KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B166 Development of Computer Games and Interactive Applications

Escape the Lab VR

Done by:
IFF-7/14 group
Airidas Janonis
Eligijus Kiudys

Date:
2020-05-24

Kaunas, 2020

Tables of Contents

Tables of Images

Table of Tables/functions

Work Distribution Table:

Name/Surname	***Description of game development part***
Airidas Janonis	Mostly responsible for project management, game design, 2D assets (sprites, sprite sheets, etc.), game balancing, creating visual effects (unity VFX, particle systems), a smaller portion of programming
Eligijus Kiudys	Mostly responsible for a bigger portion of the programming, creating some of the 3D assets, implementing new systems into the game (Steam VR, Virtual reality toolkit VRTK, etc.), shader writing for game mechanics such as drawing, writing, creating some of visual effects.

Description of the Game

Description of the Game.

1. **2D or 3D?** 3D.
2. **Genres:** Casual, Indie, Simulation, Educational
3. **Platforms:** PC
4. **Scenario Description:** The main concept of the game was taken from laboratory facilities of KTU Chemistry faculty. The goal of the game is to escape from the laboratory in a certain amount of time. To progress further, the player has to look for clues and perform chemical reactions/experiments. After completion of these tasks, the player is awarded another clue or a part of a three-digit code, which is required for the exit door keylock

Realization tasks

Task #1. Create a short introduction level (Airidas)

The beginning level scene was made by using all kinds of different GameObjects:

- Models, that were created inside of a 3D modeling software Blender;
- Various light sources;
- Timer;
- Various Liquids, flasks;
- Other laboratory interior objects.

Figure 1. The beginning level scene with some used prefabs below

Figure 2. Objects inside of a shelf

Task #2. Create second level (Eligijus)

Second level scene was made by using all kinds of different GameObjects:

- Models, that were created inside of a 3D modeling software Blender;
- First level models;
- Various light sources;
- Timer;
- Various Liquids, flasks;
- Other laboratory interior objects.

Figure 3 Second level scene with some used prefabs below

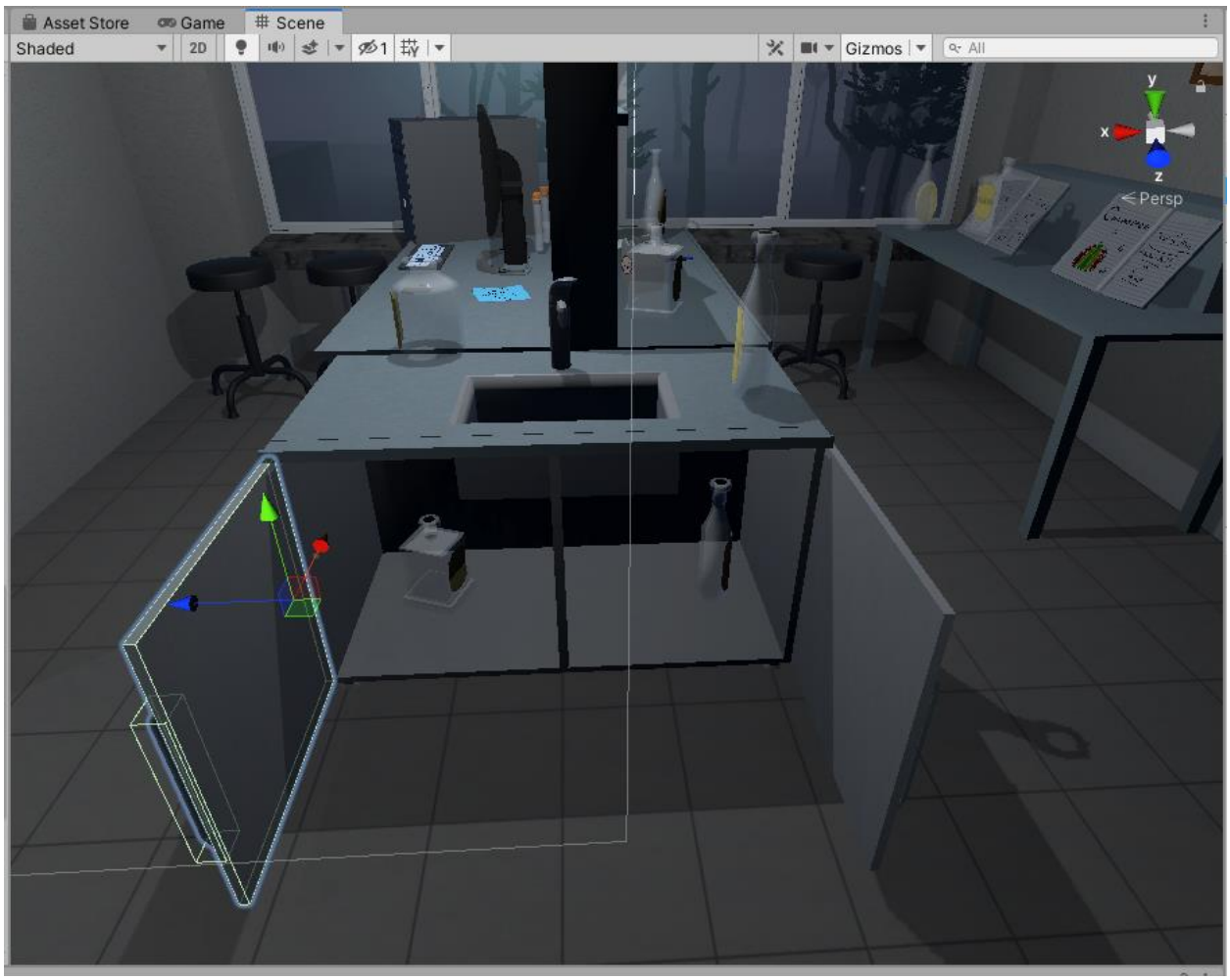

Figure 4 Objects inside of a shelf

Figure 5 Timer trigger

Task #3. Implement door unlock (level pass mechanics) (Airidas)

Door unlocking workflow:
To pass the level, player has to find a key, which is used to unlock the door. Key and doors have specific tags, which are evaluated. LockUnlockWithKey.cs script component is used to check key GameObject by tag – if the tag is correct, the script component unlocks the door.

```
public class LockUnlockWithKey : MonoBehaviour
{
    [SerializeField] GameObject[] Note;
    [SerializeField] GameObject doorsControll;
    [SerializeField] Rigidbody rb;
    [SerializeField] string checkKey;
    [SerializeField] bool x = false;
    [SerializeField] bool y = false;
    [SerializeField] bool z = false;
    bool isSnapped = false;
    IdForUnlock[] unlock;
    public bool Open = false;
    bool oneTime = true;
    bool check = false;
    private int unlockId = 0;
    bool rigidbodyExists = false;
    int index = 0;
       [SerializeField] DelegateChange Task;
    // Start is called before the first frame update
    void Start()
    {
        doorsControll.SetActive(false);
        foreach ( GameObject obj in Note )
        {
            obj.SetActive(false);
        }
        if (x)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationX;
        }
        else if (y)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationY;
        }
        else if (z)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationZ;
        }
    }
    // Update is called once per frame
    void Update()
    {
        if (!check && isSnapped)
        {
            unlock = FindObjectsOfType<IdForUnlock>();
            if (index < unlock.Length)
            {
                if (unlock[index] != null && unlock[index].tag == checkKey)
                {
                    int id = Random.Range(1, 20);
                    unlockId = id;
                    unlock[index].SetId(id);
                    check = true;
                }
                else if(unlock[index] != null && unlock[index].tag != checkKey)
                {
```

```
                    index++;
                }
            }
            else if(index >= unlock.Length)
            {
                index = 0;
            }
        }
        else
        {
            if (Open == true && oneTime)
            {

                doorsControll.SetActive(true);
                rb.constraints = RigidbodyConstraints.None;
                foreach (GameObject obj in Note)
                {
                    obj.SetActive(true);
                }
                oneTime = false;
                rigidbodyExists = true;
                           Task.AddTask();
                Destroy(this);
            }
            if (isSnapped && unlock[index].GetId() == unlockId)
            {
                Open = true;
            }
        }
    }
    public bool isChestOpen()
    {
        if (rigidbodyExists)
            return Open;
        else
            return false;
    }
    public void Snap()
    {
        isSnapped = true;
    }
    public void UnSnap()
    {
        isSnapped = false;
    }
}
```

Table 1 LockUnlockWithKey.cs script component

Task #4. Implement Virtual Reality player movement (in this case, teleportation) (Eligijus)

The Virtual reality basic controls, such as Teleportation were implemented with the help of VRTK (Virtual Reality Toolkit) plugin. The scripts and mechanics from VRTK are based on script inheritance.

There is a player's Head at the same position as a VR headset, which is actively tracked. The VR has a zone which is called play area, where player can walk freely until he reaches the end of this zone. If he steps out of the zone, the tracking of headset might be stopped.

The teleportation workflow:
First of all, the offset of player's head and the play area is calculated. After putting a finger on the teleportation button (most of the time it is touchpad button) the program reads one input, which enables curved raycast line. Another input is when the button is being pressed, which teleports the player to the targeted position (the last point of curved raycast line) by calculating the offset of the play area and the last position of the curved raycast line. Later on, the saved offset from beginning is restored and the player is spawned at the pointed location.

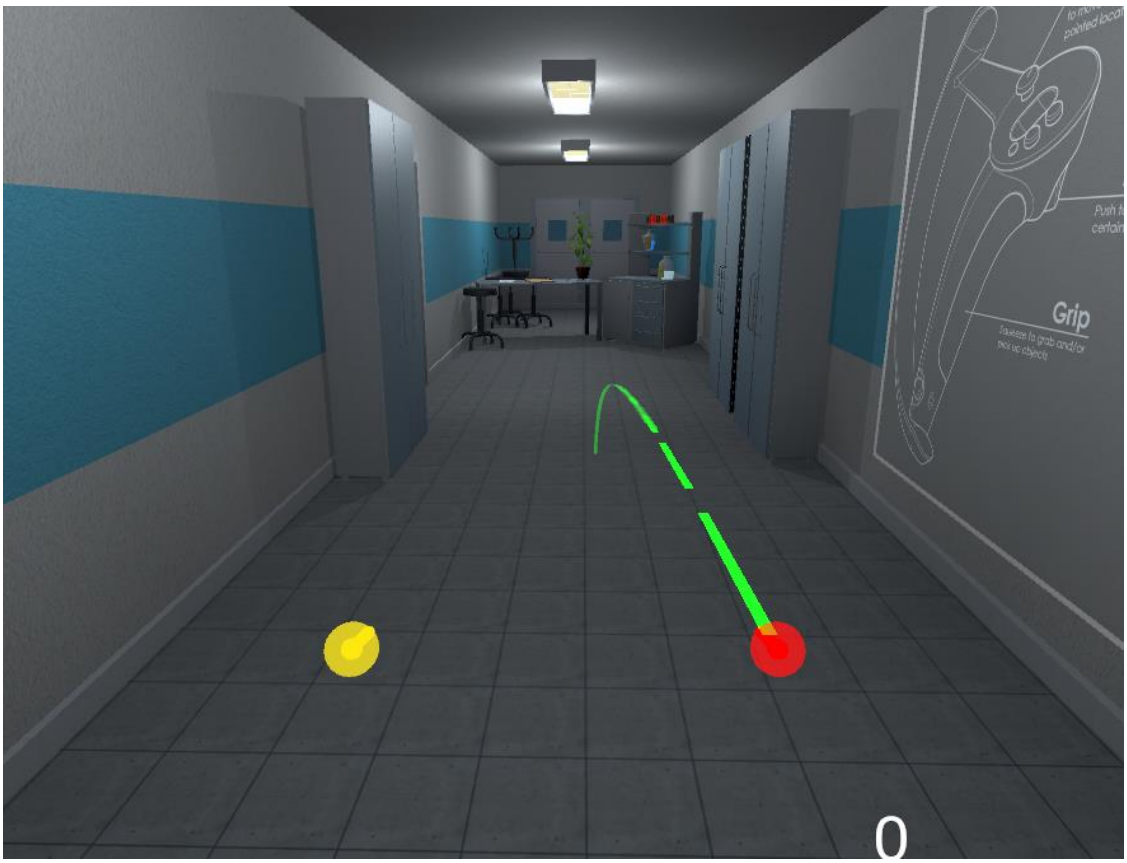

Figure 6 Correct teleportation

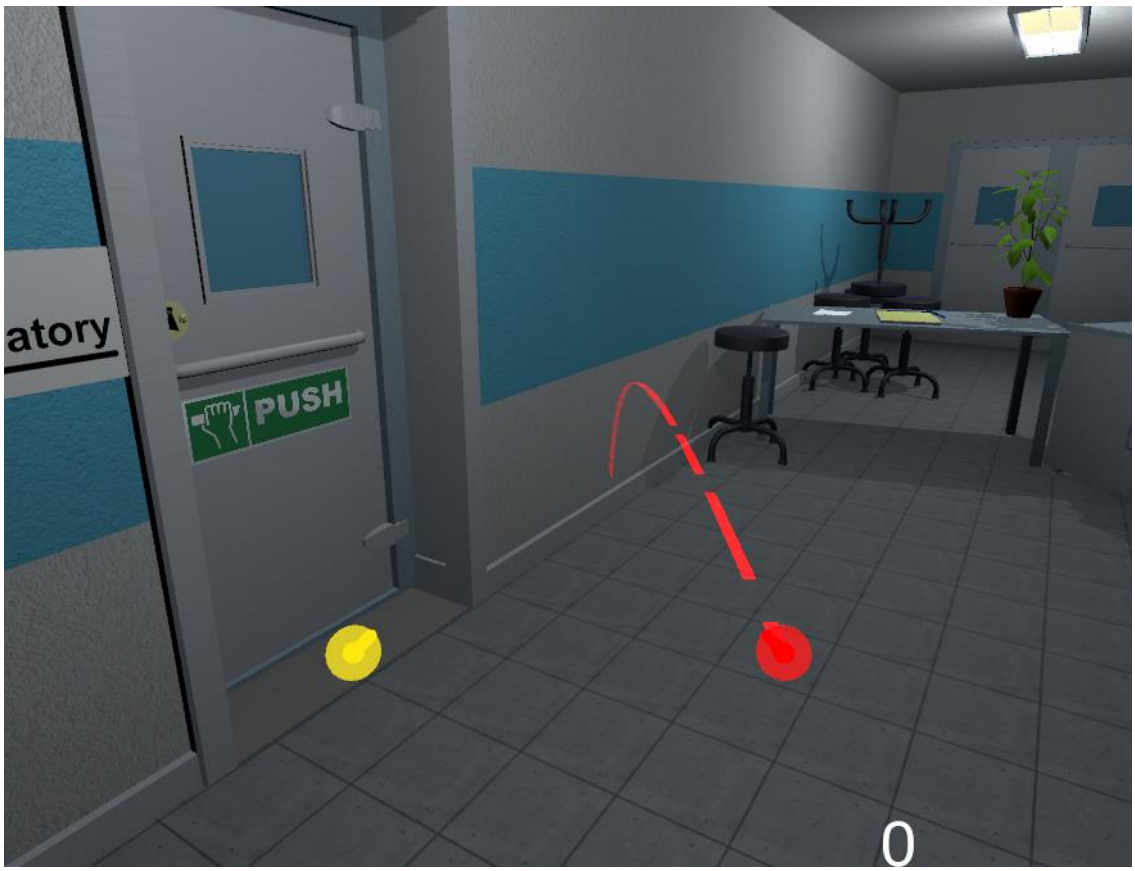

Figure 7 Incorrect teleportation

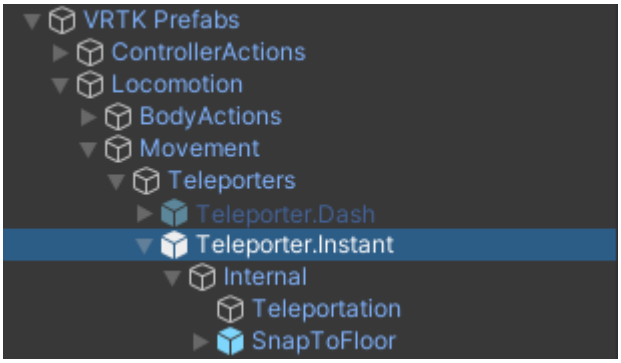

Figure 8 VRTK Teleportation hierarchy

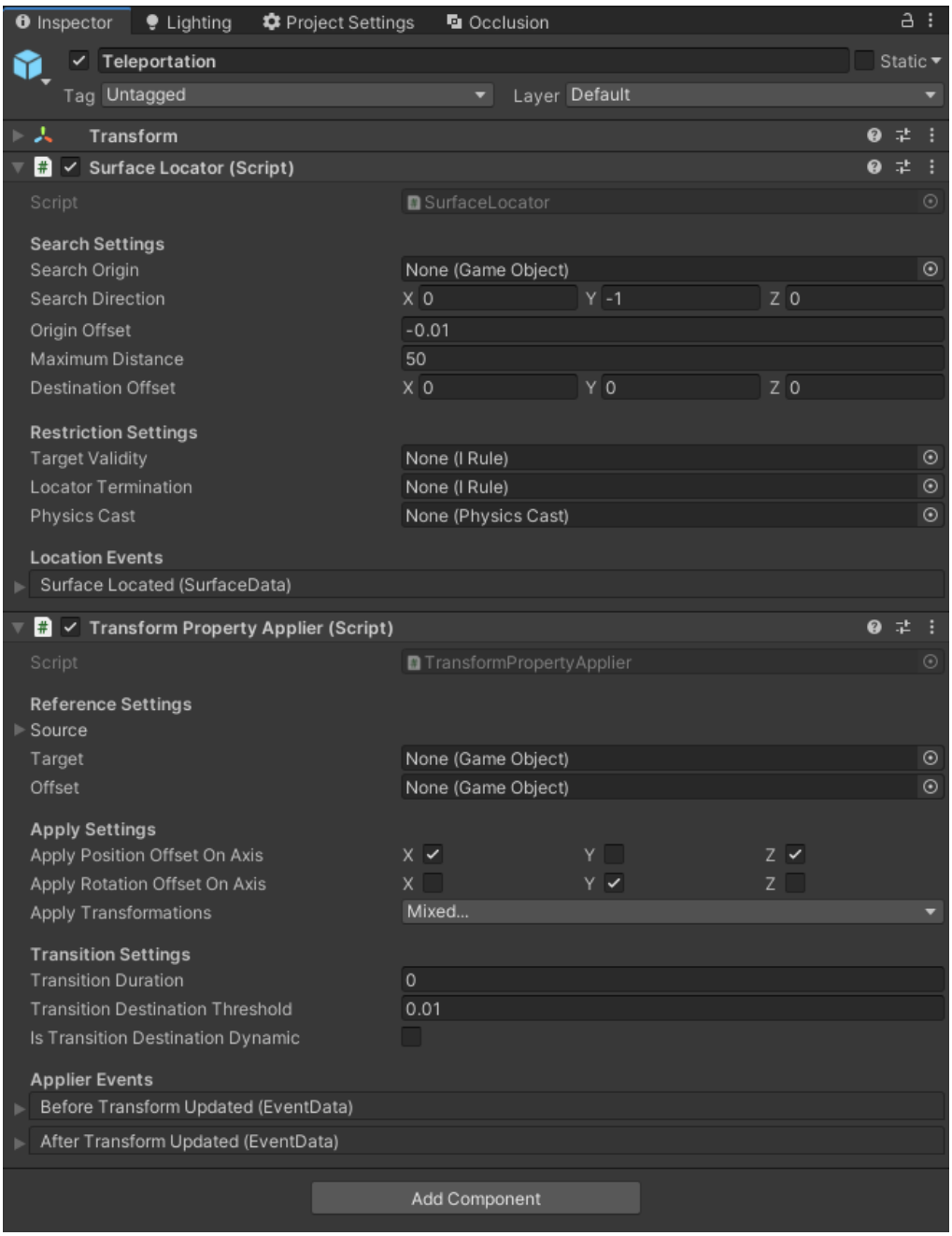

Figure 9 Main teleportation component inspector

Task #5. Decorate the level with at least 20 GameObjects, 5 Lights and 5 materials (Airidas)

The level has been decorated with objects that were modeled in Blender, as mentioned during the first task. Before using these objects inside of the scene, they have been given various components – colliders, rigidbodies, tags, layers, etc.

Figure 10 Scene decoration with various GameObjects

Some of the GameObjects that were used in the mentioned scene decoration are are shown at Figures 4-6.

