TÉCNICO LISBOA

3D Programming

Assignment #3

Unity Game

Group 7:	*Student Number:*
Luís Fonseca	ist79770
Ricardo Fonseca	ist90862

Contents

1 Game Overview

The goal of this assignment was to create a 3D game using the Unity game engine while focusing on global illumination. The game was developed using Unity 2017.3.1.

Our game mechanics and aesthetics take inspiration from the Tron movies. The player holds a disc similar to the identity disc in Tron which can be thrown to destroy obstacles and enemies thus earning score points. The player loses if it's health reaches 0 or it falls out of the map. To win, the player must reach the white highlighted area.

2 Implementation

2.1 3D Scene

The game scene is composed mainly by regular geometry to fit the overall game theme. There are two spaces in the level, indoors and outdoors, which have different textures to differentiate between the two. The light emitted from the floor and a directional light illuminate the outdoor space. On the other hand, the indoor scene is lit by several different colored point lights.

To navigate the scene, there are autonomous yellow platforms, moving up and down, that allows the player to reach higher places since it cannot jump.

2.2 Game mechanics

The game uses the first person shooter perspective with the main camera defined inside the player game object.

2.2.1 Player

The player is defined by a capsule collider and a rigidbody which allows the the character to hit other objects and be affected by gravity.

The character control is defined in three scripts: *PlayerController*, *PlayerMotor* and *CameraAim*. In order to move the player, pick objects or shoot the disc, the *Update* function in *PlayerController* checks the user input. If the user moves the mouse or moves the character, we calculate the player transform and the camera rotation and move each game object accordingly in *PlayerMotor*. The first script also allows us to control the inventory by picking objects ("P" button) or selecting items from the inventory ("1,2,3" buttons). The object pickup action utilizes rays casted from the camera. If the ray intersects an interactable object, then depending on the distance, it focus that object and the player is able to pick it up. The mouse button 1 throws the disc if it's in the players possession or retrieves it to the hand otherwise.

2.2.2 Disc

The disc serves as a weapon just like any other first person shooter. We define the disc behaviour in the *DiscController* script. Since the disc is defined by a cylinder collider and a rigidbody, we define two states: when its on the players hand it can't interact with the player's collider and when is thrown it bounces of the scene objects with colliders. The script allows us to set the hit damage, the speed, the player's catch radius and the maximum throw distance of the disc. When the disc is thrown, the collider listens to a trigger event in order for the player to catch the disc when it is in its radius.

Different types of discs were made into prefabs where different behaviour was possible by labeling them with different tags.

2.2.3 Inventory

Throughout the scene there are a few discs that were defined with *Interactble* behaviour. Whenever the player is close enough and looking at the objects, they are highlighted which means they can be picked up. The player starts with a normal disc in the inventory and is able to pickup the ice and fire discs during gameplay. The ice and fire discs differ from the normal not only on the textures but also on behaviour. The ice disc inflicts less damage but freezes the enemies motion temporarily. As for the fire, it deals damage over time.

2.2.4 Enemies and Obstacles

Enemies are turrets that fire bullets towards the main character. An enemy is defined by two colliders. One collider is used to trigger the turret target. *OnTriggerEnter* sets the player as the target and starts shooting bullets at him and *OnTriggerExit* sets the turret to an idle state. The other collider checks if it's hit by a disc object and decreases the enemy health according to the disc type. At certain stages, the enemy's health triggers different particle systems to indicate its life status.

We also define moving walls with rigidbodies as obstacles where each one is contained inside a collider. The movement is defined by trigger events: when the wall is inside the collider, we add a force to the rigid body. On the other hand, when the wall exits the collider, we change the direction of the force. Doing so keeps the wall moving back and forth perpetually. The walls rigibodies are frozen on the y and z axis to obtain movement only in x direction.

