Strike&Shine

"Merging Digital Entertainment with Real-World Interactivity."

Technology Summer Internship -2023

Mariana Carvalho
Miguel Rocha
Fátima Napoleão

accenture

Contents

ABSTRACT

This report describes the development of a Unity-based Bowling Game, for a VR immersive experience whit real world interactivity.

Each time the player knocks down all the pins (strike), the score increases by one unit and an external lamp is activated via HTTP communication.

The report covers game development, integration, challenges faced and the connection between the virtual and real worlds.

INTRODUCTION

The initial plan was to build a Unity game which would allow the user to interact with a physical coffee machine, using VR. Yet, this idea would require a high level of expertise in hardware manipulation, since the coffee machine would have to be cut opened and altered to allow this to be possible.

The final project came out to consist of building a Unity based game that when certain thing of our choice happens, an external lamp is activated.
The primary goal of this project is to explore the dynamic interplay between Unity game development, HTTP communication, and smart device integration.

Our game serves as an engaging and entertaining platform for users to enjoy a virtual bowling experience, while also creating a tangible connection to the gameplay through a physical lamp that reacts to a strike.

It allows the user to interact with a ball, by grabbing it, and then, throwing it against the target pins. The player can also select different types of balls, each one with specific mass and color, as in a real bowling game.

LAMP

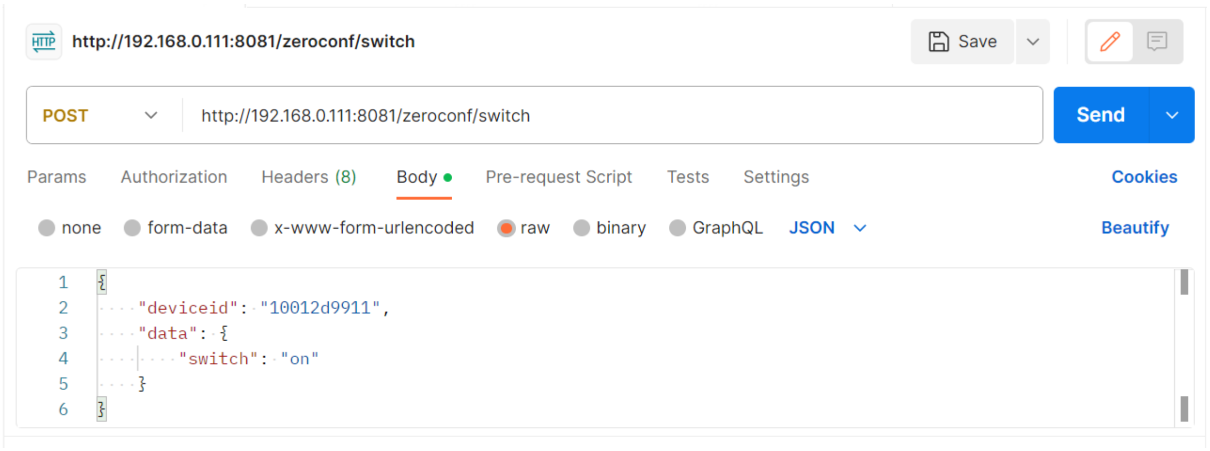
The lamp works through a Sonoff smart switch that in our case, was already implemented.
The goal is to turn on the lamp through an http signal sent to the switch.
The setup is explained step by step in the “Setup” document.

After connecting the switch to the Wi-Fi using the computer, we followed the following steps to test sending the http request:

1. Install Postman: Download Postman | Get Started for Free
2. Find the IP address of the switch using a net analyzer app.