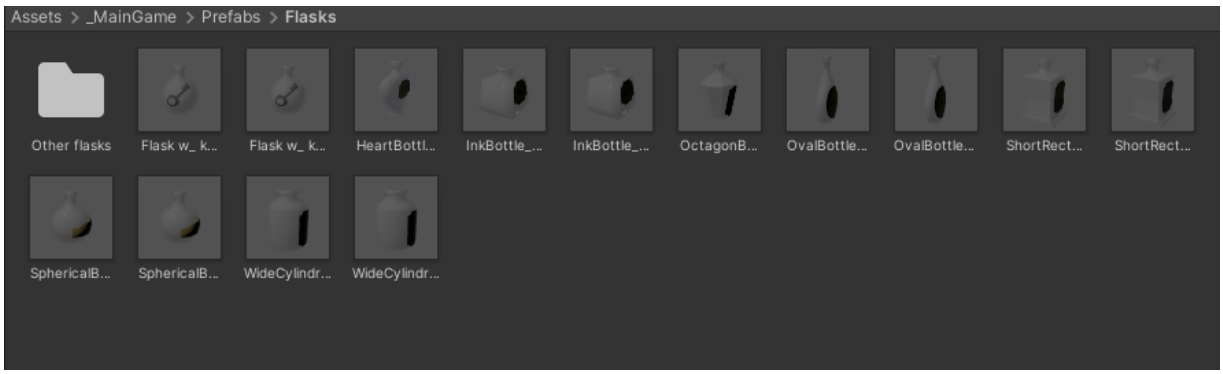

Figure 11. Objects used in the scene #1

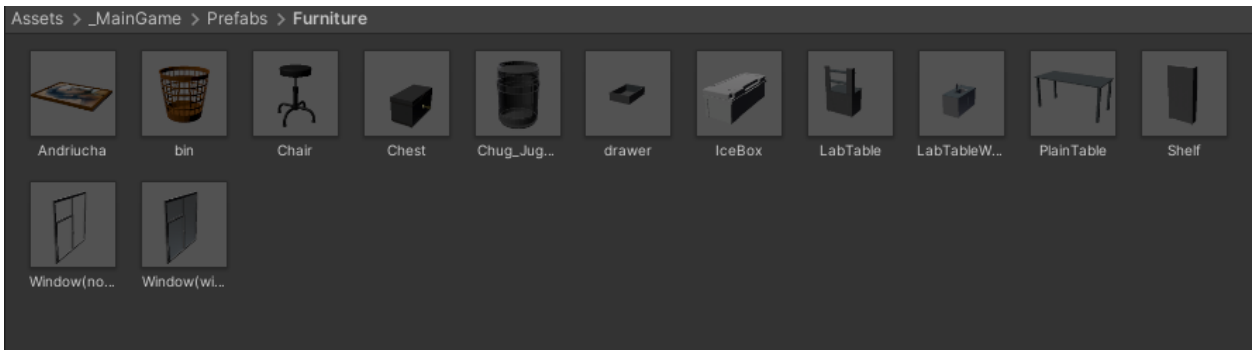

Figure 12. Objects used in the scene #2

Figure 13. Objects used in the scene #3

The light sources has been made with a few different models.

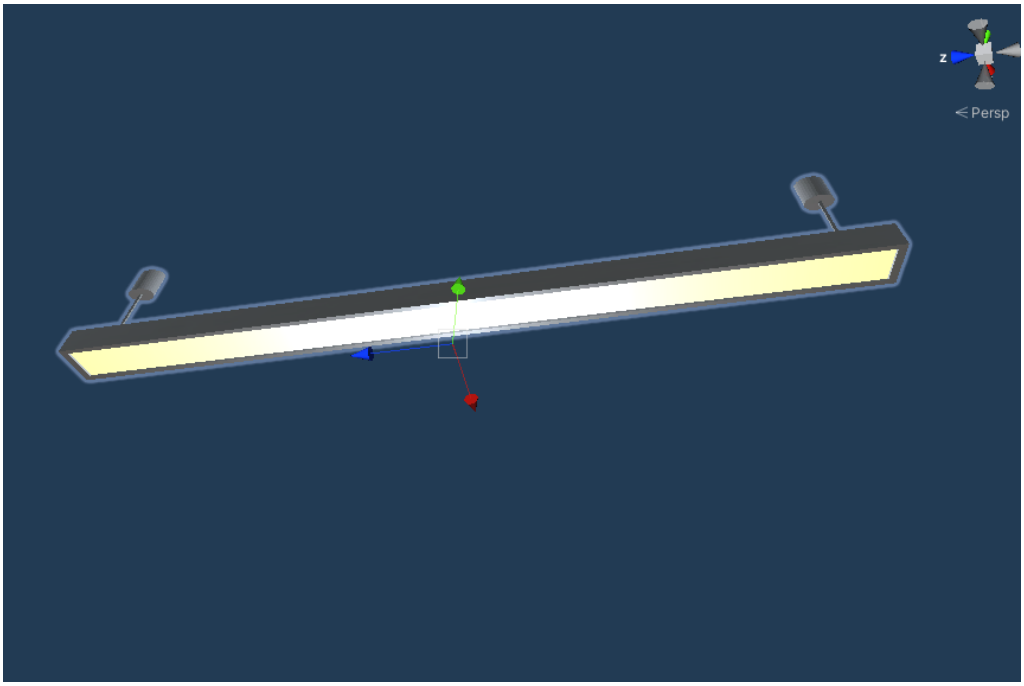

Figure 14. Lamp model #1

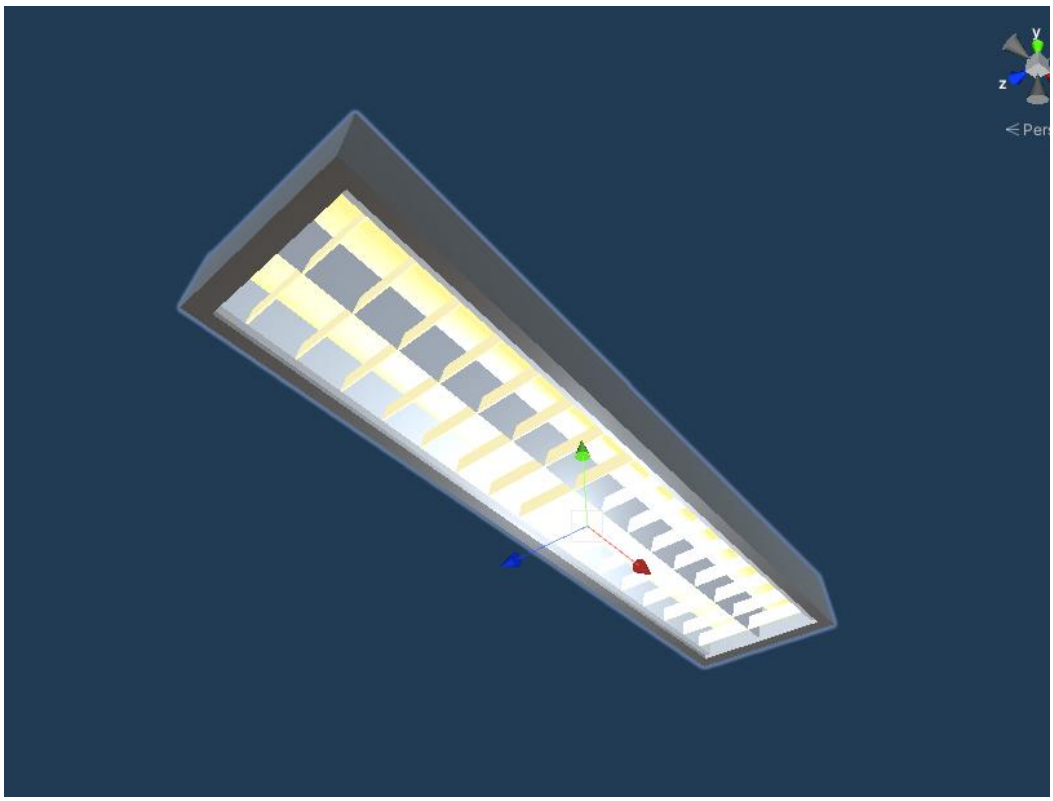

Figure 15. Lamp model #2

Task #6. Create a level timer mechanics (Airidas)

Timer workflow:
The timer is activated via TimeLeft.cs script component when the player hits a trigger collider (Figure 3). If the timer is over, player loses the game. If the player completes the level during the time limit, the timer stops via StopTime.cs script component.

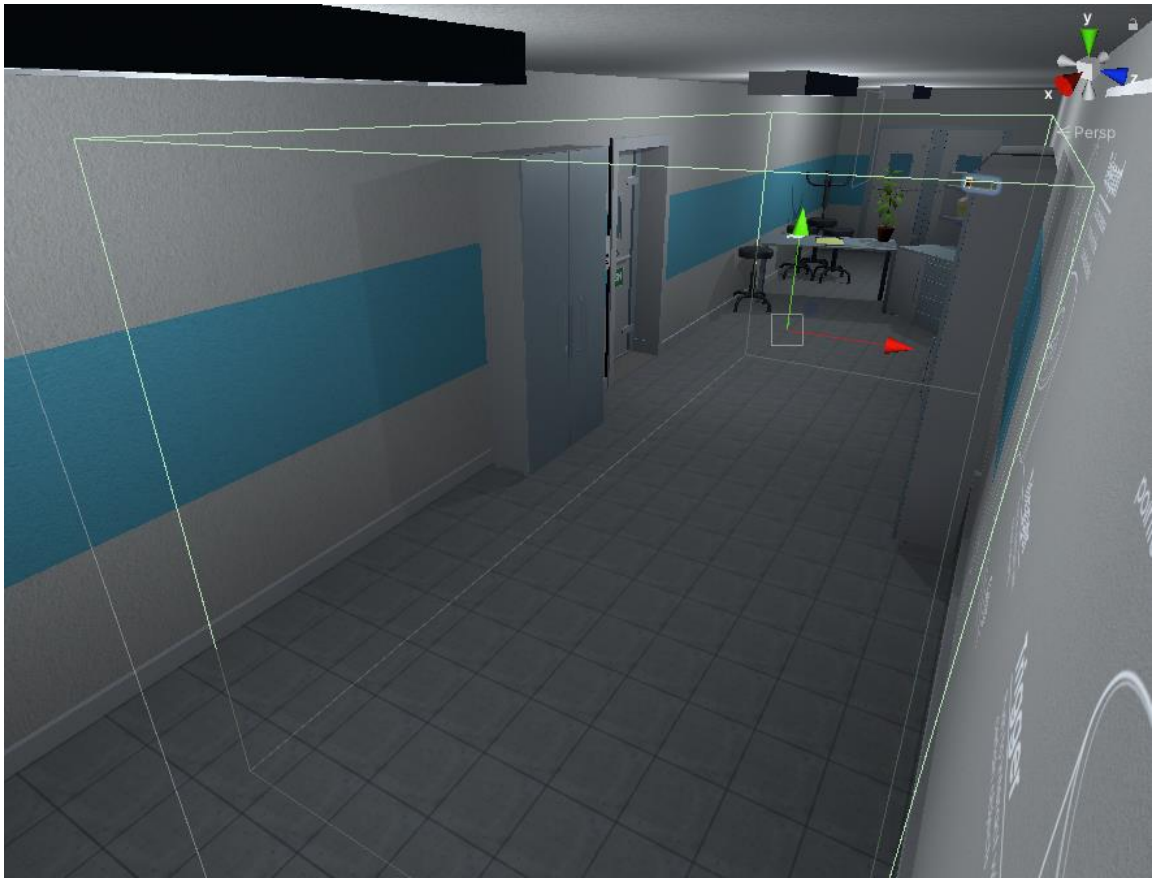

Figure 16. Timer trigger

```
public class TimeLeft : MonoBehaviour
{
    [SerializeField] TextMeshPro timerMinutes;
    [SerializeField] TextMeshPro timerSeconds;
    private float stopTime;
    public float timerTime;
    public bool finalStop = false;
    private bool isRunning = false;
    private bool check = false;

    private void Start()
    {
        StopTimer();
              GlobalData.LevelsTime += timerTime;
       }

    private void Update()
    {
        if (isRunning && !finalStop)
        {
            timerTime = stopTime + (timerTime - Time.deltaTime);
            int minutesInt = (int)timerTime / 60;
            int secondsInt = (int)timerTime % 60;
```

```
            timerMinutes.text = (minutesInt < 10) ? "0" + minutesInt :
minutesInt.ToString();
            timerSeconds.text = (secondsInt < 10) ? "0" + secondsInt :
secondsInt.ToString();
            if(timerTime <= 0)
            {
                GameData.SetEnd(true);
                GameData.SetVictory(false);
                isRunning = false;
            }
        }
              else if(!isRunning && finalStop && !check)
              {
                     OneTimeSend();
                     check = true;
              }
    }
    public void StopTimer()
    {
        isRunning = false;
       }
    public void TimerStart()
    {
        if(!isRunning)
        {
            print("timer is running");
            isRunning = true;
        }
    }
    public void MinusTime(float seconds)
    {
        timerTime -= seconds;
    }
       public void OneTimeSend()
       {
                     //Debug.Log(timerTime.ToString());
                     GlobalData.Timer.Add(timerTime);
       }
}
```

Table 2. Timeleft.cs script component

```
public class StopTime : MonoBehaviour
{
    public LockUnlockWithKey locks;
    [SerializeField] TimeLeft time;
    // Start is called before the first frame update

    // Update is called once per frame
    void Update()
    {
        if (locks.isChestOpen())
        {
            time.StopTimer();
            time.finalStop = true;
                    //time.OneTimeSend();
        }
    }
}
```

Table 3. StopTime.cs script component

Task #7. Create flask breaking onCollisionEnter (Eligijus)

Implemented the functionality by adding BootleSmash.cd script to GameObject.
Script checks if GameObject is not in the hand and it collides. onCollisionEnter checks how fast it collides if it faster than set minimum limit then flask shards is enabled and breaking particles are enabled, after particles and flask shard are enabled game object is destroyed. Shards after 5 seconds are destroyed.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BottleSmash : MonoBehaviour {

       // Use this for initialization
    //all of the required items in order to gie the impression of hte glass breaking.
    [ColorUsageAttribute(true, true, 0f, 8f, 0.125f, 3f)]
    public Color color;
    //to use to find any delta
    [HideInInspector]
    private Color cachedColor;
    //used to update any colors when the value changes, not by polling
    [SerializeField]
    [HideInInspector]
    private List<ColorBase> registeredComponents;

    public GameObject Cork, Liquid, Glass, Glass_Shattered, Label;
    //default despawn time;
    public float DespawnTime = 5.0f;

    public float time = 0.5f;

    float tempTime;

    float clacTime;

    public float splashLevel = 0.006f;

    public bool colided = false;

    //splash effect.
    public ParticleSystem Effect;
    //3D mesh on hte ground (given a specific height).
    public GameObject Splat;
    //such as the ground layer otherwise the broken glass could be considered the
'ground'
    public LayerMask SplatMask;
    //distance of hte raycast
    public float maxSplatDistance = 5.0f;
    //if the change in velocity is greater than this THEN it breaks
    public float shatterAtSpeed = 2.0f;
    //if the is disabled then it wont shatter by itself.
    public bool allowShattering = true;
    //if it collides with an object then and only then is there a period of 0.2f
seconds to shatter.
    public bool onlyAllowShatterOnCollision = true;
    //for the ability to find the change in velocity.
    [SerializeField]
    [HideInInspector]
    private Vector3 previousPos;
    [SerializeField]
    [HideInInspector]
    private Vector3 previousVelocity;
```

```
    [SerializeField]
    [HideInInspector]
    private Vector3 randomRot;
    [SerializeField]
    [HideInInspector]
    private float _lastHitSpeed = 0;
    //dont break if we have already broken, only applies to self breaking logic, not by
calling Smash()
    public bool broken = false;
    //timeout
    float collidedRecently = -1;

    LiquidVolumeAnimator lva;

    SteamVrSceleton skeleton;

       void Start () {
        if (Liquid != null)
        {
            lva = Liquid.GetComponent<LiquidVolumeAnimator>();
        }
        skeleton = GetComponent<SteamVrSceleton>();
        clacTime = time;
        tempTime = time;
        previousPos = transform.position;

       }


       //Smash function so it can be tied to buttons.
    public void RandomizeColor()
    {
        color = new Color(Random.Range(0, 1), Random.Range(0, 1), Random.Range(0, 1),
1);
    }
    void OnCollisionEnter(Collision collision)
    {
        //set a timer for about 0.2s to be able to be broken
        _lastHitSpeed = collision.impulse.magnitude;
        if (collision.transform.tag != "Liquid" && collision.transform.tag !=
"Absorver")
        {
            //Debug.Log(collision.transform.name);
            collidedRecently = 0.2f;
        }

    }

    public void AttemptCollision(Collision col)
    {
        OnCollisionEnter(col);
    }

    public void RegisterColorBase(ColorBase cb)
    {
        registeredComponents.Add(cb);
    }

    public void ChangedColor()
    {
        if(cachedColor != color)
        {
            cachedColor = color;

            //update all registered components
```

```
            foreach (ColorBase cb in registeredComponents)
            {
                cb.Unify();
            }
        }
    }
    public Vector3 GetRandomRotation()
    {
        return randomRot;
    }
    public void RandomRotation()
    {
        randomRot = (Random.insideUnitSphere + Vector3.forward).normalized;
    }

    public void Smash()
    {

        skeleton.UnGrab();
        broken = true;
        //the Corks collider needs to be turned on;
        if (Cork != null)
        {
            Cork.transform.parent = null;
            Cork.GetComponent<Collider>().enabled = true;
            Cork.GetComponent<Rigidbody>().isKinematic = false;
            Destroy(Cork.gameObject, DespawnTime);
        }
        //the Liquid gets removed after n seconds
        if (Liquid != null)
        {
            float t = 0.0f;
            //if (Effect != null)
            //    t = (Effect.main.startLifetime.constantMin +
Effect.main.startLifetime.constantMax)/2;
            Destroy(Liquid.gameObject, t);
        }
        //particle effect
        if(Effect != null && lva != null && lva.level > splashLevel)
        {
            Effect.Play();
            Destroy(Effect.gameObject, Effect.main.startLifetime.constantMax);
        }
        else if (Effect != null && lva != null && lva.level < splashLevel)
        {
            Destroy(Effect.gameObject, Effect.main.startLifetime.constantMax);
        }
        else if (Effect != null && lva == null)
        {
            Destroy(Effect.gameObject, Effect.main.startLifetime.constantMax);
        }

        //now the label;
        if (Label != null)
        {
            //Label.transform.parent = null;
            //Label.GetComponent<Collider>().enabled = true;
            //Label.GetComponent<Rigidbody>().isKinematic = false;
            Destroy(Label.gameObject);
        }
        //turn Glass off and the shattered on.
        if (Glass != null)
        {
            Destroy(Glass.gameObject);
        }
```

```
        if (Glass_Shattered != null)
        {
            Glass_Shattered.SetActive(true);
            Glass_Shattered.transform.parent = null;
            Destroy(Glass_Shattered, DespawnTime);
        }

        //instantiate the splat.
        RaycastHit info = new RaycastHit();
        if(Splat != null)
        if (Physics.Raycast(transform.position, Vector3.down, out info,
maxSplatDistance, SplatMask))
        {
            GameObject newSplat = Instantiate(Splat);
            newSplat.transform.position = info.point;

        }
        Destroy(transform.gameObject, DespawnTime);