2.2.5 Score System and Level Completion

The current score is defined by a small script which is updated when the player destroys an enemy. To end the game, the player dies (either by falling out of the map or losing all the health) or reaches the game completion zone. This prompts the end screen menu with a different message according on how the game ended. The current score, players name and game status are kept in the *PlayerPrefs* so the information isn't lost when there is a scene change. To keep records from all sessions, the current session information is saved in a persistent .json file.

2.3 UI

2.3.1 Heads-Up display

The heads-up display is divided in: current score on the left , the crosshair in the center and the health bar, mini-map and inventory on the right. All the HUD elements were created using Unity's UI elements. The health bar is composed by two images and text that shows the player health percentage. Whenever the player takes damage, we recalculate the current health and maximum health ratio and scale the bar fill on the x axis. To render the mini-map, we use an orthographic camera that follows the player in the x and z coordinates. The camera saves an image of every frame as output which is used as source for the screen UI element. The inventory bar is composed by different slots which one containing one type of disc. When the player picks a new disc, the first empty slot is filled with a image according to the type of disc.

2.3.2 Main Menu

Our main menu was created on a new scene using Unity's UI. The main screen is composed by buttons that lead to different sub-menus, e.g the Help Menu. The buttons behaviour were implemented using the *OnClick* function where we hide the main screen menu and enable the game object associated with the sub-menu. Using the same functionality, we implement the return buttons in the sub-menus.

Pressing play directs us to a screen with a text input field used to define the player's name. When finished, the player presses enter to load the level and start the game.

2.3.3 End Menu

Much like the Main Menu, the end menu was created on a different scene. It contains a button that allows the user to return to the main menu and a table with the best scores. On top of the screen, we have a message that changes according to how the game ended ("*Game Over!*" or "*Level Complete*!"). The current score is displayed below that message and it's only recorded if the player is able to complete the level.

2.4 Visual Effects

All the visual effects were obtained using built-in shaders from Unity and using the Post Processing stack.

2.4.1 Particle Systems

Two types of particle systems were used in the game: smoke and explosions. The smoke (fig.3a) changes to a darker color over lifetime according to the enemy's damage. The explosion (fig.3b) is triggered by the enemy's death. Both particle systems were obtained from the unity asset store and use textures with normal maps with different sizes over the lifetime. The explosion uses a point light for a more vivid effect.

2.4.2 Bump Mapping

We implemented normal mapping on the stone walls and floor of the indoor section of the scene. In fig.4, we see a block of stone illuminated by the emissive floor and the sun (direct light) where the normal maps makes it look like an irregular surface.

2.4.3 Lightmap and Ambient Occlusion

In order to allow light mapping, we set all the game objects except the enemies to static objects. In Unity lighting settings, we enable the baked global illumination option and set to baked indirect mode. We also set ambient occlusion in Unity lighting settings.

2.4.4 Reflection Probes

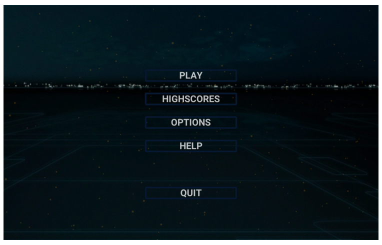
We use reflection probes to generate cubemaps that are used by the scene walls with reflective materials. As we can see in fig.7, the cube material reflects the skybox.

2.4.5 HDR Bloom

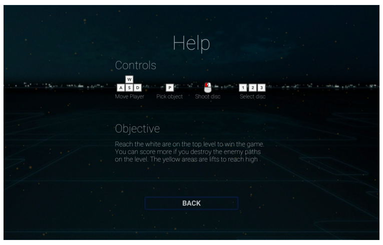
Using a free asset from the asset store, we enable bloom in order to obtain the emission effect from the disk and floor materials.

A Images

A.1 User Interface



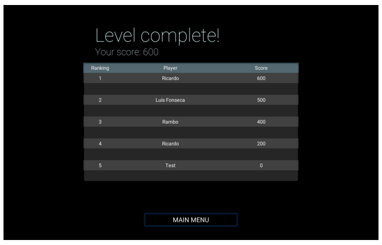
(a) Main menu.