First we used Postman to test the IP address and the http request sending. For help we watched the following link : https://youtu.be/LcqV4a8yN0E

To turn the light on:

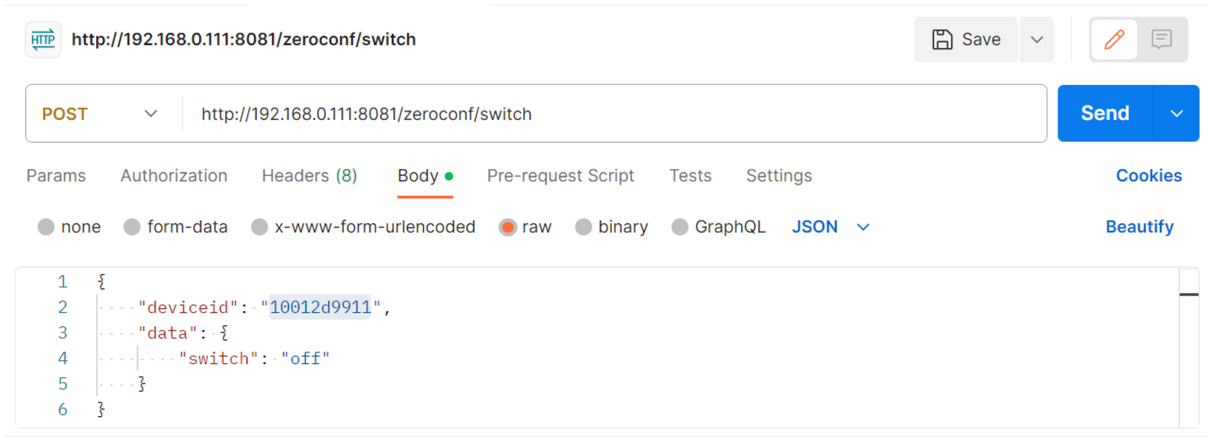

http://192.168.0.111:8081/zeroconf/switch

POST http://192.168.0.111:8081/zeroconf/switch

Body: raw, JSON

```
{
    "deviceid": "10012d9911",
    "data": {
        "switch": "on"
    }
}
```

To turn the light off:


http://192.168.0.111:8081/zeroconf/switch

POST http://192.168.0.111:8081/zeroconf/switch

Body: raw, JSON

```
{
    "deviceid": "10012d9911",
    "data": {
        "switch": "off"
    }
}
```

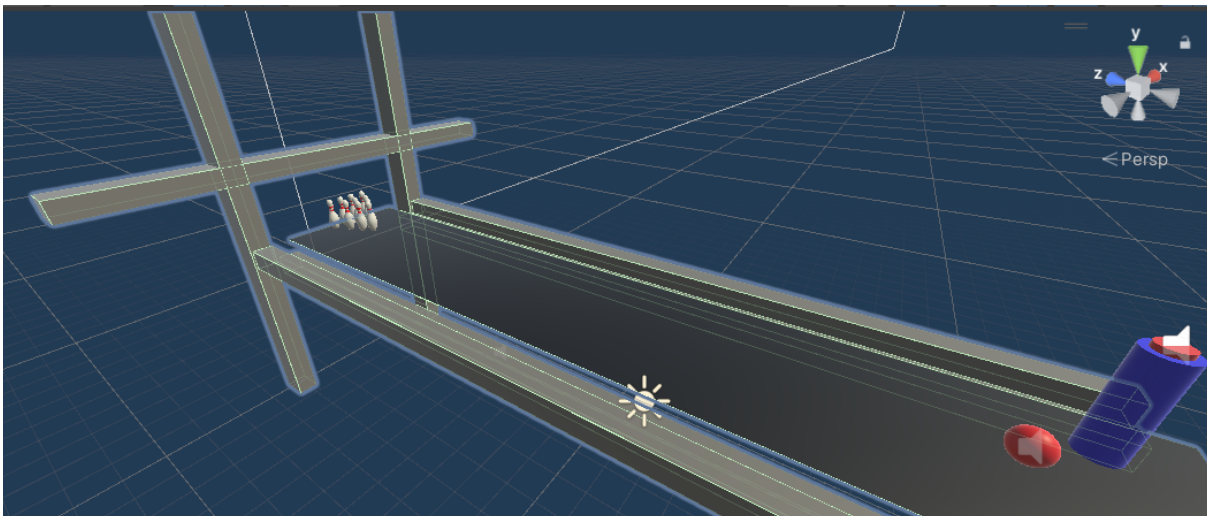
UNITY IMPLEMENTATION

In this chapter, all implementation details for creating the bowling game or gameplay mechanics will be explained.