    }
      // Update is called once per frame, for the change in velocity and all that
jazz...
      void FixedUpdate () {
        ChangedColor();
        collidedRecently -= Time.deltaTime;
        Vector3 currentVelocity = (transform.position - previousPos) /
Time.fixedDeltaTime;
        if ((onlyAllowShatterOnCollision && collidedRecently >= 0.0f) |
!onlyAllowShatterOnCollision)
        {
            if (allowShattering)
            {
                if (Vector3.Distance(currentVelocity, previousVelocity) >
shatterAtSpeed || _lastHitSpeed > shatterAtSpeed)
                {
                    if (!broken)
                        Smash();
                }
            }
        }
        _lastHitSpeed = 0;

        previousVelocity = currentVelocity;
        previousPos = transform.position;
      }

    public void ResetVelocity()
    {
        previousVelocity = Vector3.zero;
        previousPos = transform.position;
    }
}
```

Table 4 BottleSmah.cs script component

Task #8. Experiment with Polybrush / SpriteShapes / ProBuilder / Tilemap Palette tools, start expanding, building your World environment (Airidas)

The new level terrain was created by using unity built-in terrain system and some additional Game Objects:

- Models, that were created inside of a 3D modeling software Blender;
- Custom skybox
- Unity Wind Zone component;
- Particle system for clouds;
- Tree's from Unity standard asset pack.

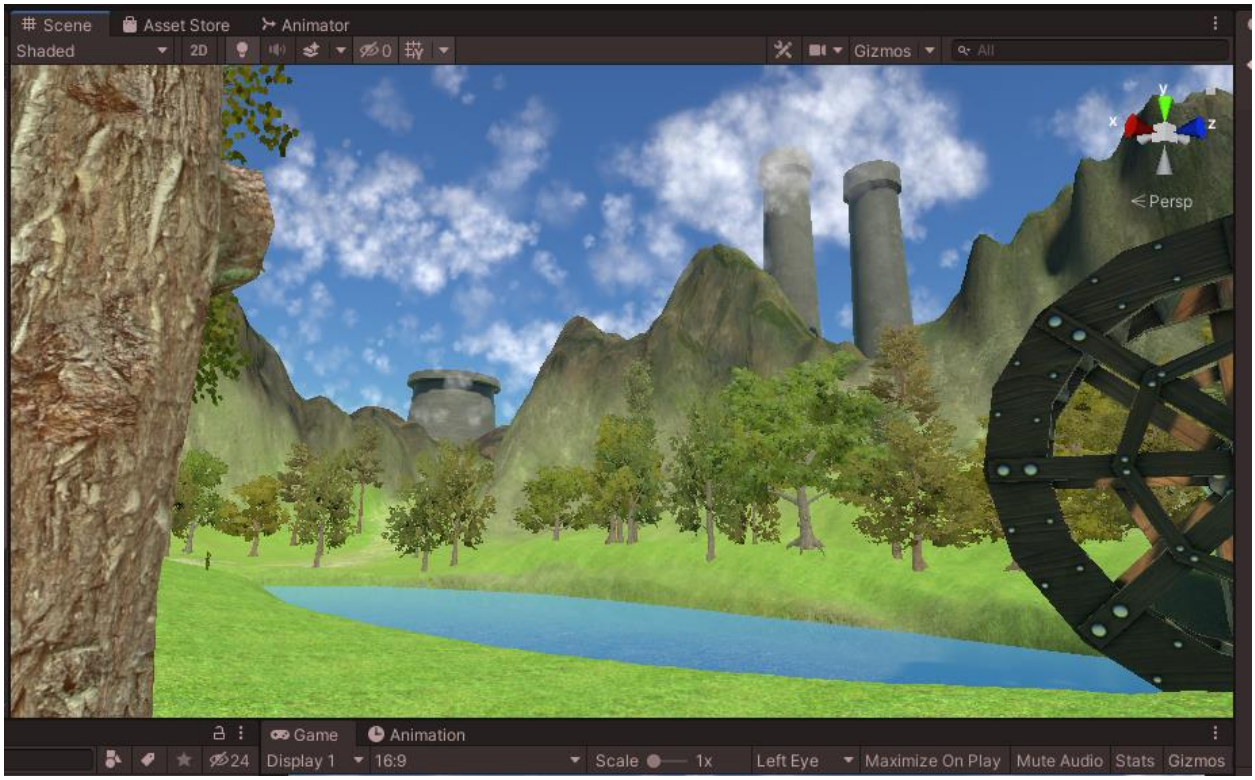

Figure 17 New level Terrain

Figure 18 New level Terrain

Task #9. Experiment with Polybrush / SpriteShapes / ProBuilder / Tilemap Palette tools, start expanding, building your World environment (Eligijus)

Created main scene terrain with polybrush and unity terrain tool.

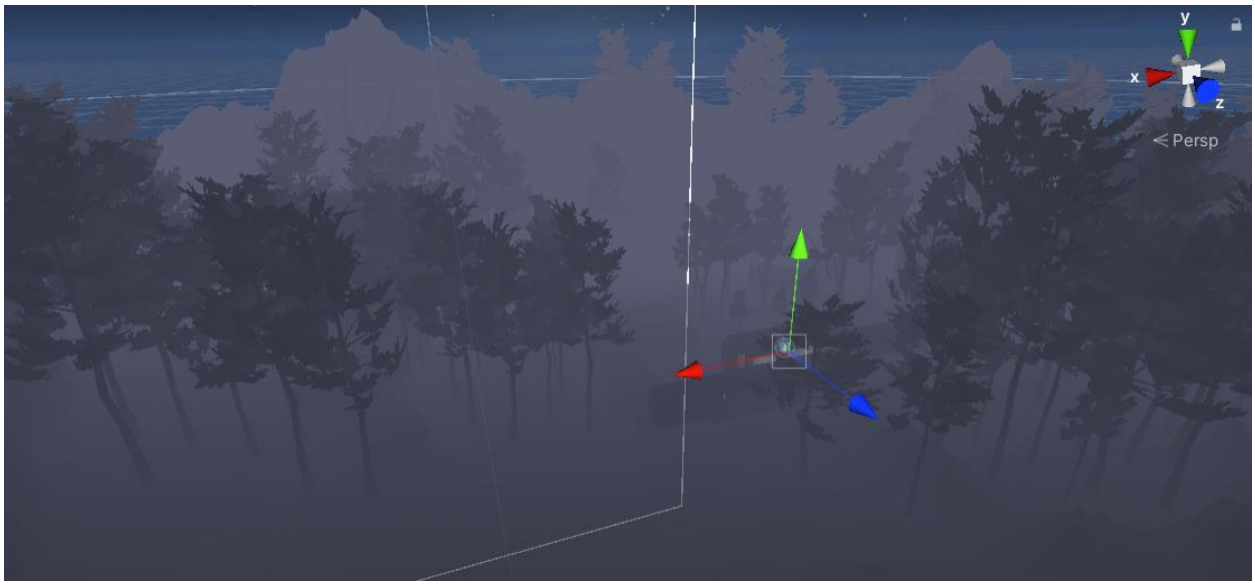

Figure 19 Created terrain

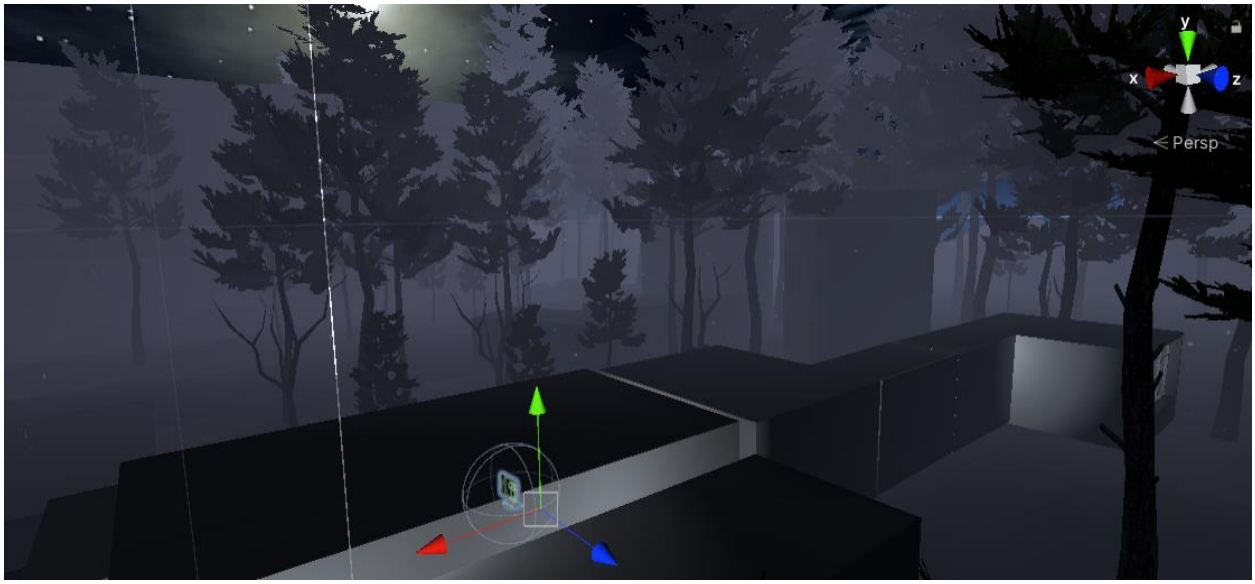

Figure 20 Created terrain

In this scene I was using polybrush for hills and trees

Task #10 Add animations to your game character (Assuming you have a working PlayerController) (Airidas and Eligijus)

In Escape the Lab the player is the game character, which means that the character's animations are based on real-time inputs through the Virtual Reality controllers – currently only the hands and fingers are animated. There are a few different approaches to these hand inputs:

- If the character hands are controlled with Valve Index device controllers, the in-game hands and fingers will be moved accordingly;
- If the character hands are controlled with HTC Vive, Oculus Rift or any other similar controller, the fingers moves inwards when the player puts his fingers on a button, and it does the opposite when the player removes his finger from the button.

Figure 21 Hand movement

Figure 22 Hand movement

Figure 23 Hand movement

Task #11 Create and/or animate 5 objects of your choice in your World (you can use Animator Component and/or Timeline (Airidas)

The game is fully interactable, which means that there are loads of objects that are mostly based on physics and collisions, not animations.
Some of the animated objects:

- Flask humanoid – when the player comes too close to the flask, the humanoid flask gets scared away and hides under a shelf
- Flickering lights – corridor lights are flickering from time to time
- Driving vehicle – a car that drives through the terrain from time to time.

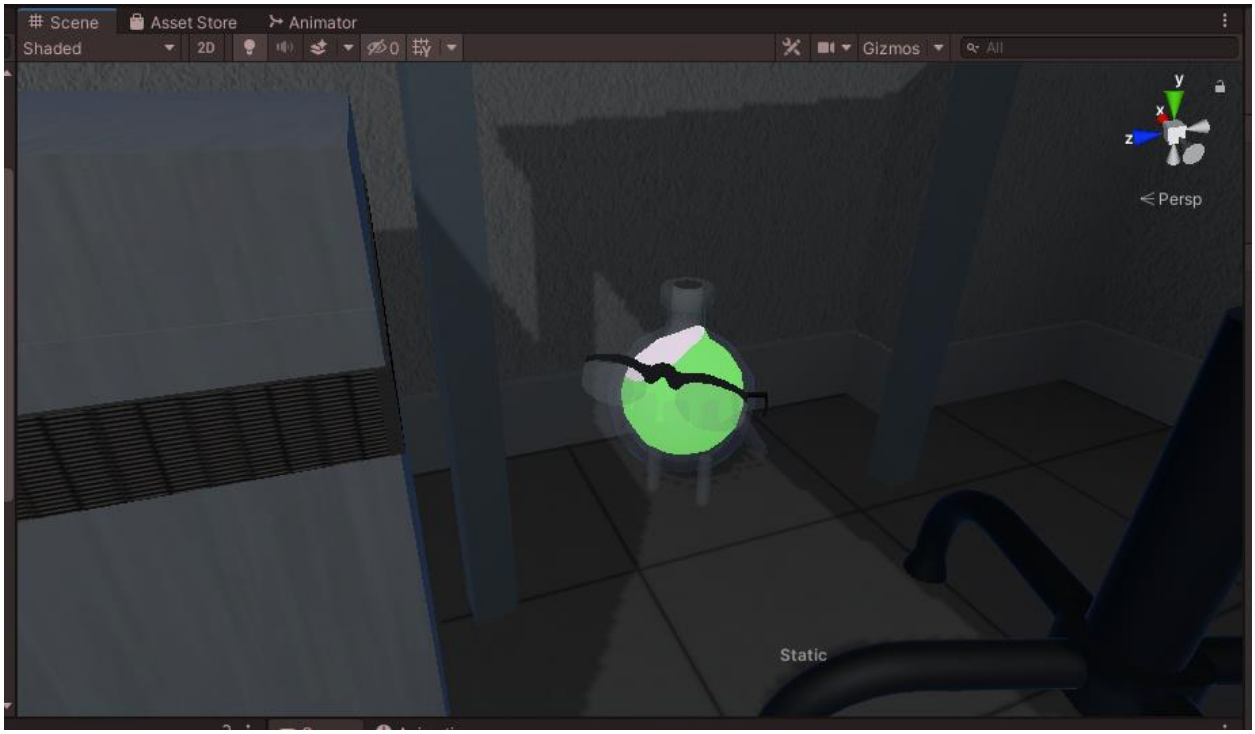

Figure 24 Flask humanoid

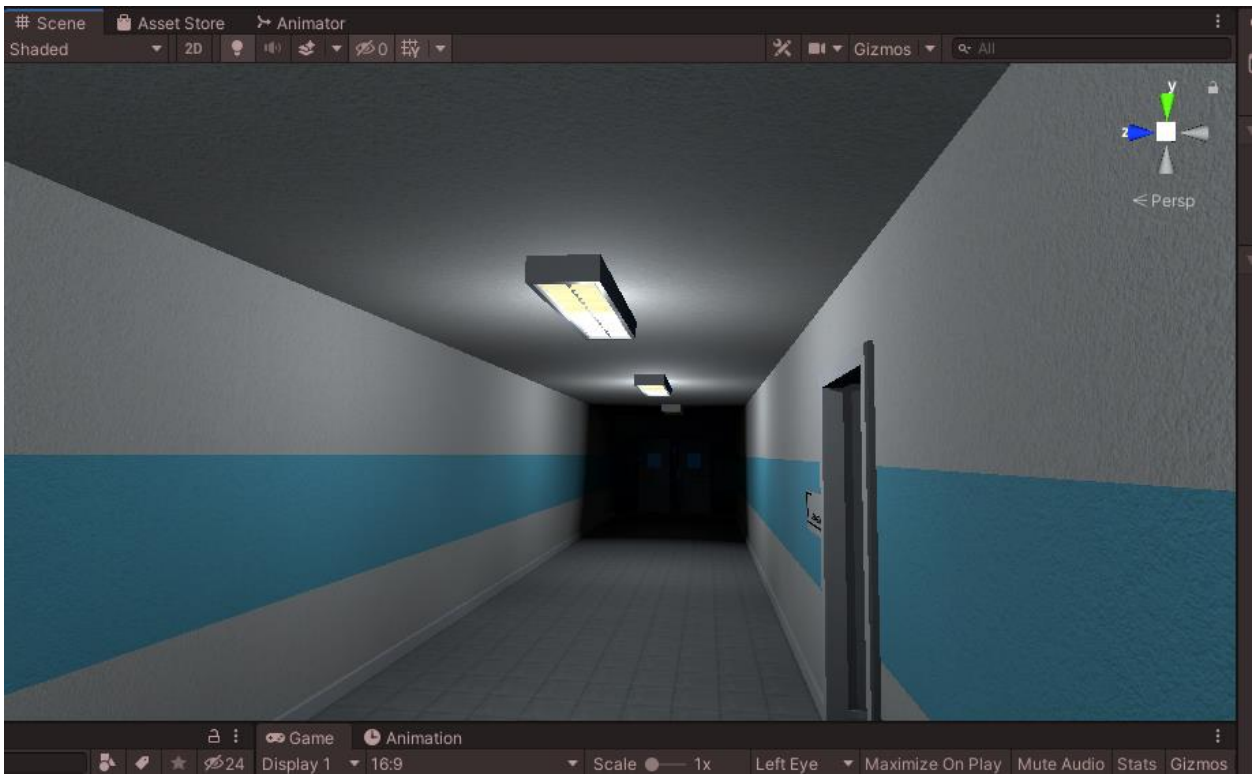

Figure 25 Corridor lights

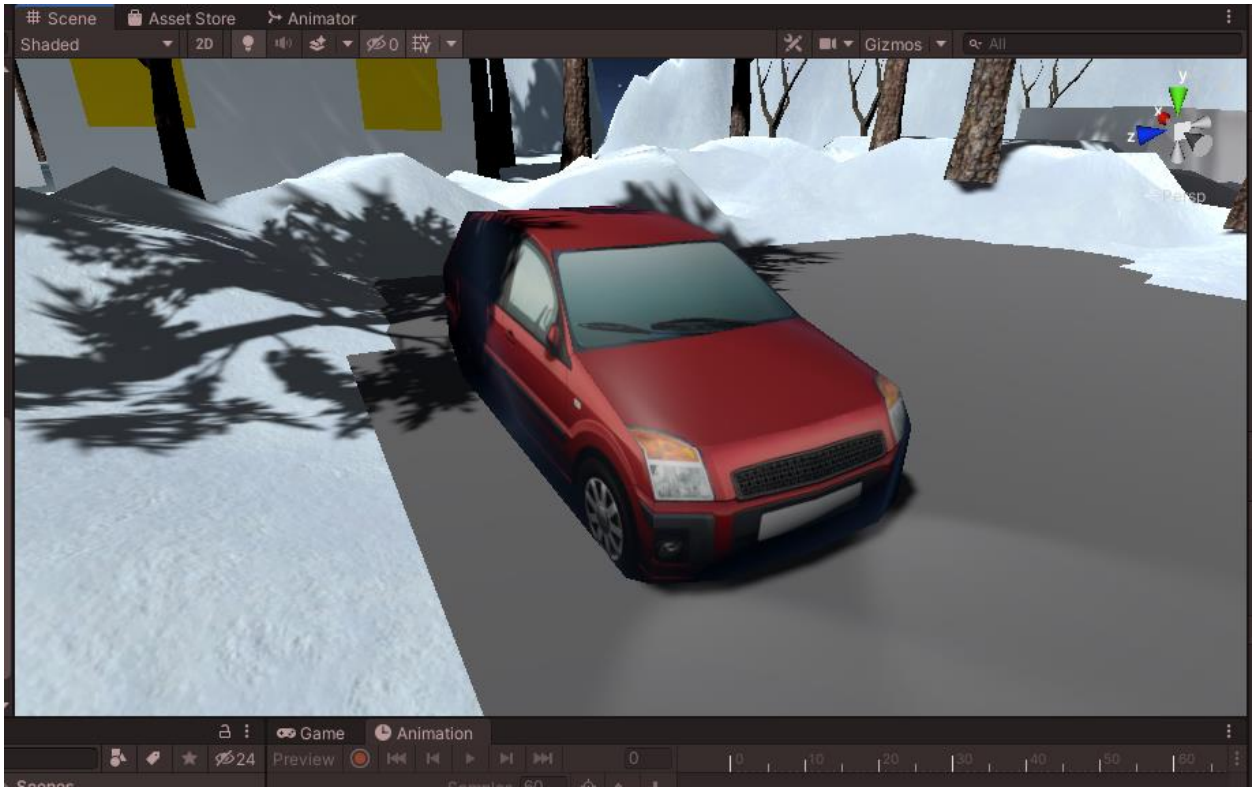

Figure 26 Vehicle

Task #12 Create at least 5 particle effects for your environment (dust, explosion, smoke gas, light sparks, etc) (Eligijus)

In this project we use two particle systems physic and graphics. Almost all particles are using physics system because VFX doesn't interact with physic colliders.

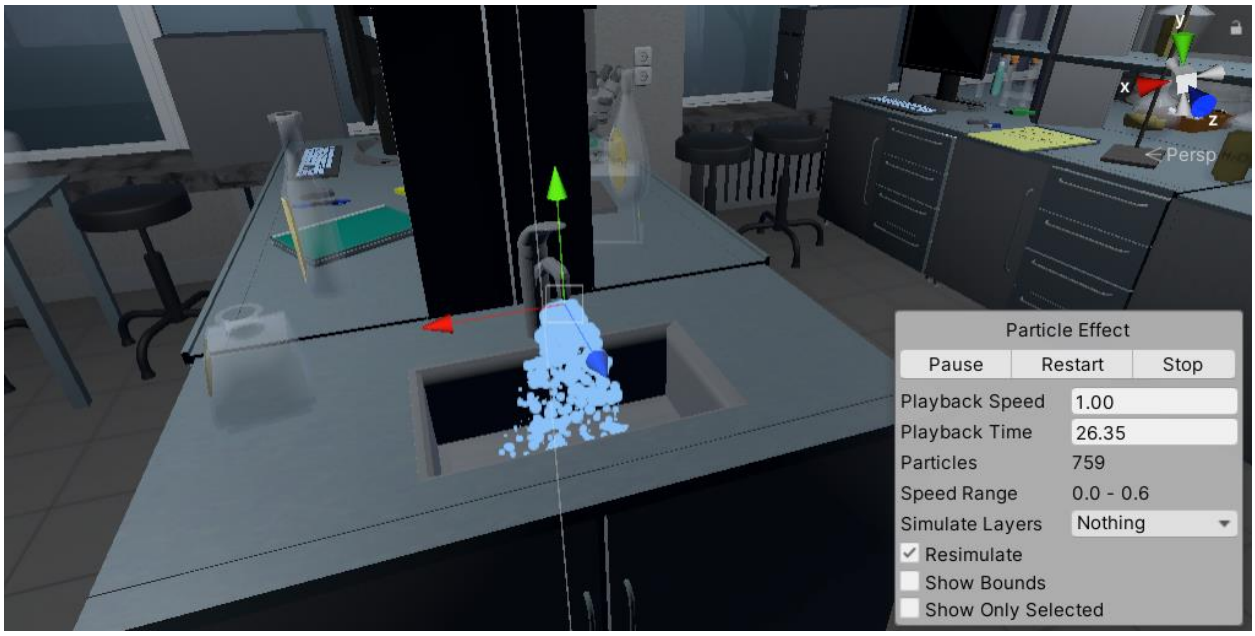

Figure 27 Water particles

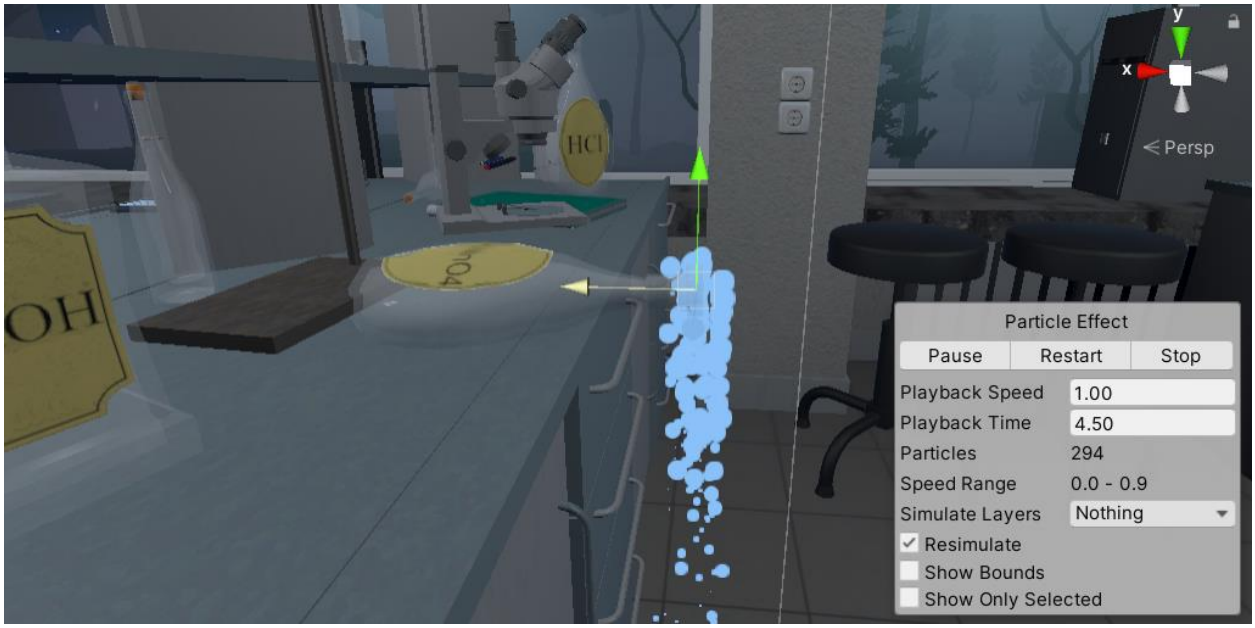

Figure 28 Liquid chemical particles

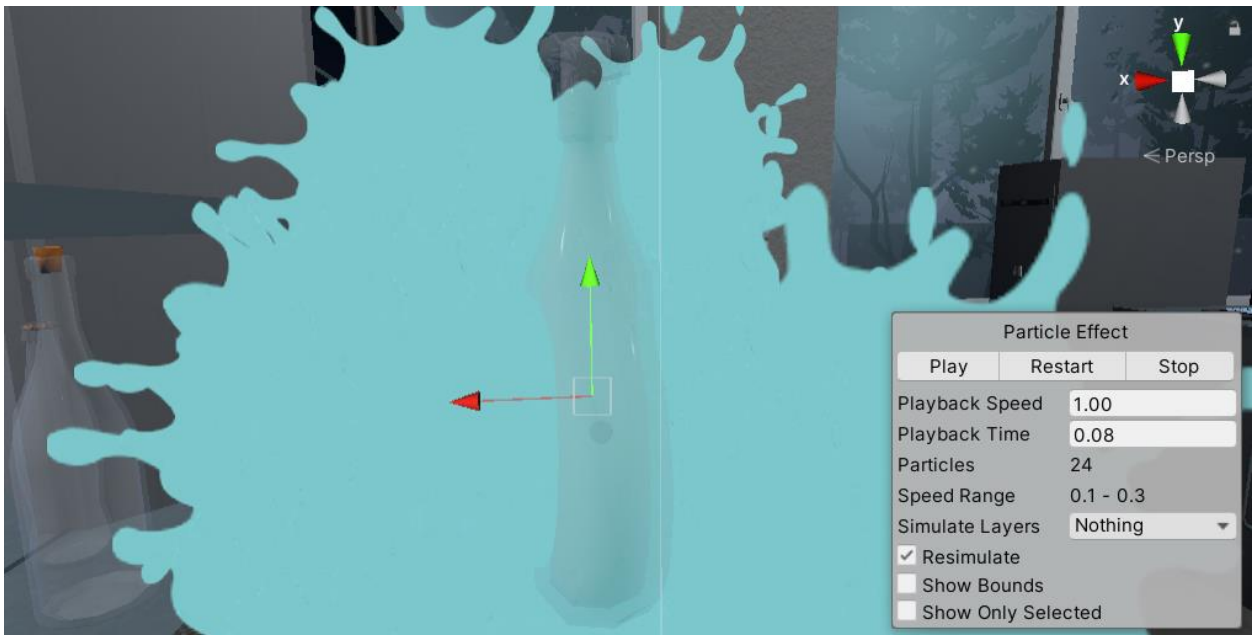

Figure 29 Flask smash particle

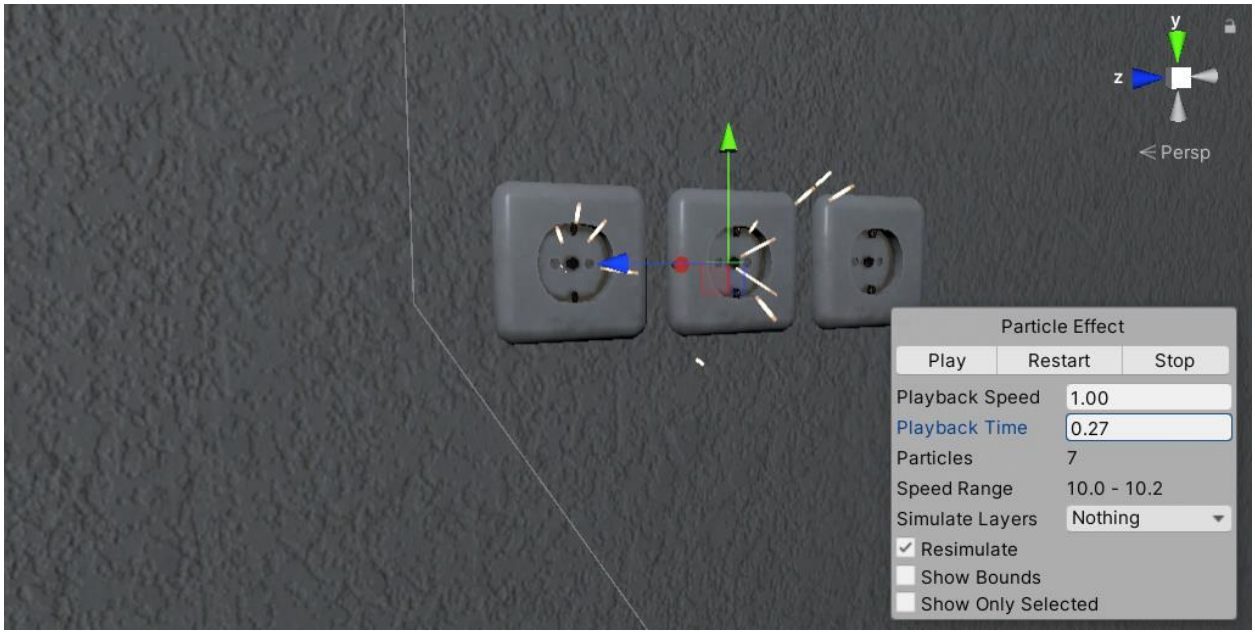

Figure 30 Outlet particles

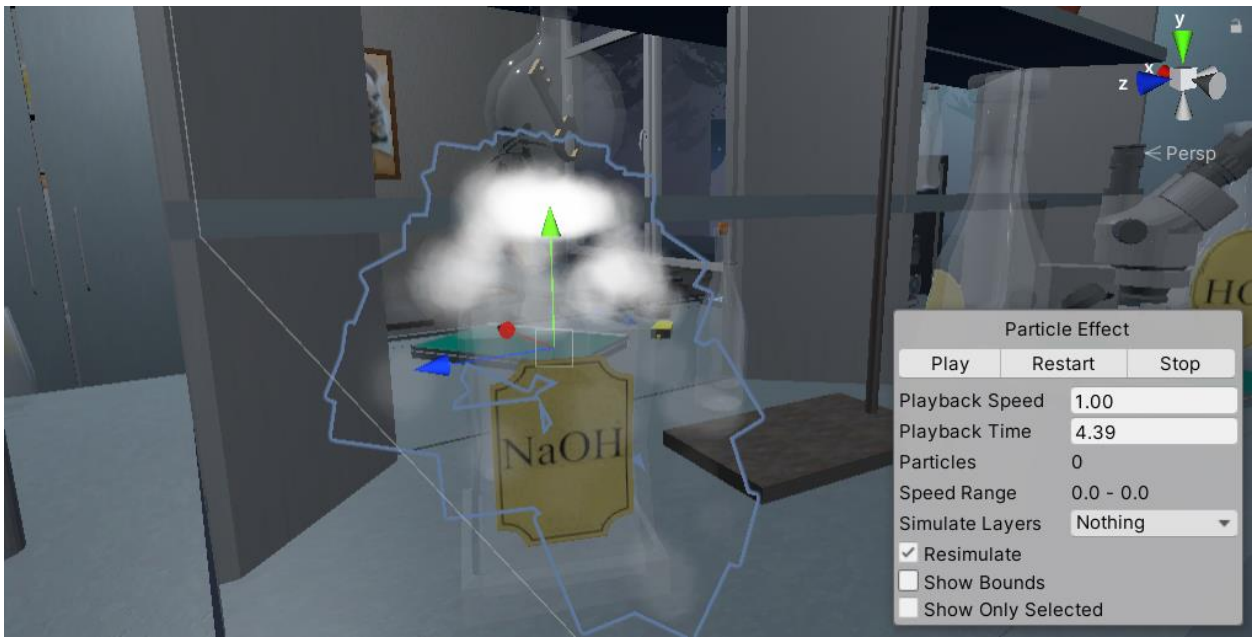

Figure 31 Dry ice reaction

Task #13 Create at least 5 particle effects for your environment (dust, explosion, smoke gas, light sparks, etc) (Airidas)

There are loads of different particle effects in the game.
Some of these particles:

- Snowflakes (made with VFX)
- Clouds (made with particle system)
- Thick smoke (made with VFX)
- Lighter fire (made with particle system)
- Blue fire (made with particle system)

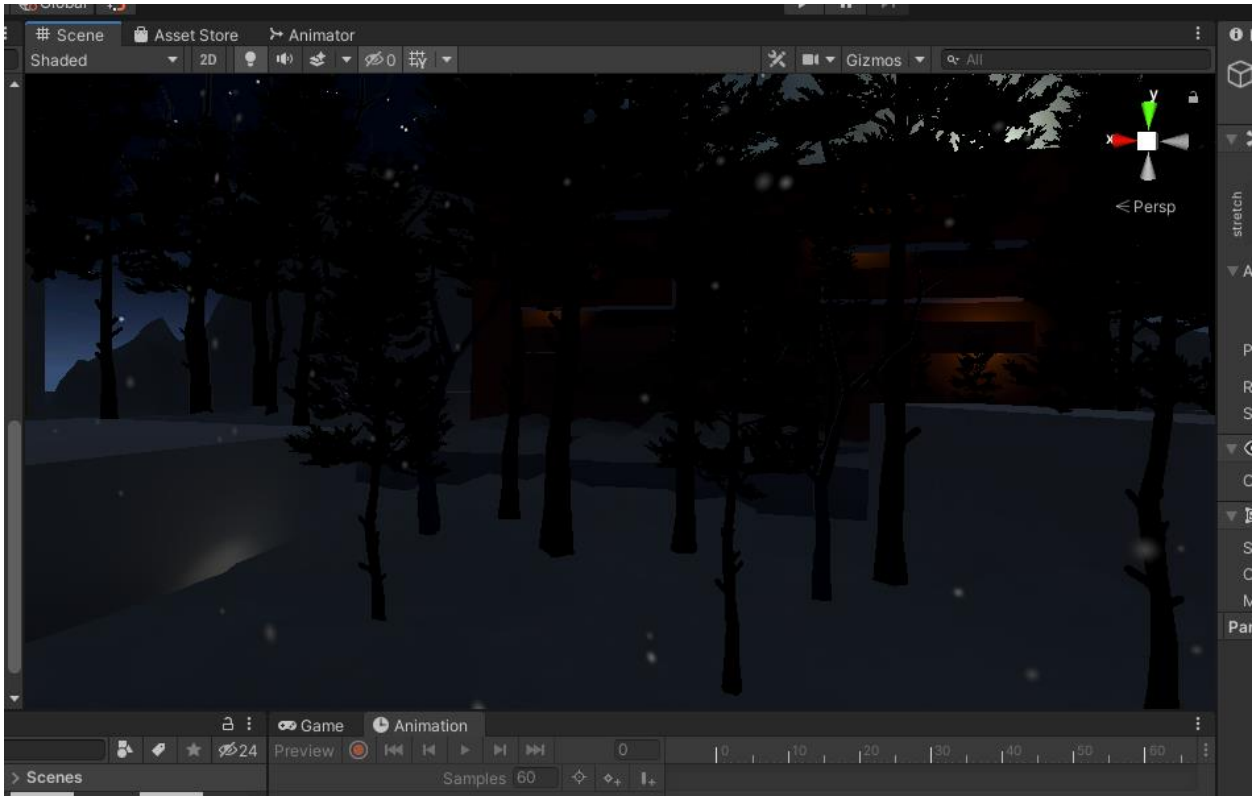

Figure 32 Snowflakes particles

Figure 33 Clouds VFX

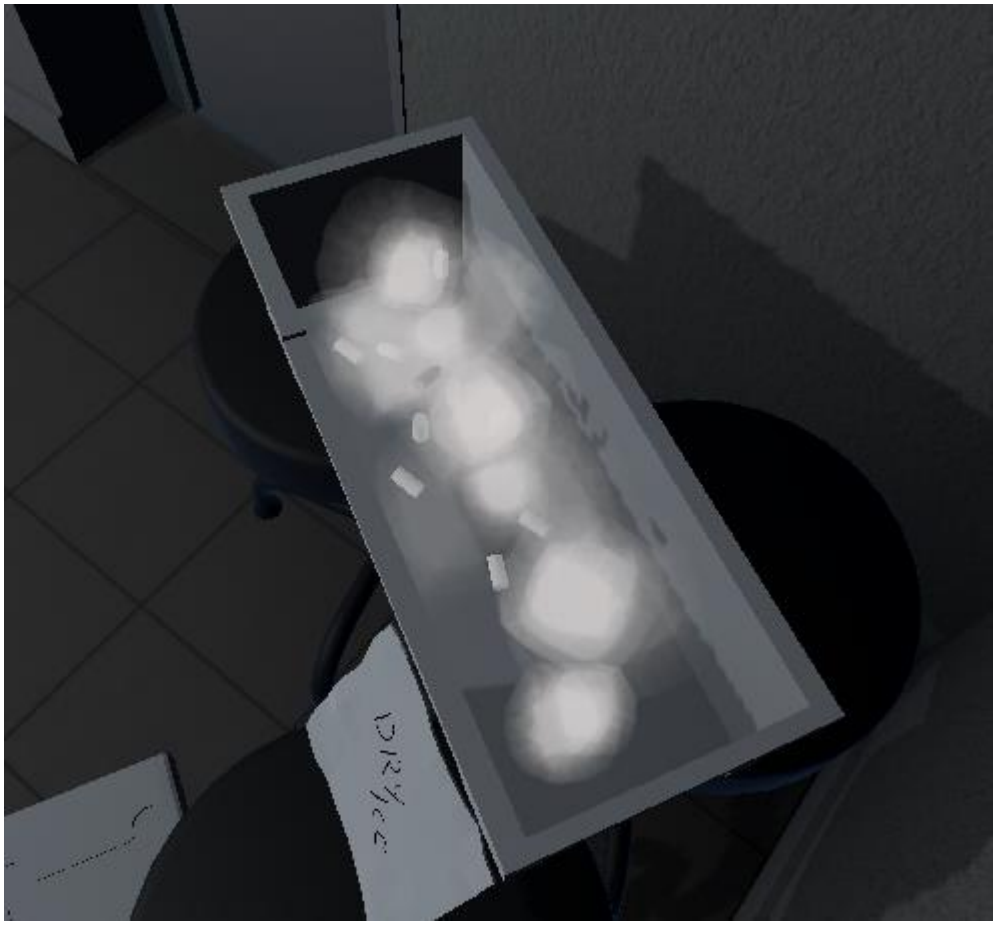

Figure 34 Thick smoke VFX

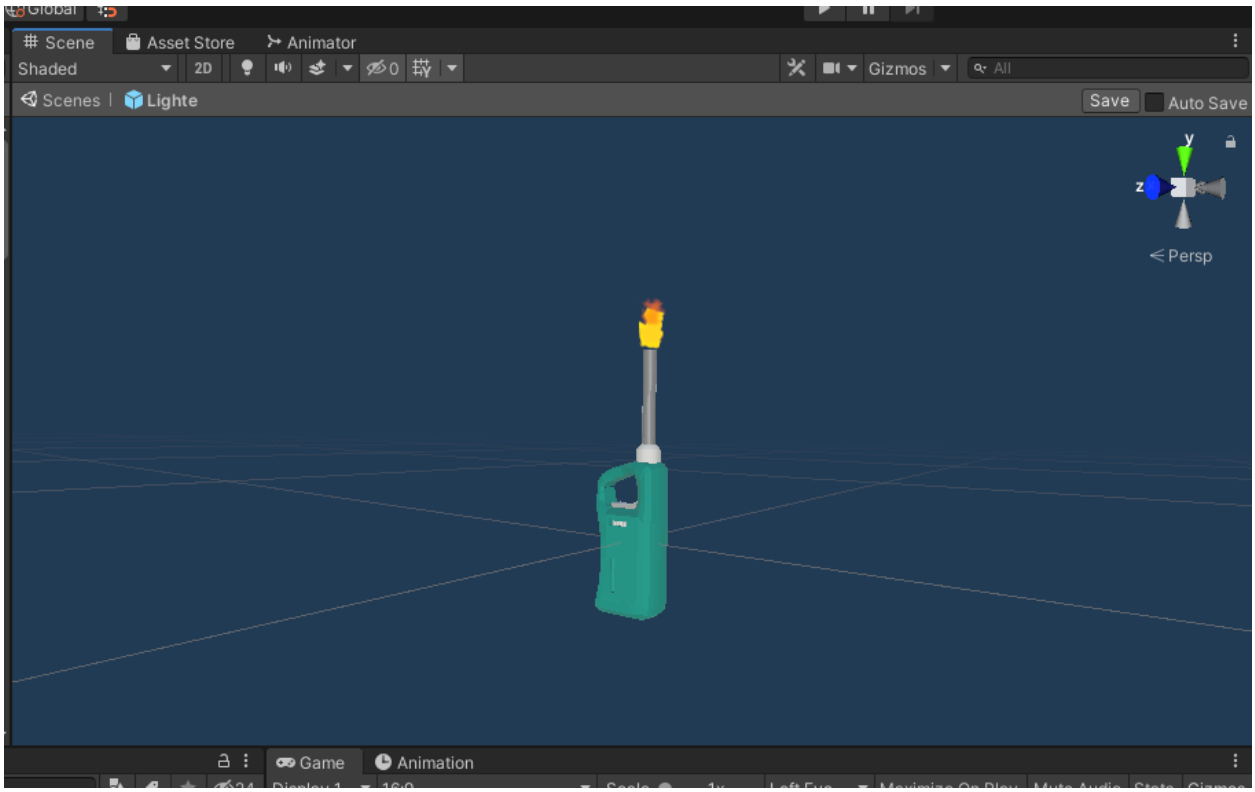