Objective

Reach the white are on the top level to win the game. You can score more if you destroy the enemy paths on the level. The yellow areas are lifts to reach high

(b) Sub menu.



Level complete!

Your score: 600

Ranking	Player	Score
1	Ricardo	600
2	Luís Fonseca	500
3	Rambo	400
4	Ricardo	200
5	Test	0

(c) End menu.

Figure 1: Anti-aliasing sampling comparison.

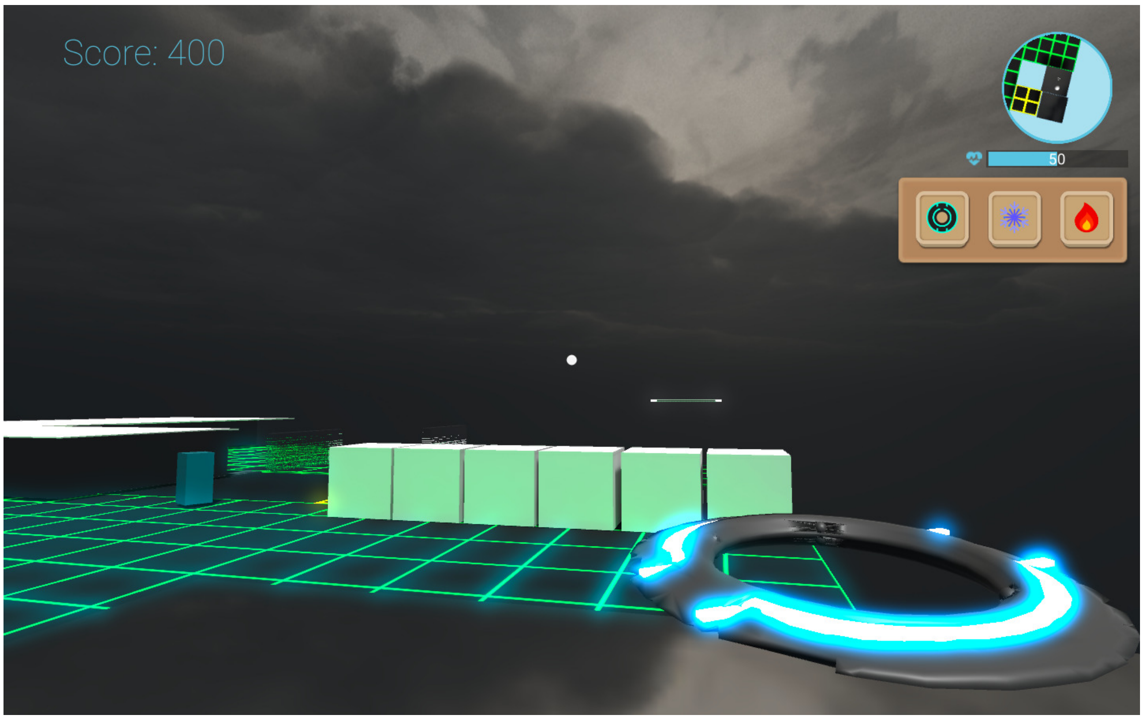
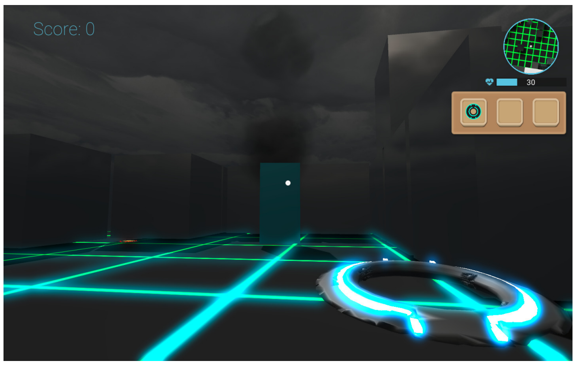
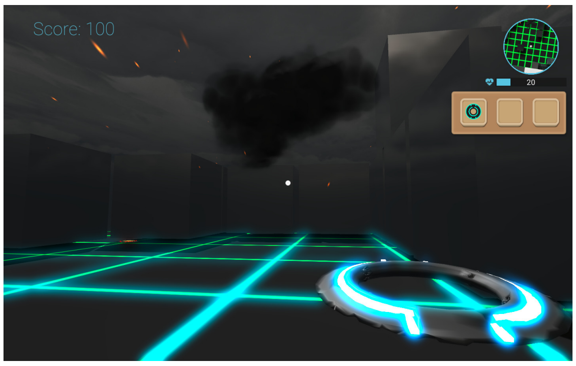