BOWLING ALLEY

The alley is supposed to mimic the real bowling alley and represents the area where the action will be developed. It consists of:

- The plane, that represents the bowling alley where the ball will roll towards the pins.



As you can see in the image above, the plane has a Box Collider element, since it should provide a base to the other game objects. Also, note that it doesn't have a rigid body element, otherwise, physics would be applicable to the plane, and it would fall into infinity.

- The boarders, which limit the size of the alley, preventing the ball from falling into the limbo. In the left and right side, there are two depressions that drive the ball to the end without hitting any pin.

- Invisible corners (Corner_0 to Corner_3) to define the playable area. This is used to define the limits in which the player can move.

BALL/SPHERE

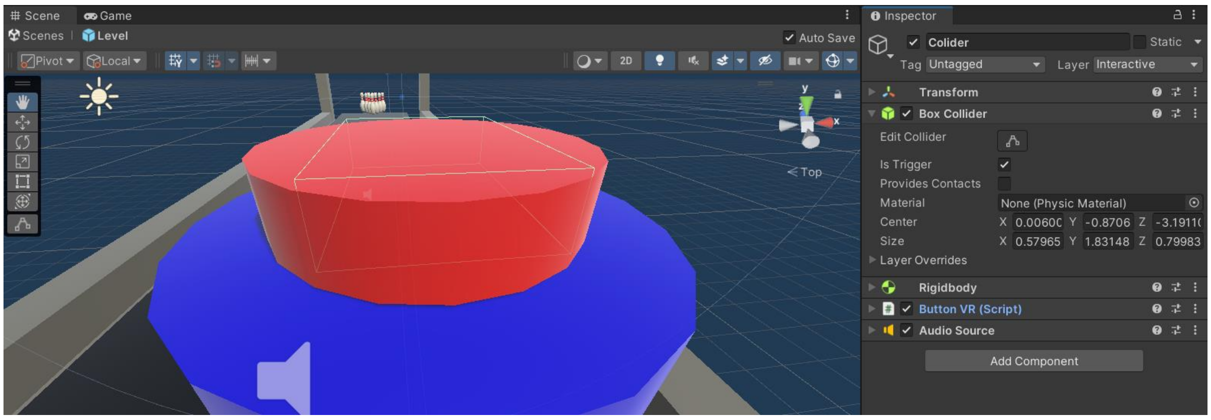
To represent the ball, we’ve created a sphere 3D object. This game object has:

- A script to control it respawn functionality. This script observes if the ball fell down the alley, or the current ball velocity is zero, and, in that case, it changes its position to the initial ball position, and adjusts its velocity to zero.
- Rigid body to be subjected to physics, which is important since the user will throw the ball into the air, and the ball must fall. This will also take care of velocity changes, when the ball hits the target pins.
- Sphere collider to be able to interact with other objects available in the game (ex: alley borders, pins, alley ground, etc.).
- Sound manager script, that plays an audio when the sphere falls into the alley ground.

BALL SELECTOR

Like any real bowling game, the player should be able to switch through different balls, each one with different colors and weights. To implement a ball selector, a 3D interactable button was built. When the user presses this button, the ball is switched to another one, following a pre-defined cycle to iterate through all the ball types. This approach is designed to avoid the necessity to have more buttons to select an individual kind of ball.

Duo to technical difficulties, this only works the use grabs the ball first, and then presses the button with it.