Figure 35 Lighter fire particles

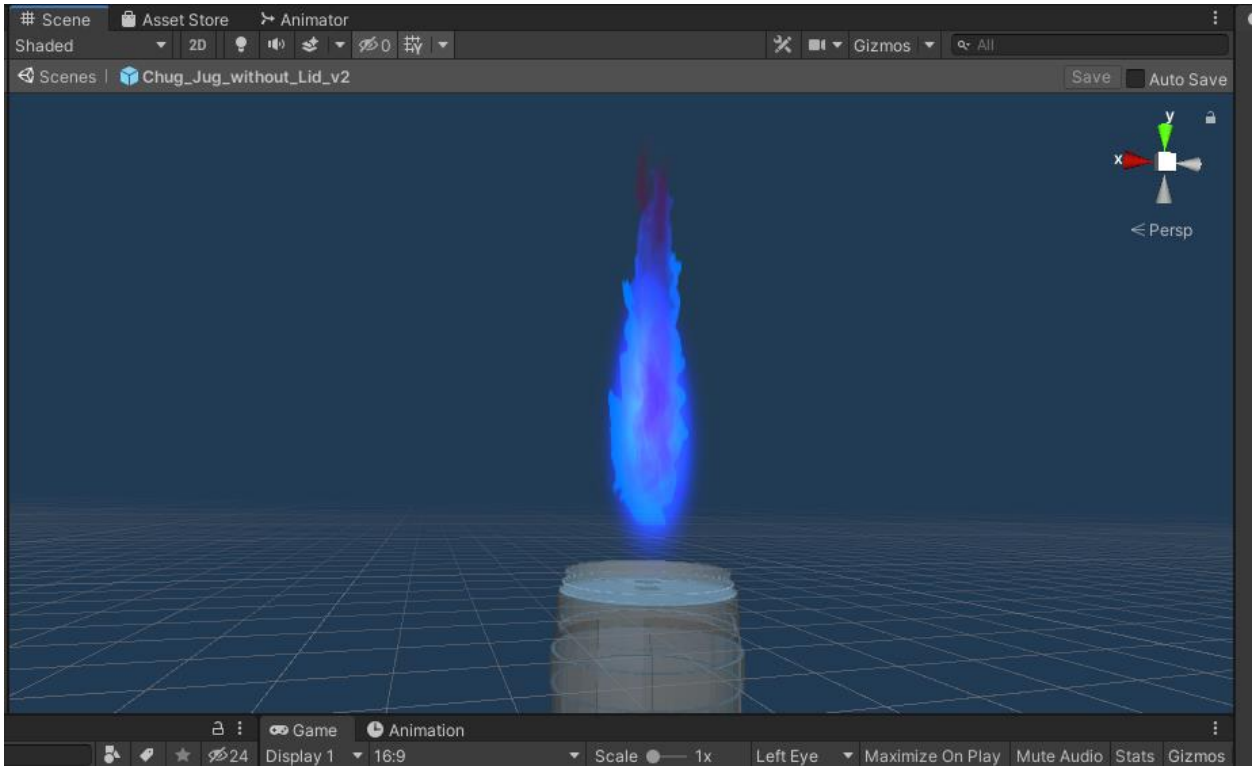

Figure 36 Blue fire particles

Task #14 Add a custom skybox (Airidas)

Added a custom skybox into the tutorial scene, to make the background more immersive.

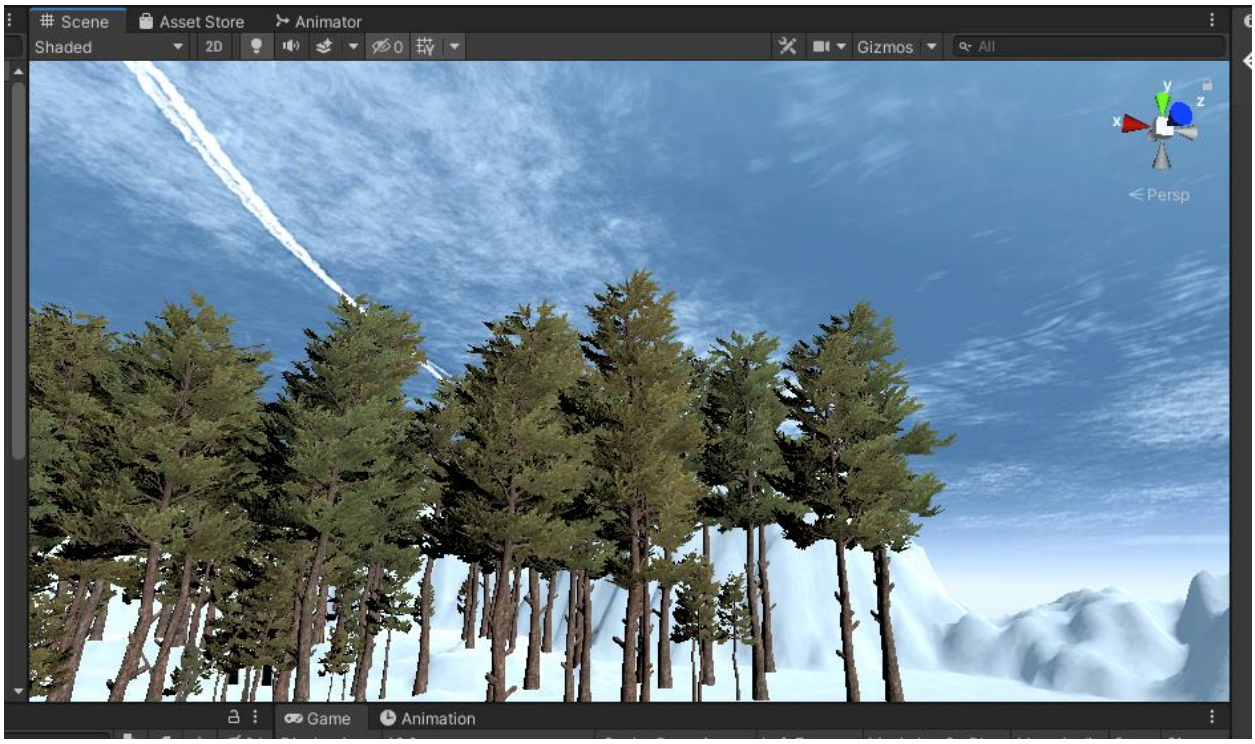

Figure 37 Custom skybox

Task #15 Add a custom skybox (Eligijus)

This project contains several skyboxes, this is one of them (main scene skybox).

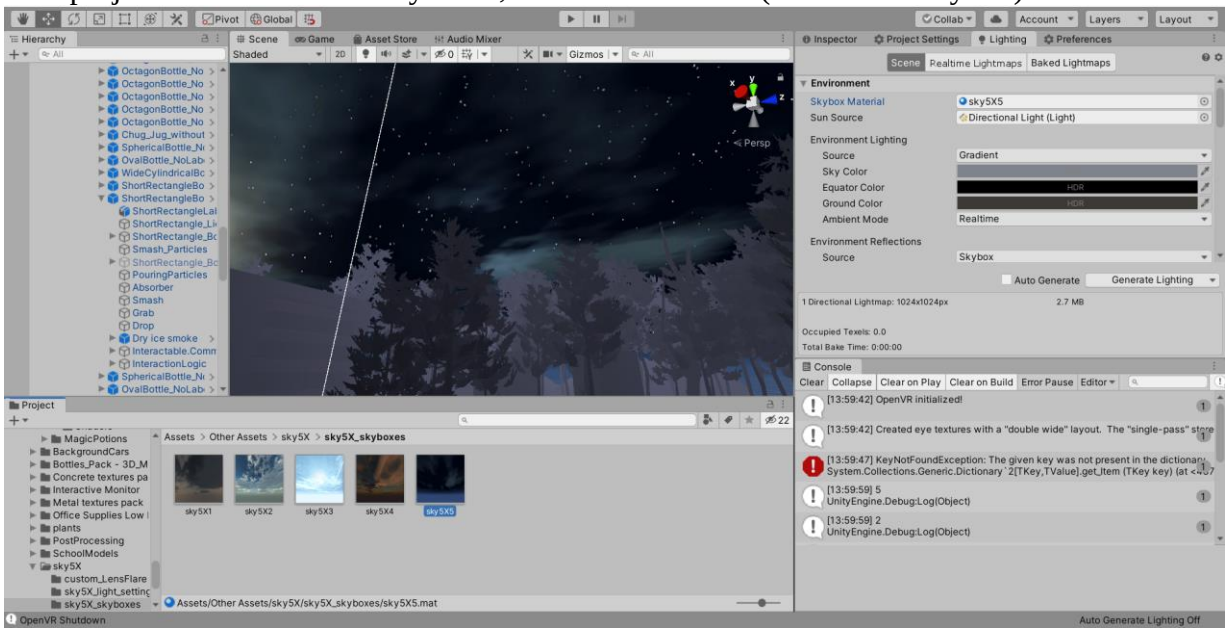

Figure 38 Editor woth open scene with night skybox

Task #16 Create at least 3 different Physics Materials for various parts of the map (either it's a slippery platform/ice, bouncy wall, non-slippery ground, etc) (Airidas)

Created three different physics materials to apply on to required Game Objects.

Figure 39 Slippery material

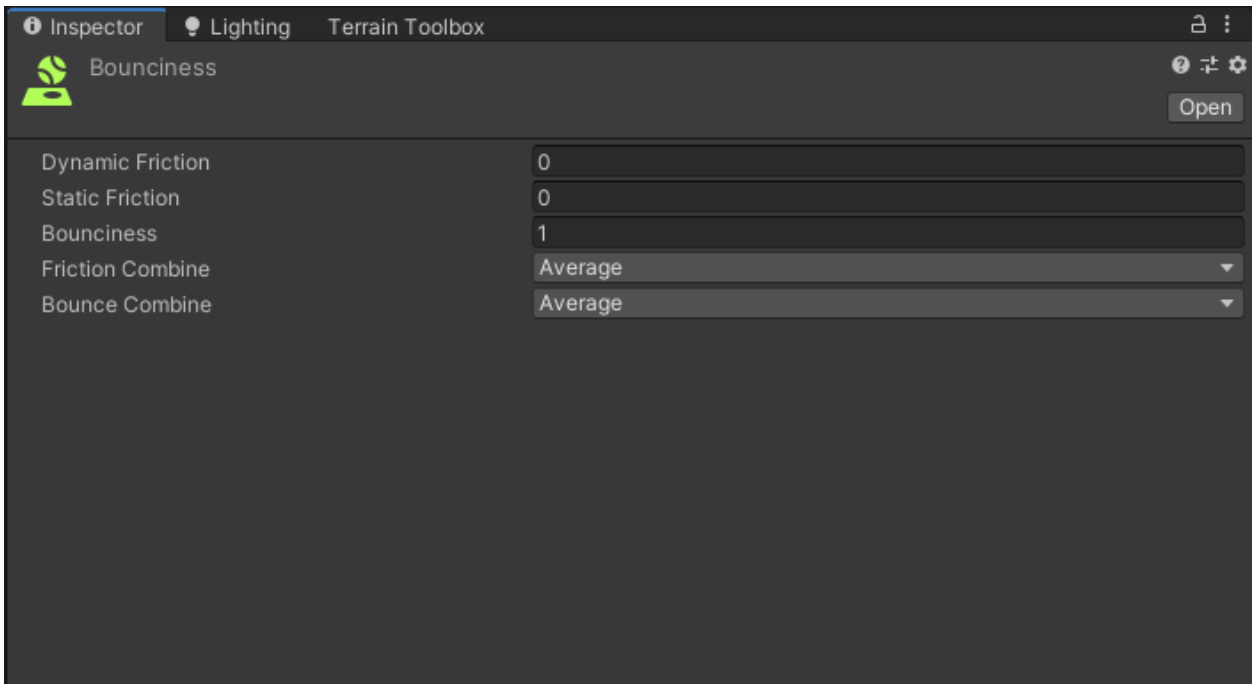

Figure 40 Bouncy material

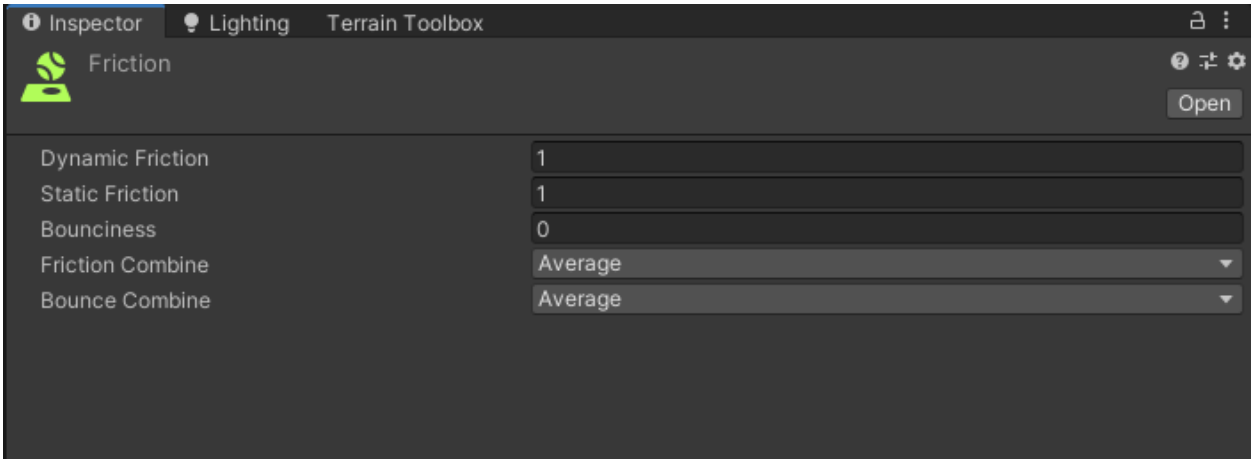

Figure 41 Friction material

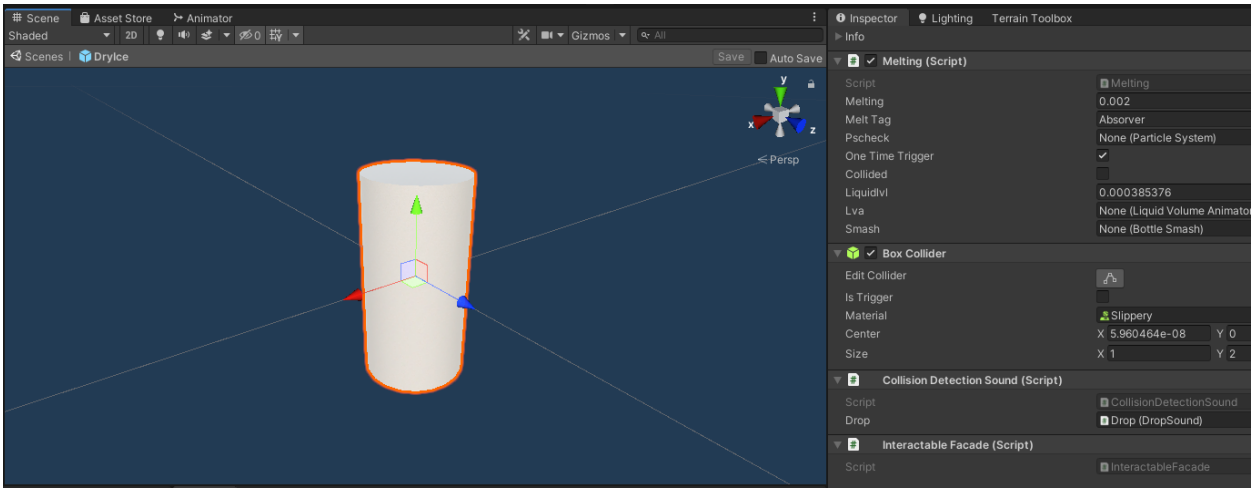

Figure 42 Applied slippery material

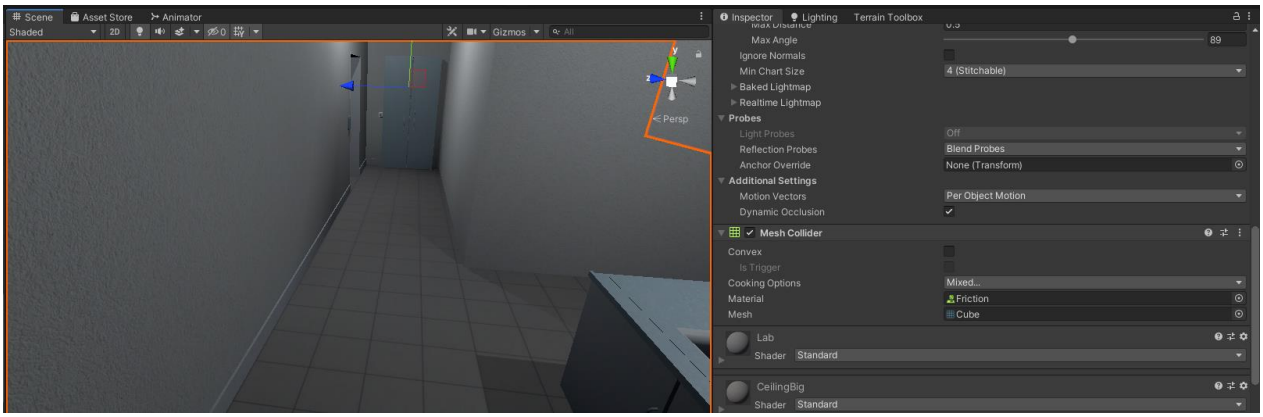

Figure 43 Applied friction material on the ground

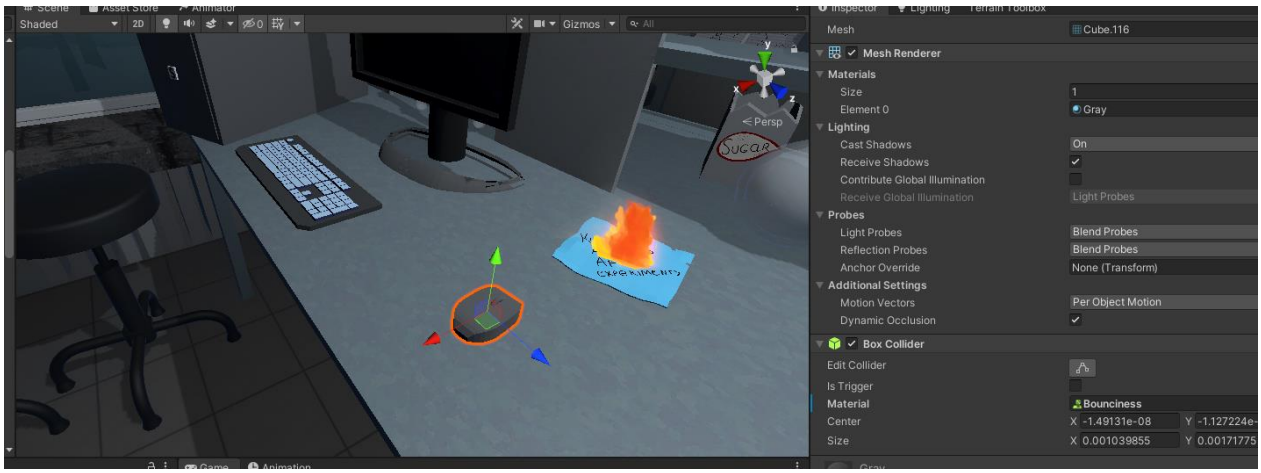

Figure 44 Applied bouncy material on the mouse

Task #17 Create at least 3 different Physics Materials for various parts of the map (either it's a slippery platform/ice, bouncy wall, non-slippery ground, etc) (Eligijus)

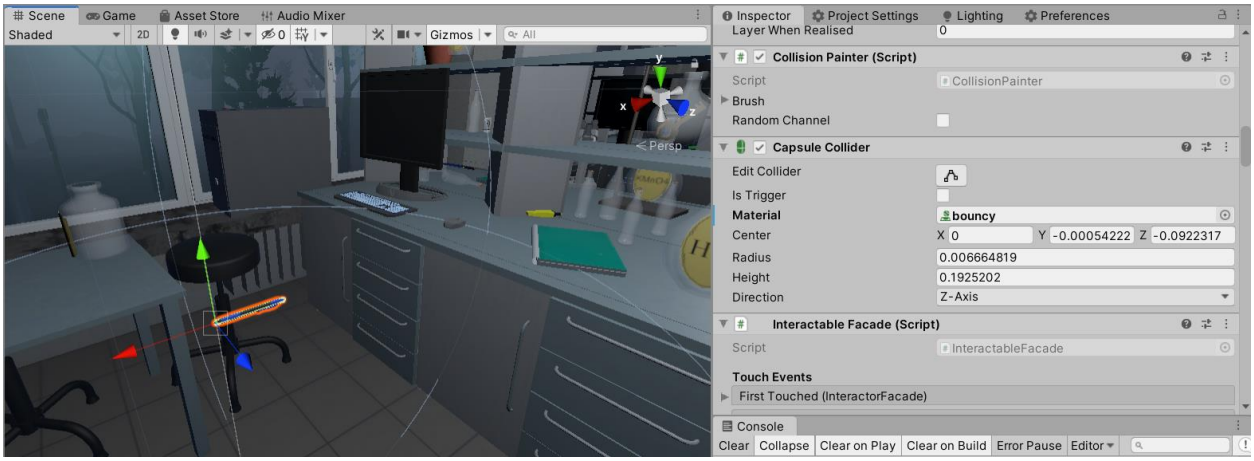

Figure 45 Bouncy pen

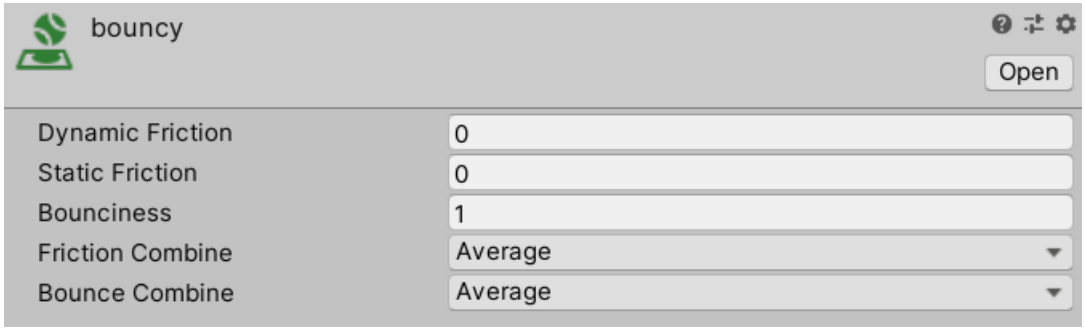

Figure 46 Bounciness material

Figure 47 Flask with friction

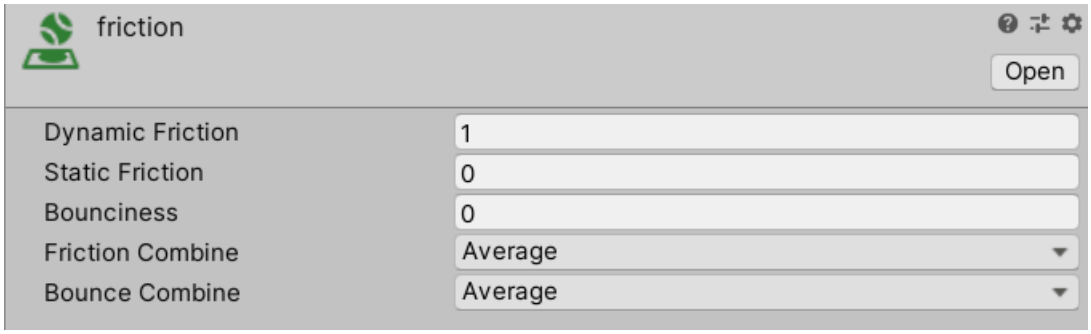

Figure 48 Friction material

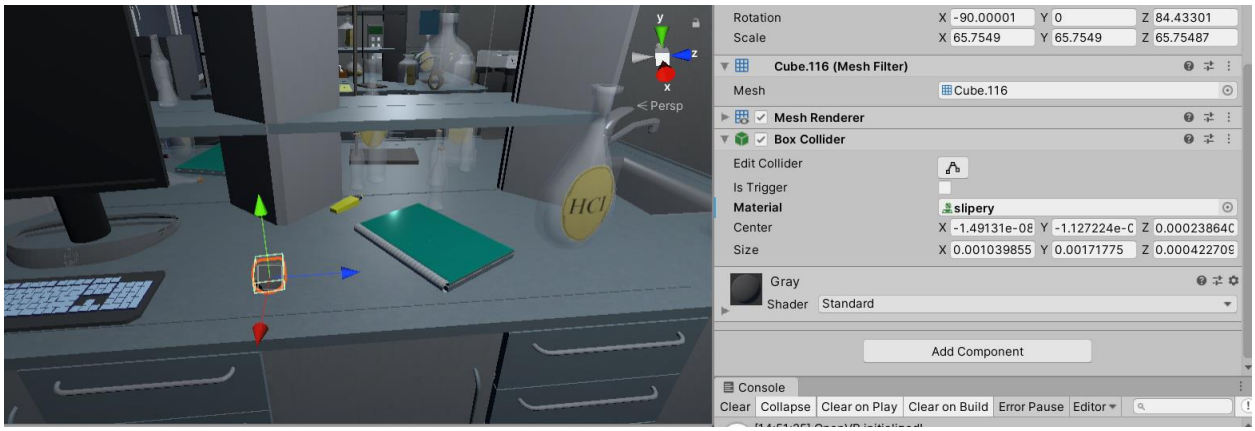

Figure 49 Slippery mouse

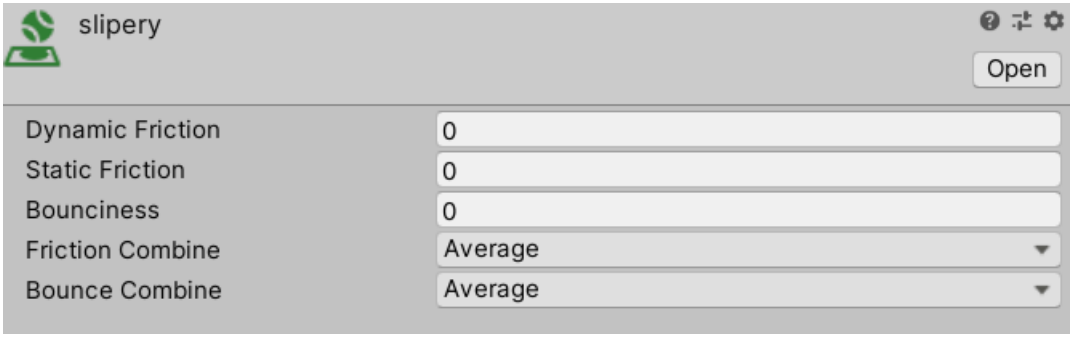

Figure 50 Slippery material

Task #18 Create 4 types of objects that use OnCollisionEnter / OnCollisionEnter2D / OnTriggerEnter / OnTriggerEnter2D: (Airidas)

Created four different Game Objects that uses these scripting methods:

- Flask breaks on collision with other GameObjects at a certain velocity
- Key plays a sound when it collides with any other GameObject
- Flask humanoid starts running away when player collides with a certain point in the scene
- Wall plug flashes sparks after colliding with any other GameObject

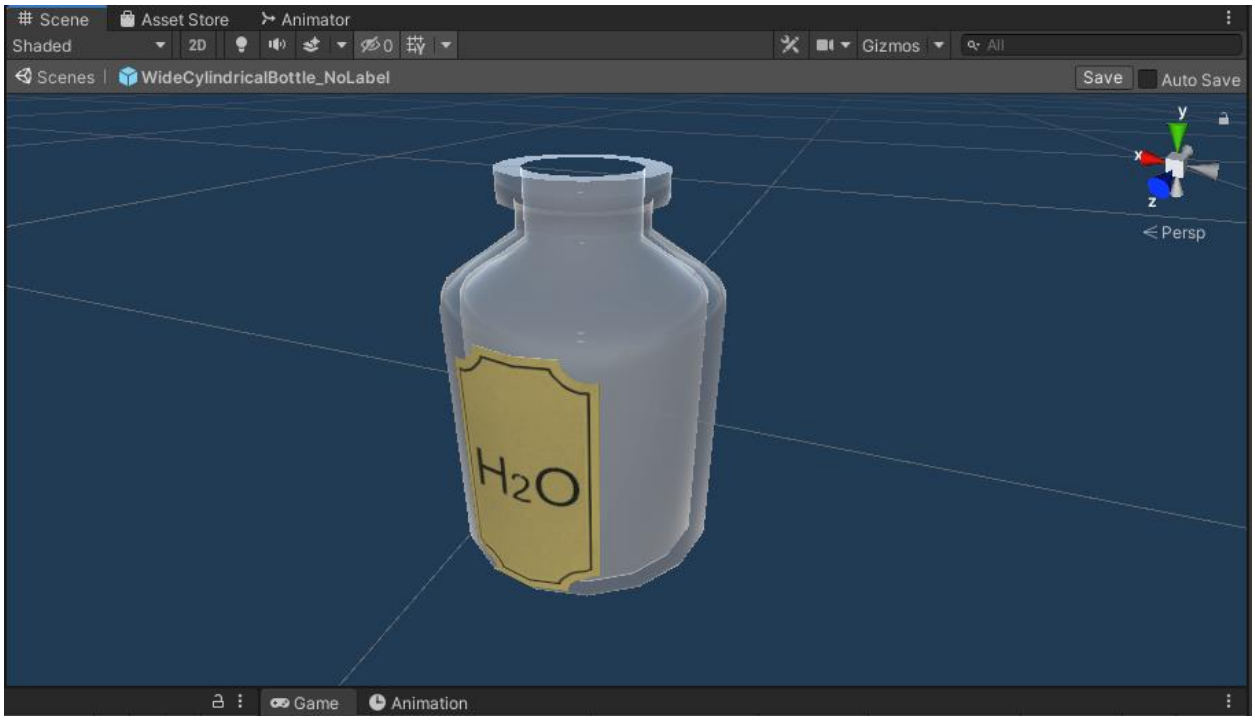

Figure 51 Flask

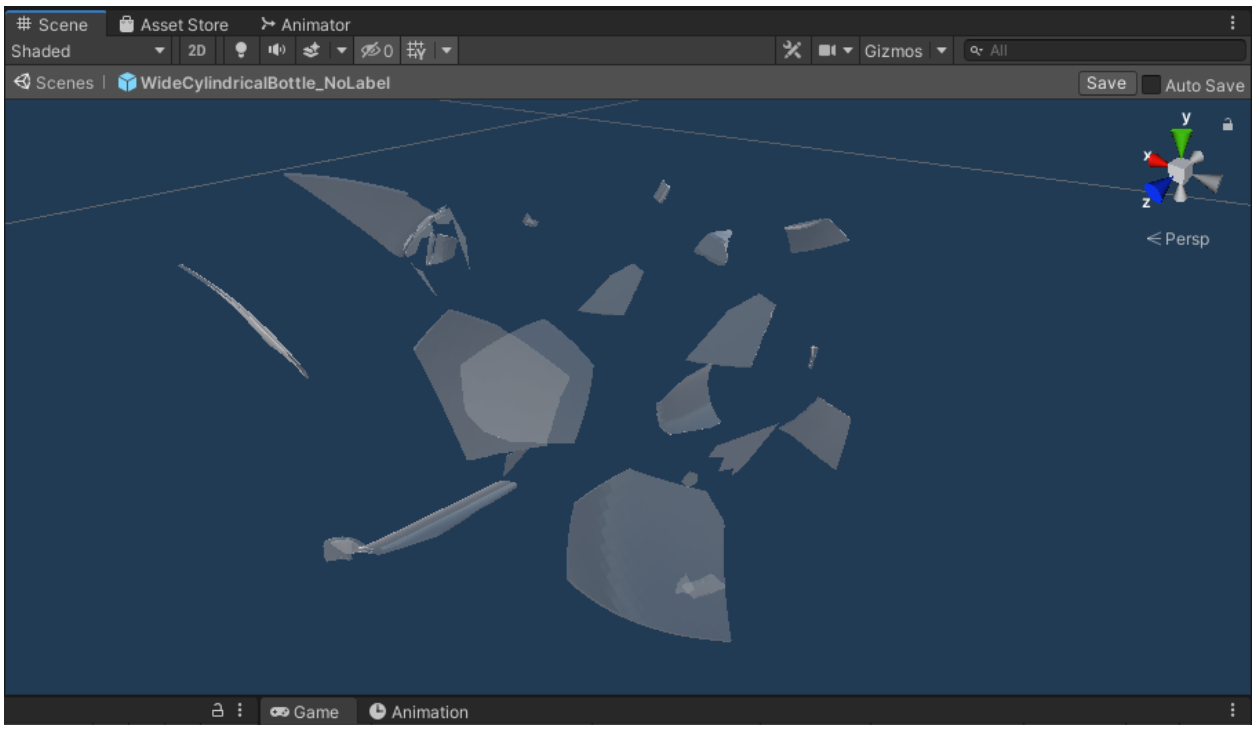

Figure 52 Shattered flask