Figure 2: Heads-Up display containing the current score, a mini map, health bar and the inventory.

A.2 Visual Effects



(a) Smoke.



(b) Explosion.

Figure 3: Particle Systems.

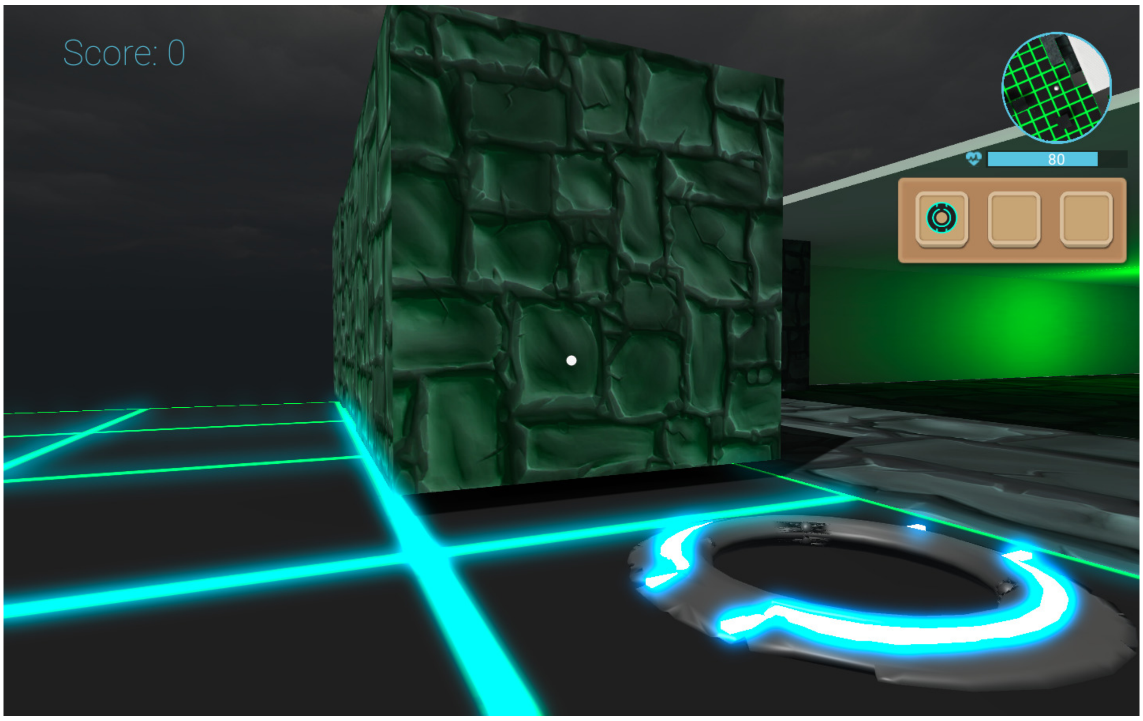


Figure 4: Stone texture using a normal map.It is also visible shadows obtained from a shadowmap.