Colider ✓ | Static
Tag Untagged | Layer Interactive

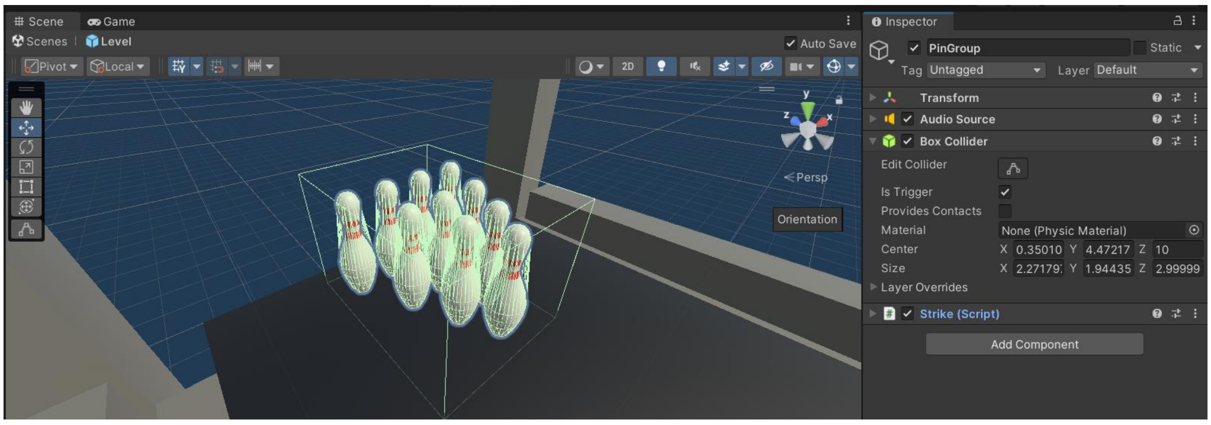
Transform
✓ Box Collider
Edit Collider
Is Trigger ✓
Provides Contacts
Material None (Physic Material)
Center X 0.0060C Y -0.8706 Z -3.1911(
Size X 0.57965 Y 1.83148 Z 0.79983
Layer Overrides
Rigidbody
✓ Button VR (Script)
✓ Audio Source
Add Component

PINS

The pin group prefab is composed of 10 similar pins. This component plays an audio sound of pins being hit, when the ball collides with it.

In practice, it has a Box collider, which detects when the ball enters in the game space. Logically, the first thought would be to place a box collider for each pin, but since the audio we have used was recorded when multiple pins were being hit, if we went with this approach, a lot of confusing noise would be heard, which would spoil the user experience. Another important component is the XR Grab Interactable Unity built-in component, which deals with all the boilerplate code necessary to the sphere to be grabbed by the VR controllers.

To design the pins, we used blender and followed the instructions from the following video: https://www.youtube.com/watch?v=6cv7r6UD9xM



PinGroup
Static
Tag Untagged
Layer Default
Transform
✓ Audio Source
✓ Box Collider
Edit Collider
Is Trigger ✓
Provides Contacts
Material None (Physic Material)
Center X 0.35010 Y 4.47217 Z 10
Size X 2.27179: Y 1.94435 Z 2.99999
Layer Overrides
✓ Strike (Script)

VR CAMERA AND HAND CONTROLS

During the game implementation, we tried to implement a vr camera from scratch, and apply some Unity built-in components, so that the camera and controls could respond to the actions made by the player when he/she is using the VR equipment.