```
    void OnCollisionEnter(Collision collision)
    {
        //set a timer for about 0.2s to be able to be broken
        _lastHitSpeed = collision.impulse.magnitude;
        if (collision.transform.tag != "Liquid" && collision.transform.tag !=
"Absorver")
        {
            //Debug.Log(collision.transform.name);
            collidedRecently = 0.2f;
        }

    }
```

Table 5 Flask shatter script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CollisionDetectionSound : MonoBehaviour
{
    [SerializeField] DropSound drop;
    // Start is called before the first frame update
    void OnCollisionEnter(Collision collision)
    {
        if (collision.relativeVelocity.magnitude > 1)
        {
            drop.PlaySound();
        }
    }
}
```

Table 6 Key collision with other objects sound script

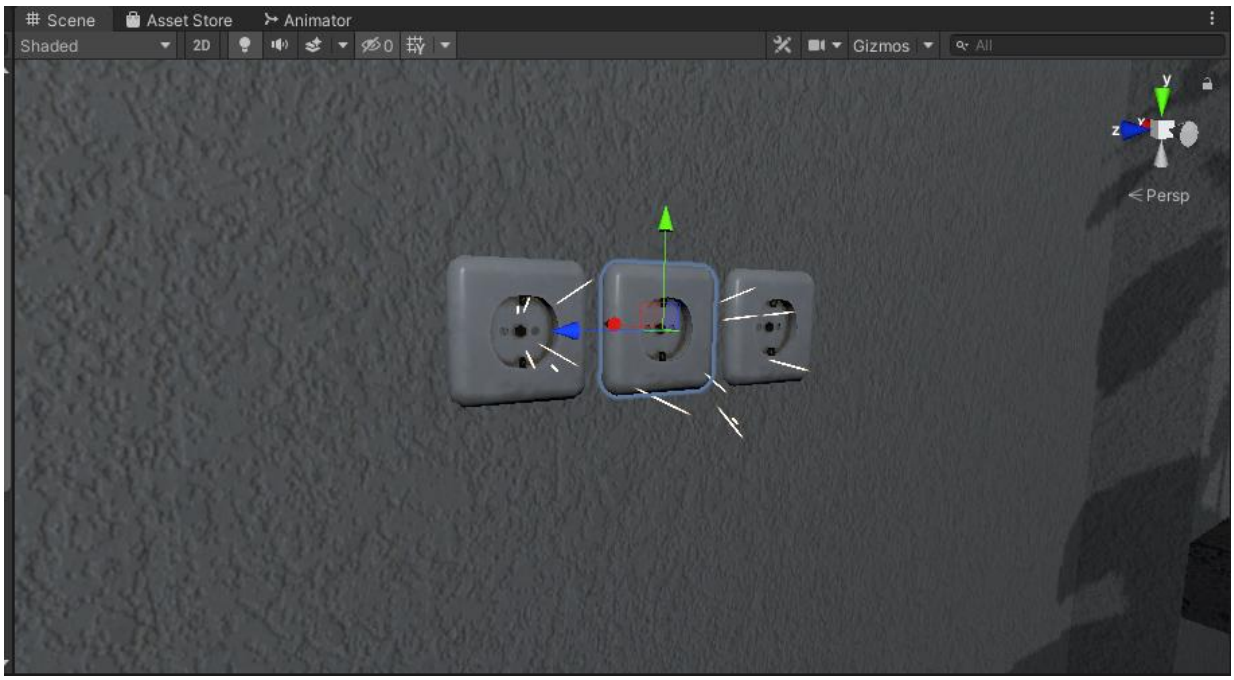

Figure 53 Wallplug particles

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag != "GameController" && other.name != 
"[VRTK][AUTOGEN][Controller][NearTouch][CollidersContainer]" && other != null)
        {
            particle.Play();
            aud.Play();
            spark = true;
            for (int i = 0; i < timeToSpark.Count; i++)
            {
                timeToSpark[i] = Random.Range(RandonTimeFromTo.x, 
RandonTimeFromTo.y);
            }
        }
    }
```

Table 7 Wallplug electricity particles activation script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RunningFlask : MonoBehaviour
{
    [SerializeField] Animator anim;
    bool animatorActive = false;

    void Start()
    {
        anim.enabled = false;
    }

    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.name == "HandAvatar" && !animatorActive)
        {
            anim.enabled = true;
        }
    }
}
```

Table 8 Flask humanoid activation script

Task #19 Create 4 types of objects that use OnCollisionEnter / OnCollisionEnter2D / OnTriggerEnter / OnTriggerEnter2D: (Eligijus)

This game fully utilizes on collision and on trigger enter systems. With these systems is used almost in all scritpts.

- Flask liquid absorber
  This script absorbs liquid and changes main liquid color

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
public class LiquidAbsorption : MonoBehaviour {

    //public int collisionCount = 0;
    [SerializeField] List<string> tagsIgnore = new string[] { "Fire", "DrySmoke"
}.ToList<string>();
    public Color currentColor;
    public BottleSmash smashScript;
    public LiquidLevel level;
    [SerializeField] ParticleColor particleColor;
    public float particleValue = 0.02f;
    //public LiquidVolumeAnimator LVA;

    // Use this for initialization
    void Start () {
        //if(LVA == null)
        //LVA = GetComponent<LiquidVolumeAnimator>();
       }
    void OnParticleCollision(GameObject other)
    {
        //check if it is the same factory.
        if (!tagsIgnore.Contains(other.tag))
        {
            if (other.transform.parent == transform.parent)
                return;
            bool available = false;
            if (smashScript.Cork == null)
            {
                available = true;
            }
            else
            {
                //if the cork is not on!
                if (!smashScript.Cork.activeSelf)
                {

                    available = true;
                }
                //or it is disabled (through kinamism)? is that even a word?
                else if (!smashScript.Cork.GetComponent<Rigidbody>().isKinematic)
                {
                    available = true;
                }
            }
            if (available)
            {
                currentColor = smashScript.color;
                if (level.level < 1.0f - particleValue)
                {
```

```
                    //essentially, take the ratio of the bottle that has liquid (0 to
1), then see how much the level will change, then interpolate the color based on the
dif.
                    Color impactColor;

                    if (other.GetComponentInParent<BottleSmash>() != null)
                    {
                        impactColor = other.GetComponentInParent<BottleSmash>().color;
                    }
                    else
                    {
                        impactColor =
other.GetComponent<ParticleSystem>().GetComponent<Renderer>().material.GetColor("_TintC
olor");
                    }

                    if (level.level <= float.Epsilon * 10)
                    {
                        currentColor = impactColor;
                    }
                    else
                    {
                        currentColor = Color.Lerp(currentColor, impactColor,
particleValue / level.level);
                    }
                    //collisionCount += 1;
                    level.level += particleValue;
                    smashScript.color = currentColor;
                }
            }
        }

    }
       // Update is called once per frame
       void Update ()
       {
         smashScript.ChangedColor();
         currentColor = smashScript.color;
         particleColor.Unify();
    }
}
```

Table 9 Liquid absorber code

- Note trigger
  Entering on note trigger shows note number on board

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using VRTK;

public class NoteBoard : MonoBehaviour
{
    [SerializeField] GameObject printKey;
    bool isGrabbed = false;
    TextMeshPro print;
    TextMeshPro key;
    int count = 0;
       [SerializeField] DelegateChange Task;
    // Start is called before the first frame update
    void Start()
    {
        print = printKey.GetComponent<TextMeshPro>();
```

```
        key = GetComponentInChildren<TextMeshPro>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {

        if (isGrabbed)
        {
            print.text = key.text;
                    Task.AddTask();
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        isGrabbed = true;
    }

    private void OnTriggerEnter(Collider other)
    {
        isGrabbed = false;
    }

}
```

Table 10 On trigger enter note

- Start timer when player enter on trigger

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StartTime : MonoBehaviour
{
    TimeLeft time;
    int count = 0;

    void Start()
    {
        time = GetComponentInParent<TimeLeft>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.name == "Head" && count <= 2)
        {
            time.TimerStart();
            count++;
        }
        //else if (other.name ==
"[VRTK][AUTOGEN][Controller][NearTouch][CollidersContainer]" && count >= 3)
        //{

        //}
    }
}
```

Table 11 OnTrigger enter timer

- Doors trigger
  When player enters on trigger door closes.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using VRTK;
using VRTK.Prefabs.Interactions.Controllables;
using VRTK.Prefabs.Interactions.Controllables.ComponentTags;
public class CloseDoors : MonoBehaviour
{
    [SerializeField] GameObject doorController;
    [SerializeField] float doorSpeed = 0.05f;
    [SerializeField] GroundControll controll;
    public Rigidbody rb;
    public bool isClosing = false;
    bool isGrabbed = false;
    //public RotationalJointDrive rotator;
    public GameObject rotator;
    int count = 0;
    // Start is called before the first frame update
    void Start()
    {
        //rotator = doorController.GetComponent<RotationalDriveFacade>();
        //rb = doorController.GetComponent<Rigidbody>();
    }

    private void Update()
    {
        //Debug.Log(doorController.transform.localRotation.eulerAngles.y);
        if (doorController.transform.localRotation.eulerAngles.y < 0.002 &&
doorController.transform.localRotation.eulerAngles.y > 0 && isClosing)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationY;
            Destroy(this);
            count++;
        }
        else if ((doorController.transform.localRotation.eulerAngles.y > 350 ||
doorController.transform.localRotation.eulerAngles.y < 0.002) && isClosing)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationY;
            Destroy(this);
            count++;
        }
    }

    public void Grabb()
    {
        isGrabbed = true;
    }
    public void UnGrabb()
    {
        isGrabbed = false;
    }

    private void OnTriggerEnter(Collider other)
    {

        if (other.name == "Head" && count < 1 &&
doorController.transform.rotation.eulerAngles.y > 1)
        {
            isClosing = true;
            ForceToClose();
            Lock();
            count++;
```

```
        }
    }

    void ForceToClose()
    {
        rb.AddForce(transform.right * doorSpeed);
    }

    void Lock()
    {
        //controll.coridorOff();
        rotator.SetActive(false);
    }
}
```

Table 12 Doors on trigger enter

Task #20 Assign optimal colliders for the environment objects that are not moving and set their flags to static (Airidas and Eligijus)

The optimal colliders for static objects and background elements were assigned at the beginning of the scene development. To optimize the performance even further, we have added Occlusion culling so that the objects, which are hidden or just simply cannot be seen are not being rendered.