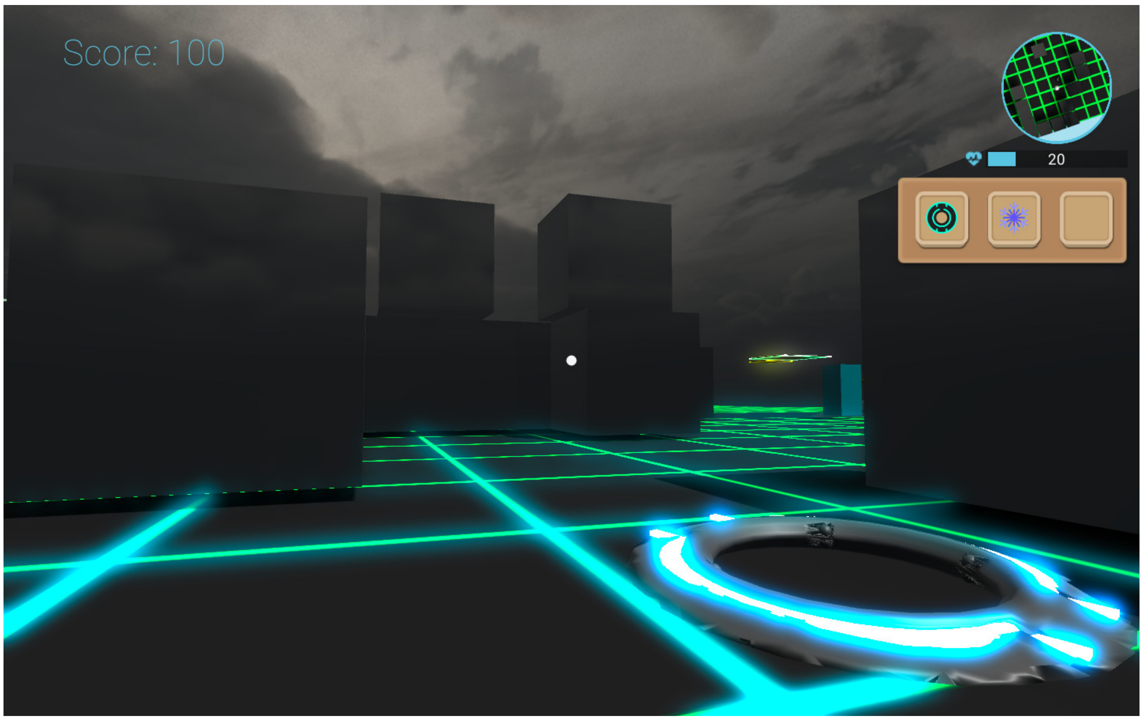


Figure 5: HDR Bloom effect in the disc and floor.