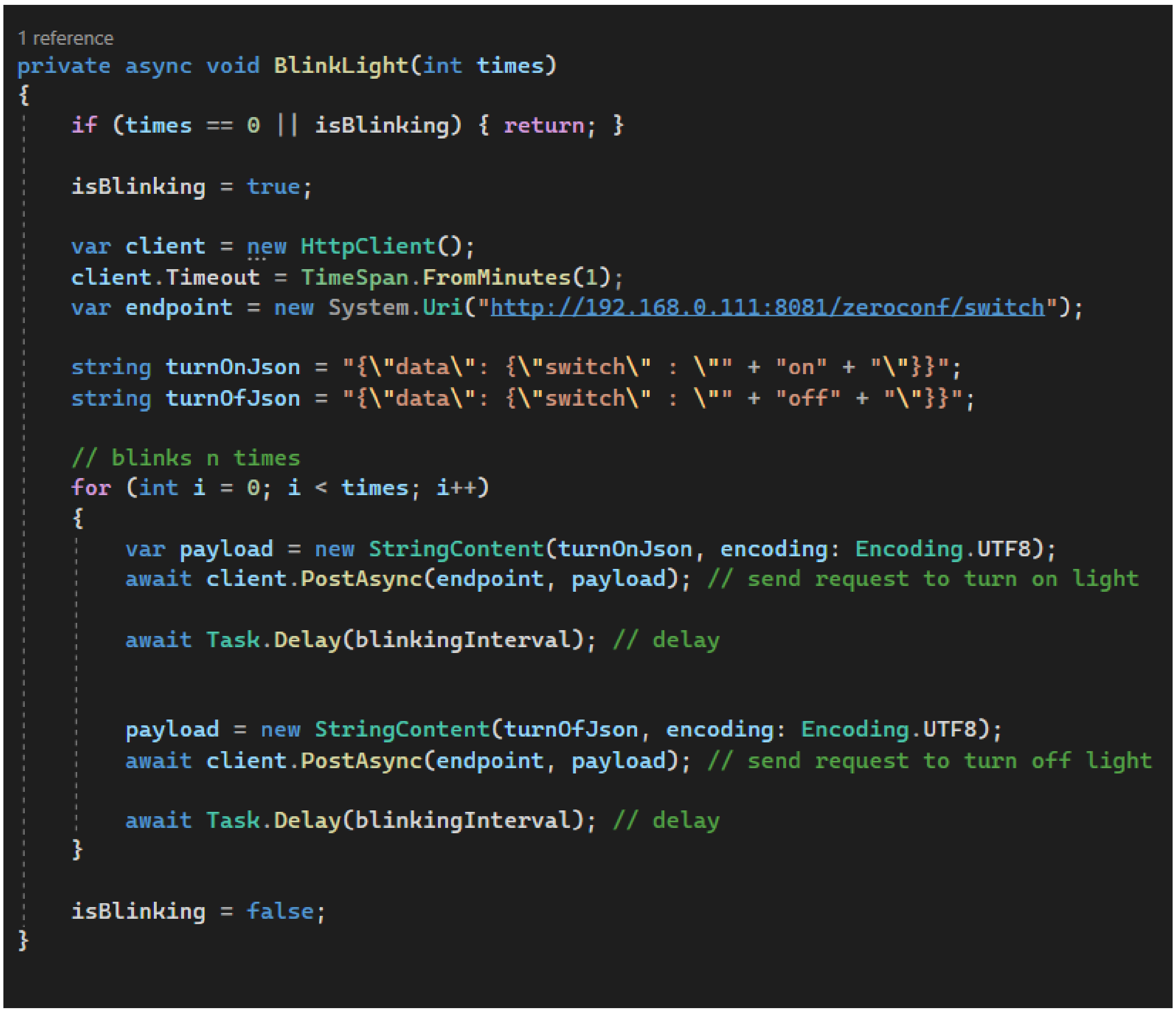
Duo to our inexperience with this technology, we couldn't make it work. Therefore, we turned to another resource, we decided to start a project with already built-in components for different target devices, in this case, we created a project for VR devices and copied the made-up game object that deals with the camera and hand controls interaction: "Complete XR Origin Set Up Variant". This game object not only deals with the camera movement, but also, implements two hand controls, corresponding to the left and right hands. The left hand is responsible for the movement, and the right one with the grabbing object action.