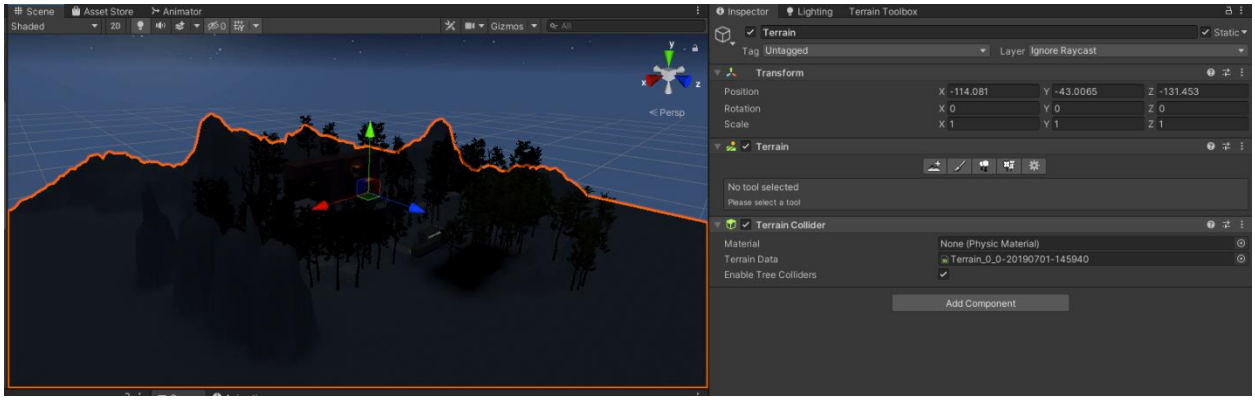

Figure 54 Terrain static flag

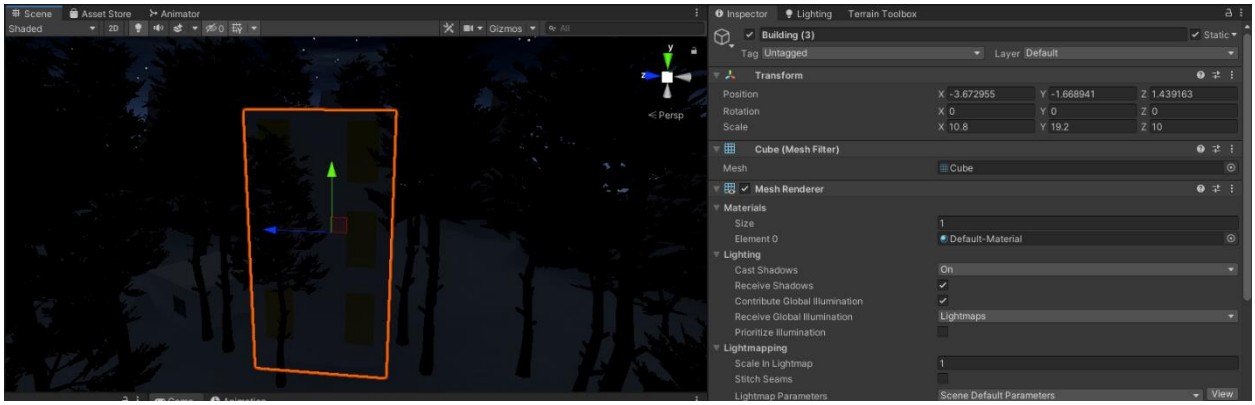

Figure 55 Background building static flag

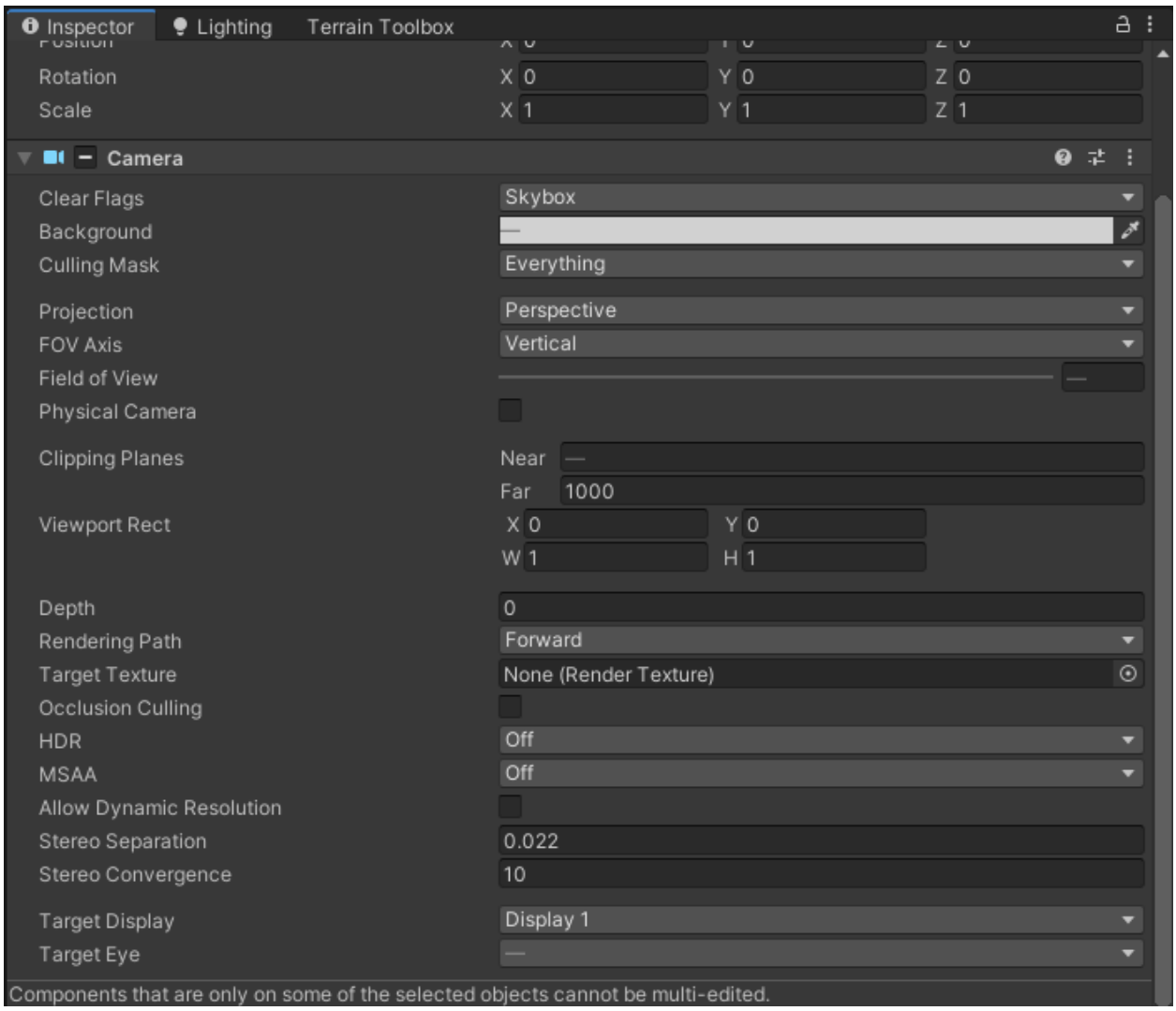

Figure 56 VR camera settings

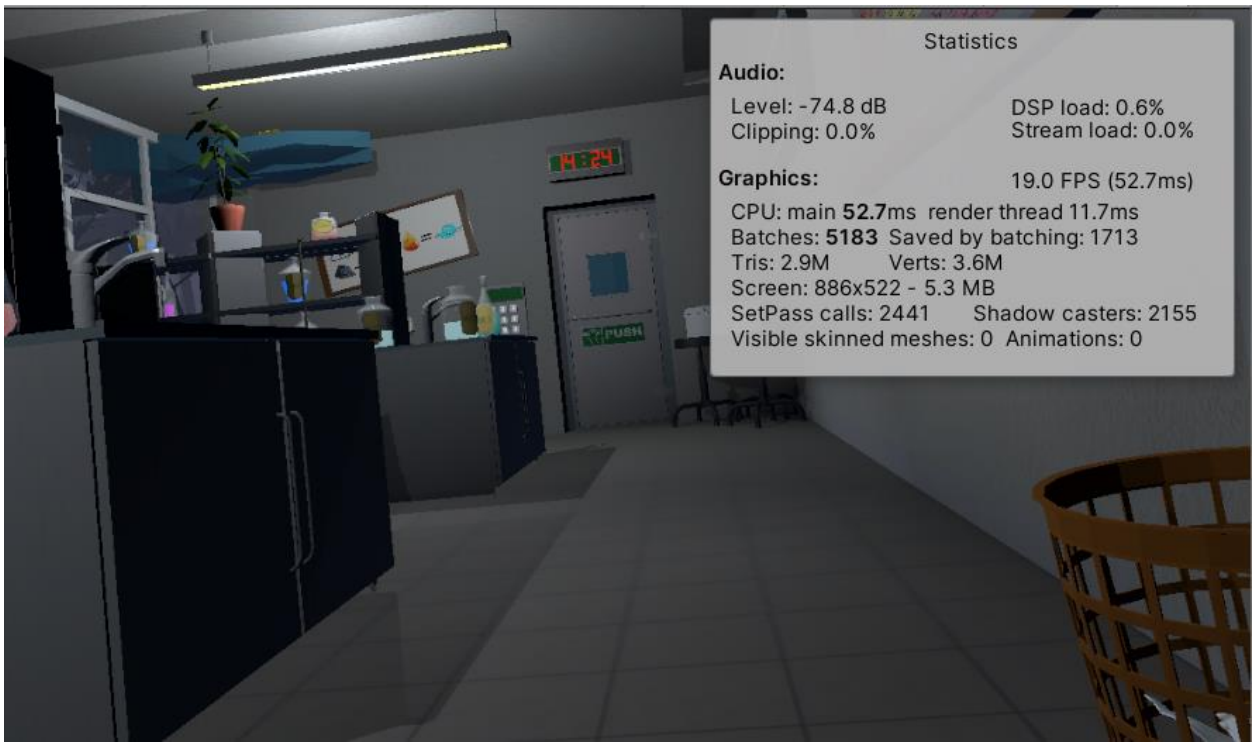

Figure 57 Performance statistics

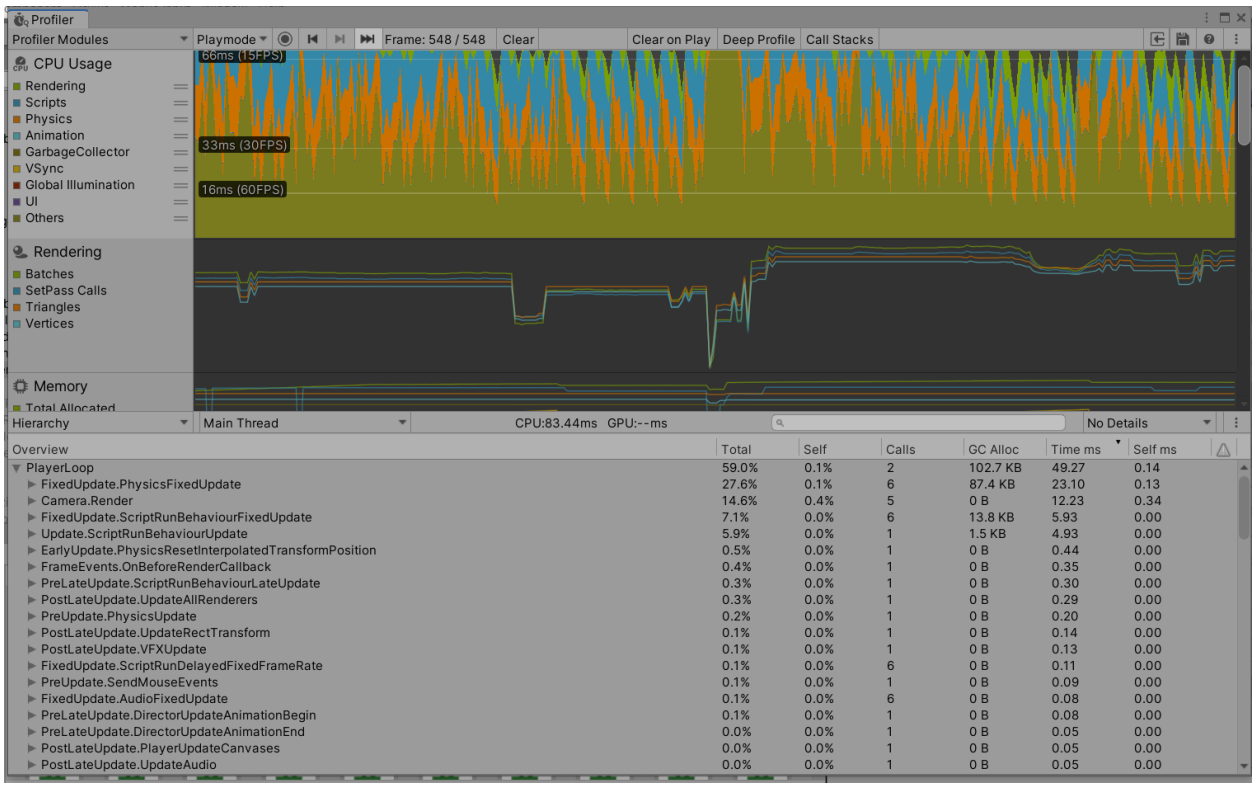

Figure 58 Profile performance statistics

Task #21 Bake a lightmap and measure performance (Airidas and Eligijus)

The baked lightmap is almost fully black, because it was baked with most of the lights turned off in scene. Most of these lights have various effects, such as turning them off and on. That is why baking the lightmap this way was the optimal variant.

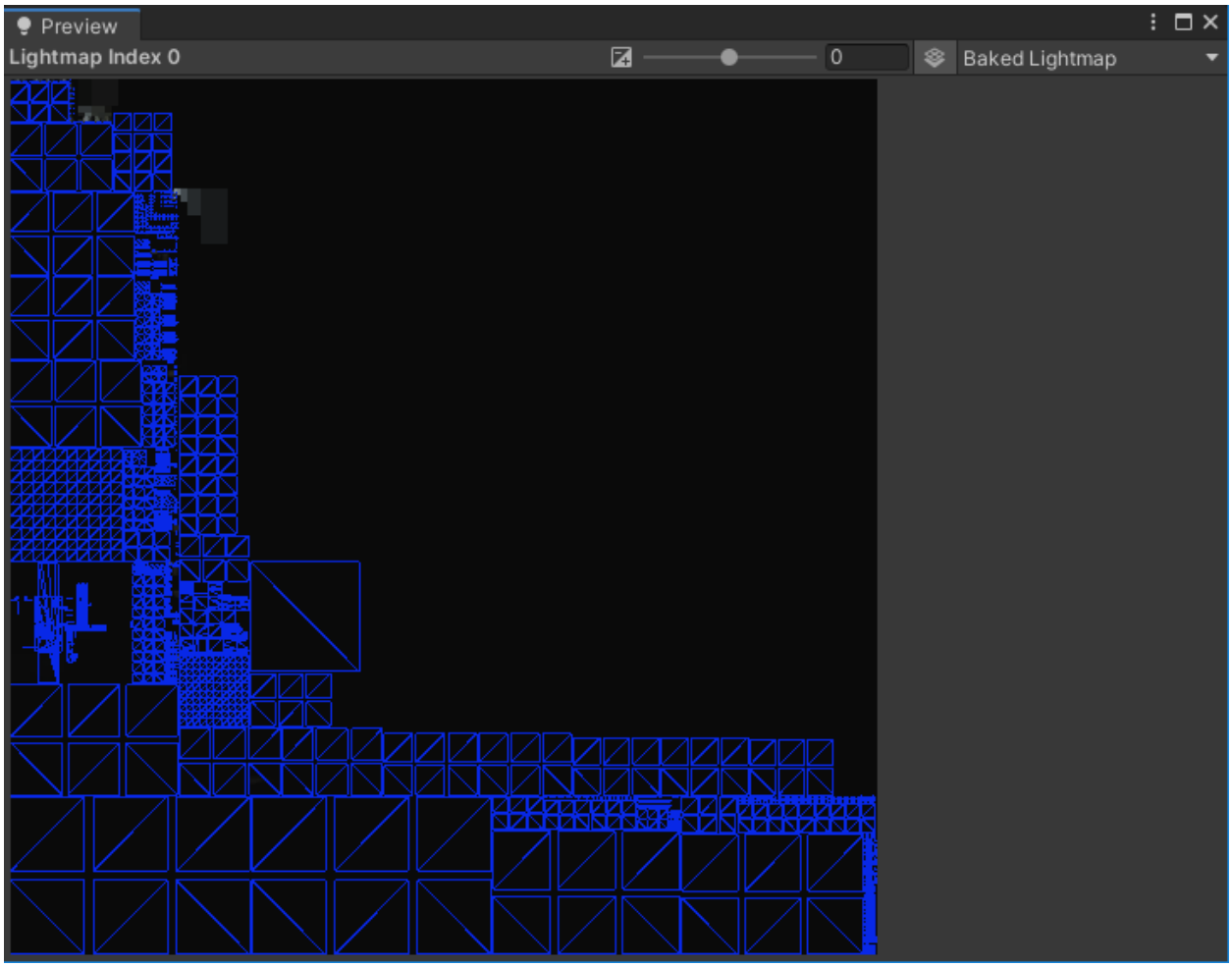

Figure 59 Main scene lightmap

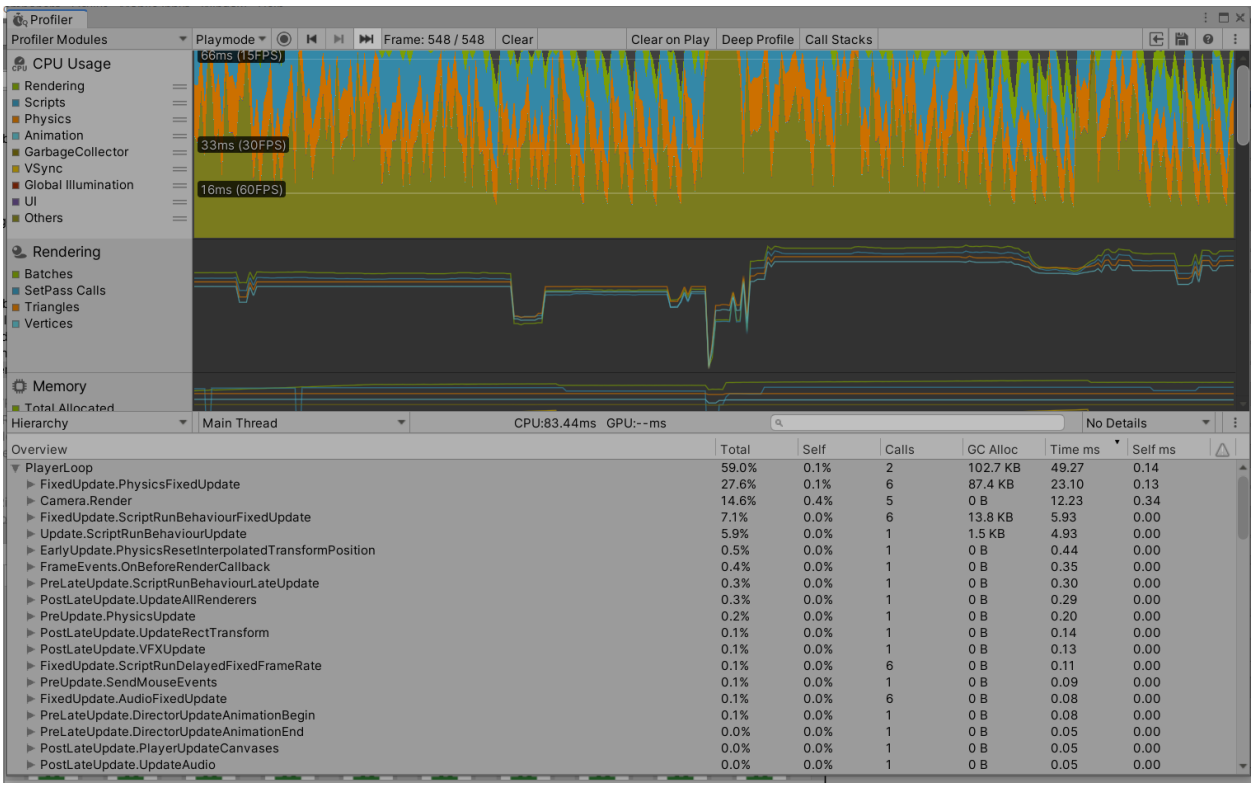

Figure 60 Main scene profiler statistics

Task #22 Optimize all textures depending on their parameters and measure graphical memory load (Airidas and Eligijus)

Optimized the textures by adjusting parameters such as Normal Maps, Mip maps, etc.