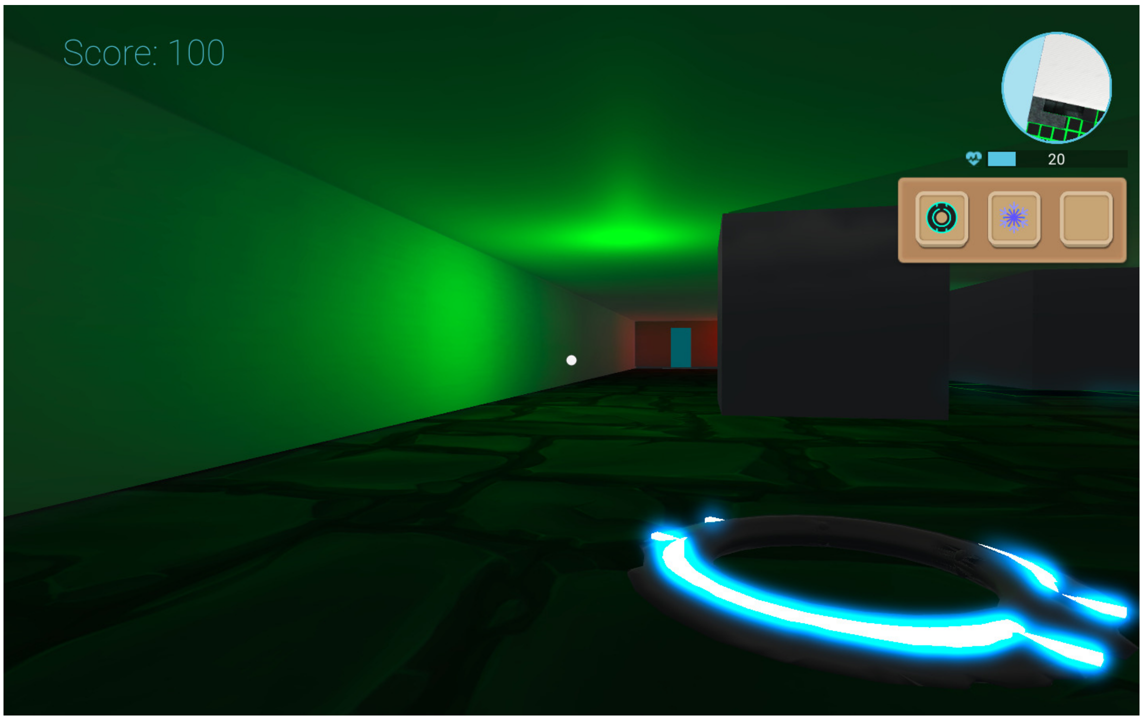


Figure 6: Lightmaps including ambient occlusion.

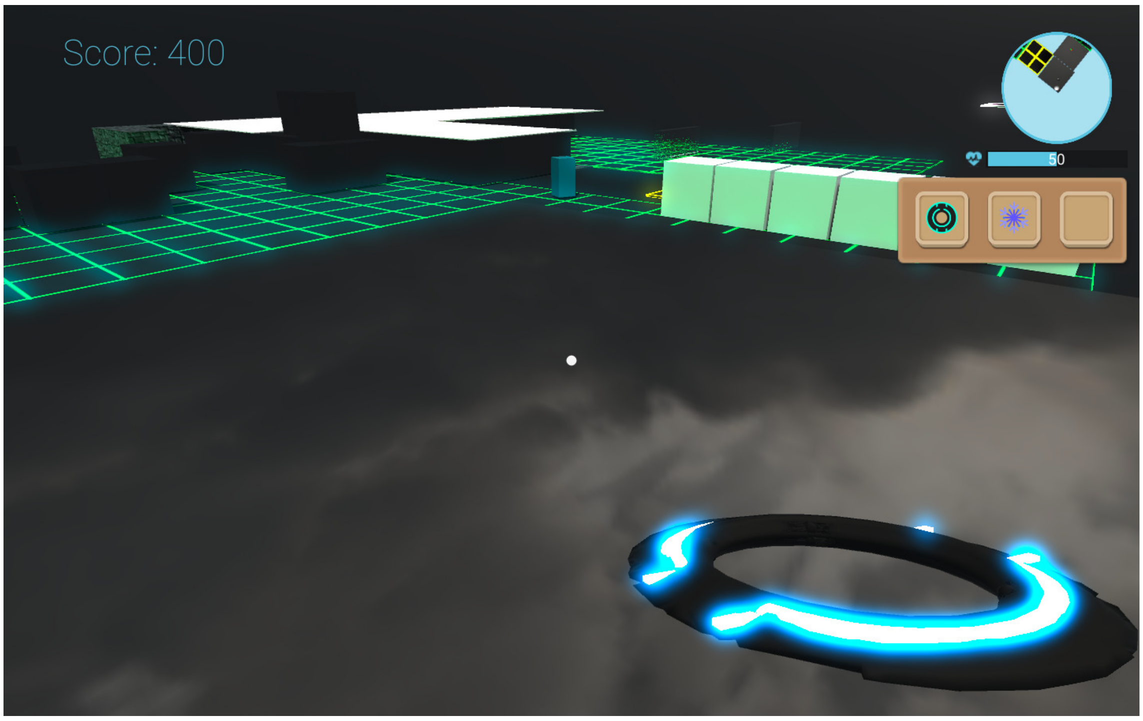


Figure 7: Reflections probes.