CANVAS

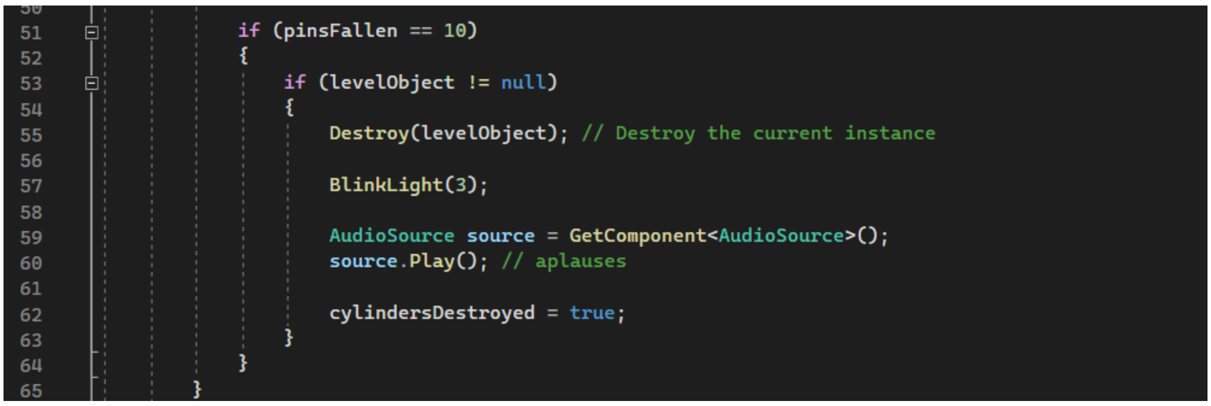
The canvas is the game object where the UI elements are placed. For this use case, only a simple score text game object was used. This is placed in the right-upper corner of the game camera, which will always be visible in that area, when the VR glasses are used. This allows the user to be always able to consult the score without having to pause the game or make any other related action.

LAMP

To turn on the lamp in real world, we implemented a function, **BlinkLight()**, on LevelManager C# script, where the only thing we need to do is connect to the HTTP client, and send a post request.



```
1 reference
private async void BlinkLight(int times)
{
    if (times == 0 || isBlinking) { return; }

    isBlinking = true;

    var client = new HttpClient();
    client.Timeout = TimeSpan.FromMinutes(1);
    var endpoint = new System.Uri("http://192.168.0.111:8081/zeroconf/switch");

    string turnOnJson = "{\"data\": {\"switch\" : \"" + "on" + "\"}}";
    string turnOfJson = "{\"data\": {\"switch\" : \"" + "off" + "\"}}";

    // blinks n times
    for (int i = 0; i < times; i++)
    {
        var payload = new StringContent(turnOnJson, encoding: Encoding.UTF8);
        await client.PostAsync(endpoint, payload); // send request to turn on light

        await Task.Delay(blinkingInterval); // delay


        payload = new StringContent(turnOfJson, encoding: Encoding.UTF8);
        await client.PostAsync(endpoint, payload); // send request to turn off light

        await Task.Delay(blinkingInterval); // delay
    }

    isBlinking = false;
}
```



```
                if (pinsFallen == 10)
                {
                    if (levelObject != null)
                    {
                        Destroy(levelObject); // Destroy the current instance

                        BlinkLight(3);

                        AudioSource source = GetComponent<AudioSource>();
                        source.Play(); // aplauses

                        cylindersDestroyed = true;
                    }
                }
            }
```

Remember that the lamp and the Meta Quest Oculus need to be connected to the same Wi-Fi.

FUTURE WORK

Although the project went well, and we successfully implemented many features, there is still significant room for improvement. In this chapter, we will elaborate on the areas where we believe our project falls short the most.

- Improve the physics when throwing the ball.
  Until the ball reaches the ground it lacks realistic physics, allowing us to throw it effortlessly in any direction we desire.

-