unity.
- A single script can be attached to one or more game objects. In this case, **be careful**, as the same script will run as many times as the number of game objects it has been attached to.
- Game objects can have multiple scripts attached to them, too. In this case, **be careful** of possible race conditions. (a race condition can occur when, for example, 2 different scripts try to move the same object at the same time).
- Game objects (or other scripts) need to be loaded in the Start() function of a script, before they can be used. However, Game Objects that are attached to the script, can be used directly and don't need to be loaded.

The links contain useful tutorials on the communication between scripts and Game Objects in Unity:
`https://unity3d.com/learn/tutorials/topics/scripting/how-communicate-between-scr`
`https://unity3d.com/learn/tutorials/topics/scripting/getcomponent`

6 UI , Audio and personal touch

6.1 User Interface (UI)

Unity's UI system provides a variety of tools to render elements, such as text or images, on the screen or at a specific position in the 3D space. The

UI system can be used for many different purposes.

In this project, you will have to utilize the UI system to display a menu and important game information, such as the current score/position. The most important UI components are the **Canvas** and the **Text** components.

`https://unity3d.com/learn/tutorials/topics/user-interface-ui`

It is totally up to you to design the UI system, as long as it meets the minimum requirements.

When creating the UI, you will have to decide whether the Canvas will be in **World Space** or in **Screen Space** mode, or **both**.

If a Canvas is in World Space mode, the UI elements it contains will be rendered at a specific location in the 3D scene, depending on the Canvas transform coordinates.

If a Canvas is in Screen Space mode, the UI elements will always be in front of the screen, no matter where the camera is or looks (like a HUD).

6.2 Audio

You can use Unity's audio components to play sounds in a loop(ex. background music) or after specific events (ex. collisions).

`https://docs.unity3d.com/Manual/AudioOverview.html`

6.3 Extended/Extra functionality and personal touch

This section is intended to give you the freedom to add your **personal style** to your game, extend or add extra functionality that is not listed in this document.

Extra functionality may include..:

- A variety of 3D level objects, such as obstacles, level parts and more.
- Visual effects
- Sound effects
- Extra control mechanisms

7 Project Guidelines

- Your code must be organized in different scripts, depending on the functionality. For example, use a script that handles the UI elemenets only (ex. menu, score display, ..), another script that handles the game logic (ex. throwing the ball), another script that handles different cameras positions and angles (if you include any) and so on.
- There is no correct amount of scripts you should use, just make sure each script has a distinct role.
- Your code must be written in C# ONLY.
- Organize your game objects in a hierarchy that makes sense and is easy to understand. For example, if you have game objects such as "Cube(1)", "Cube(2)", "Cube(3)", ... , make them children of a single "CUBES" game object.
- Organize your asset files (textures, 3d models, materials, scripts, sounds, ...) in separate folders. Obviously, the name of the folder should be representative of the content.
- You may use the Internet (YouTube, blogs, ..) to your help for tutorials and code examples, but copy-pasting entire code or functionality is obvious and will NOT be accepted.

Good Luck !