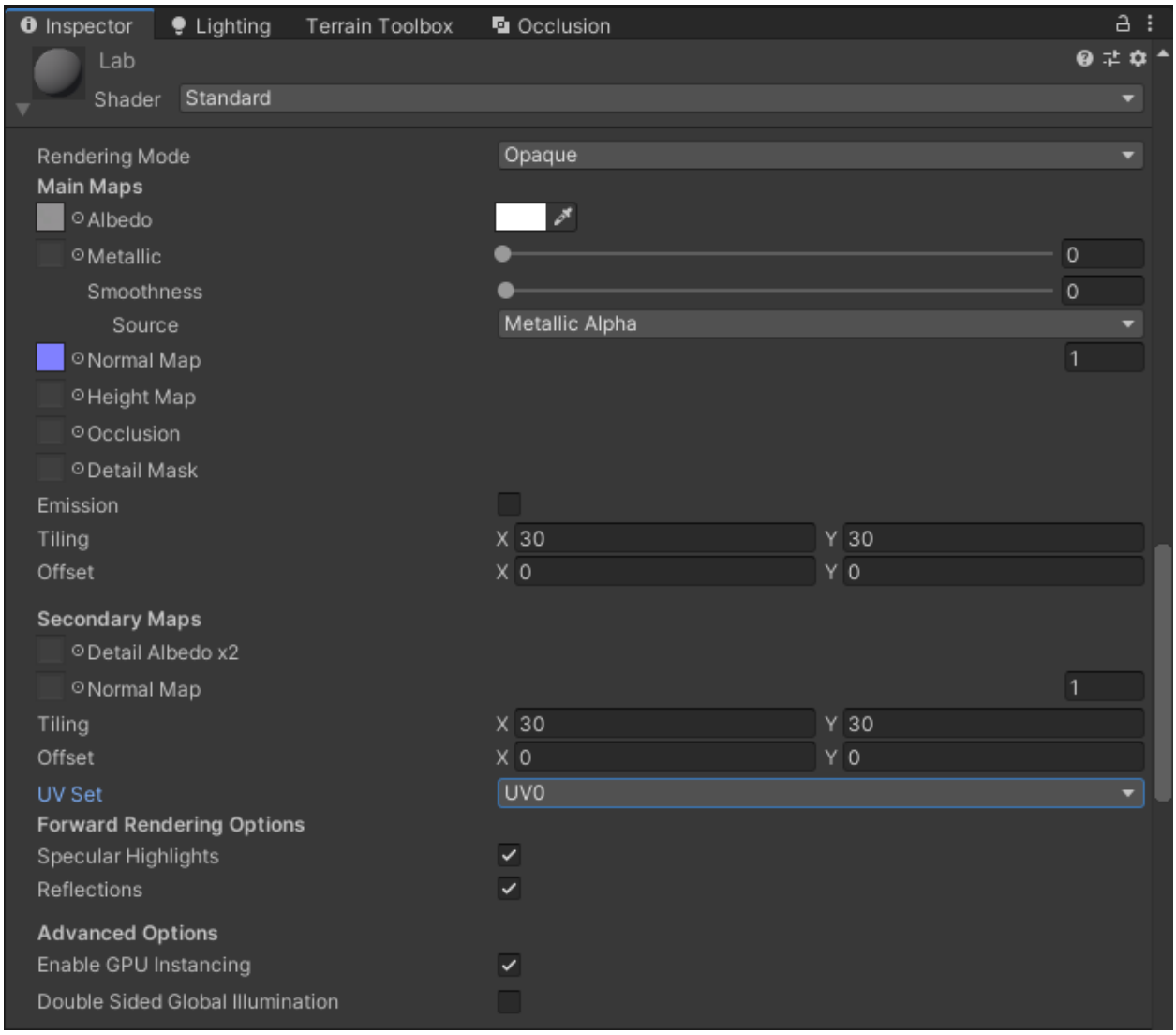

Figure 61 Material settings

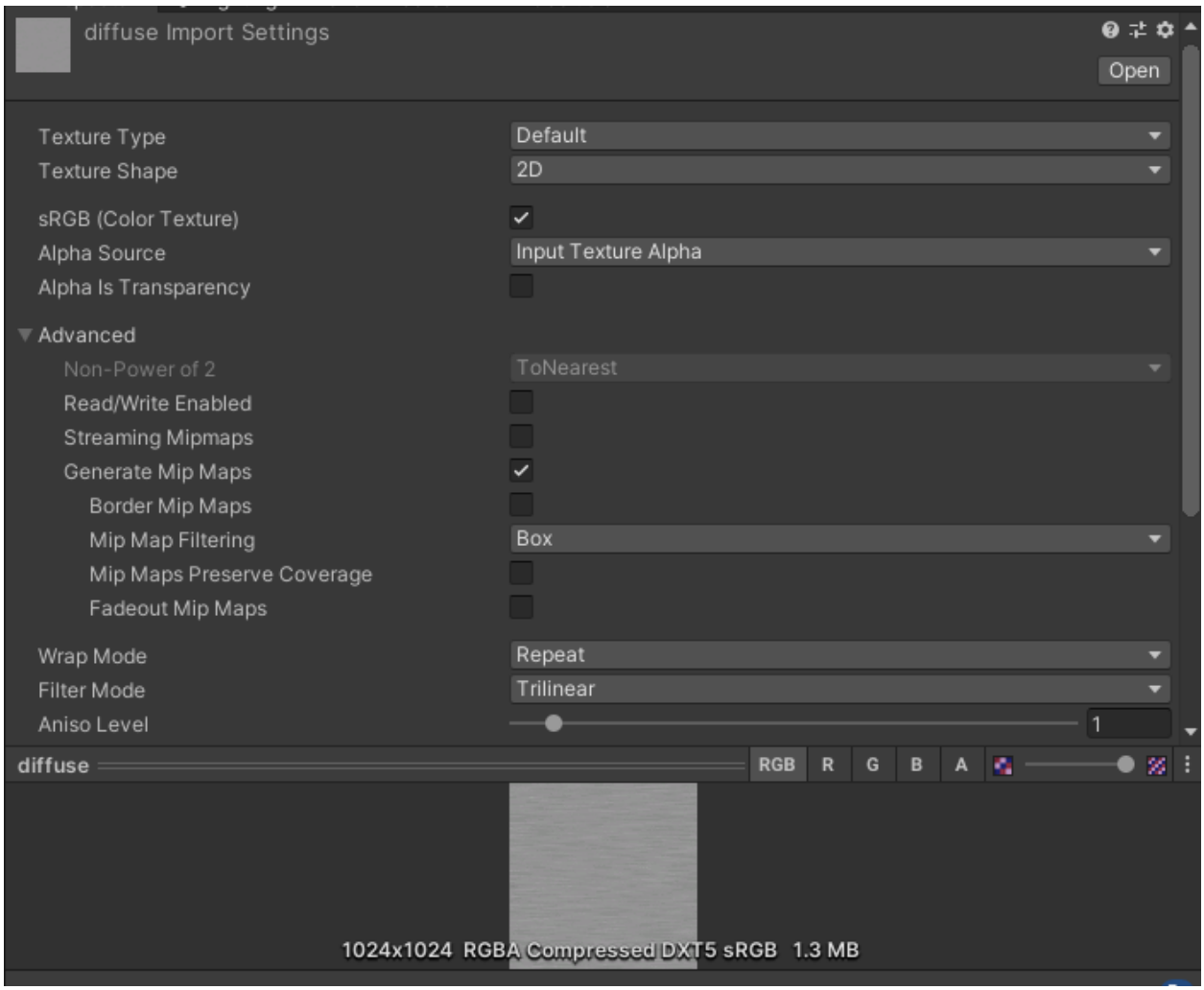

Figure 62 Texture settings

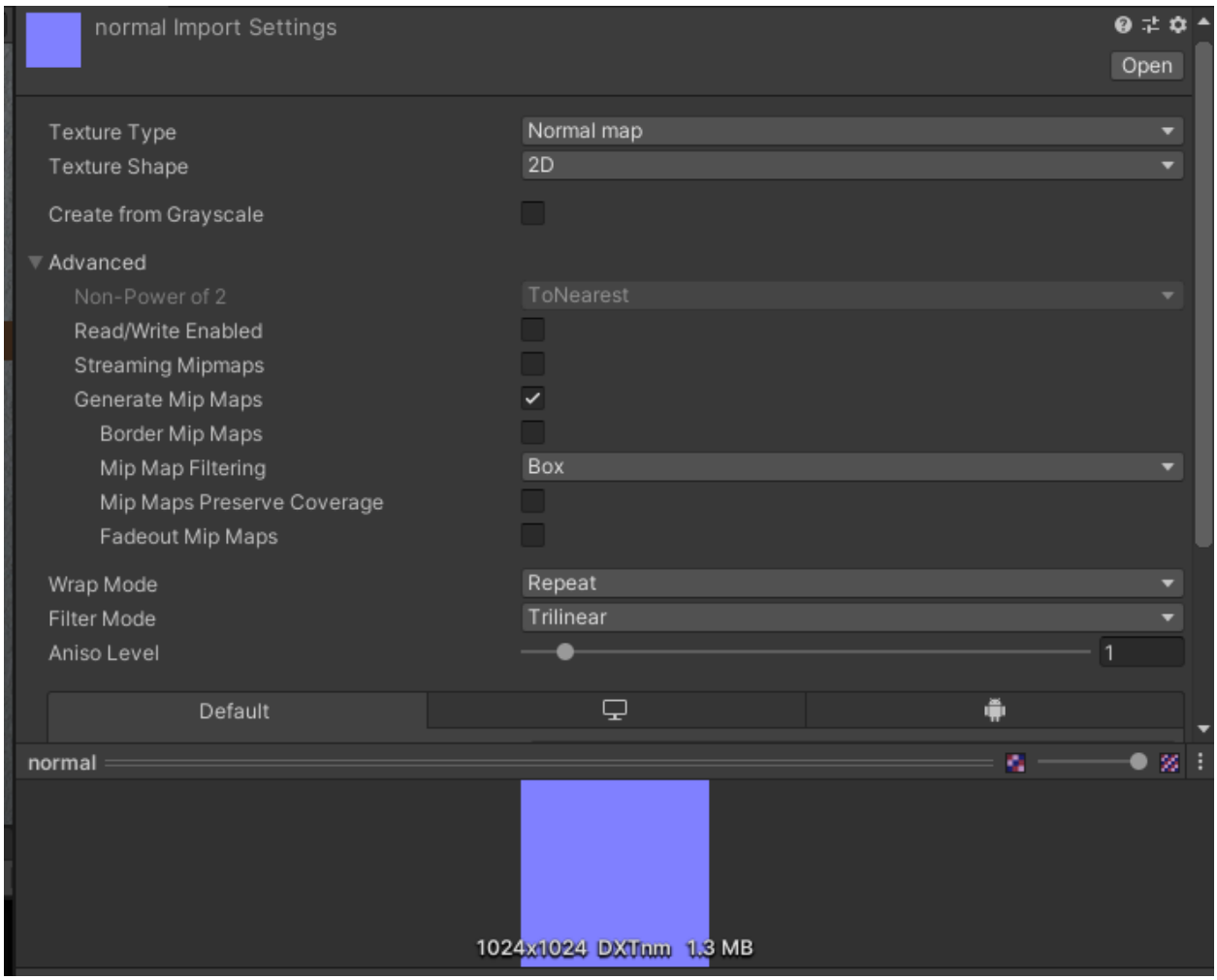

Figure 63 Normal map settings

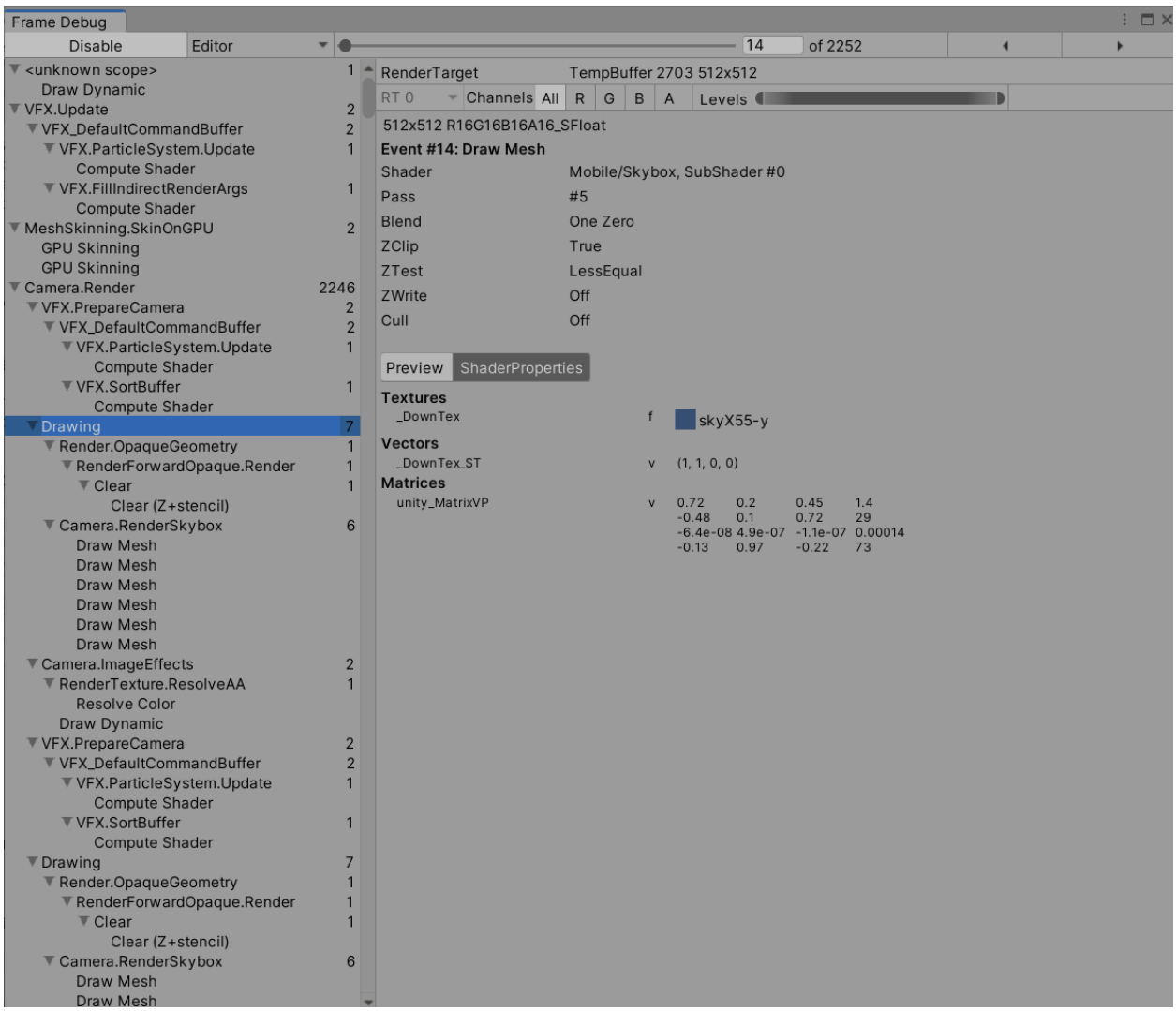

Figure 64 Graphical load measurment

Task #23 Try hard vs soft shadows, different quality settings and measure the performance (Airidas and Eligijus)

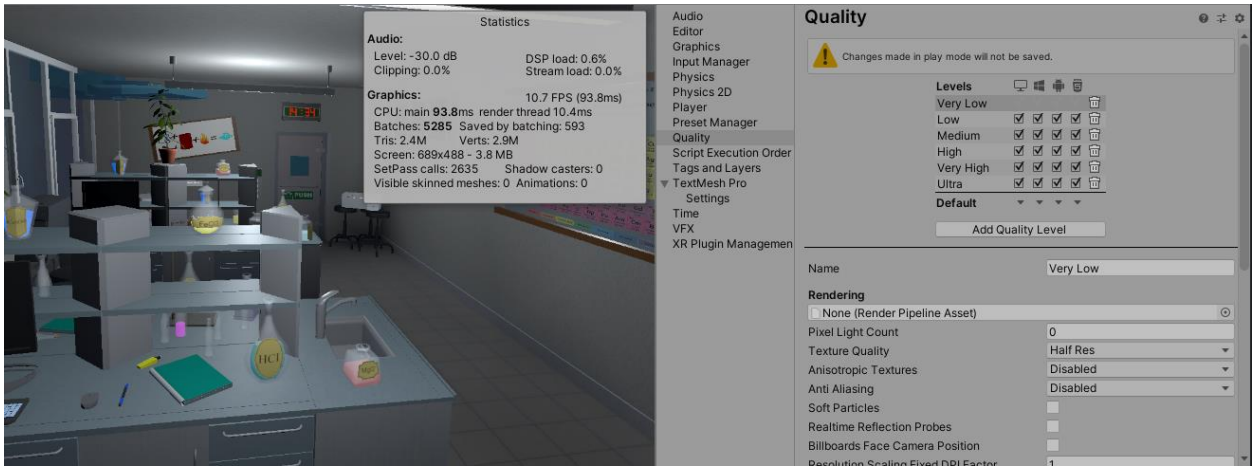

Figure 65 Low settings

Figure 66 High settings

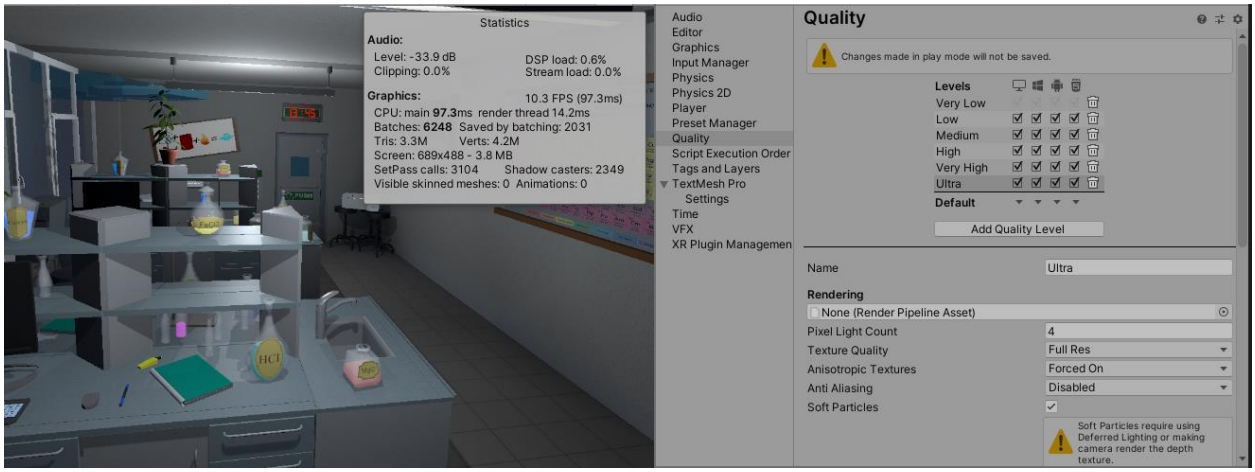

Figure 67 Ultra settings

Task #24. Add a MENU system to your game (New Game, Choose Level [If Applies], Options, About, Exit (Eligijus and Airidas)

Both Main Menu and Pause Menu systems are implemented into the game. When the player launches the game, he starts at Main Menu scene and has the possibilities to Start the main level, join a tutorial room, check out his settings and game controls, or just leave the game.

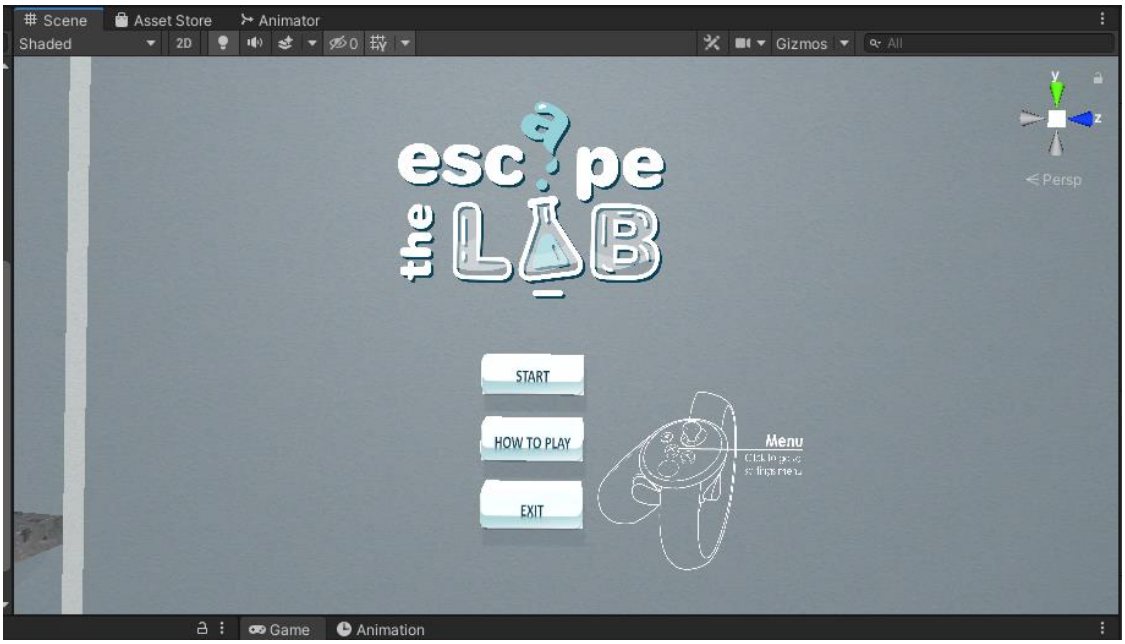

Figure 68 Main menu scene

If player enters Pause Menu during the playthrough, he can go to the same Menu pages that were mentioned in the Main Menu, Restart the level or go back to Main Menu.

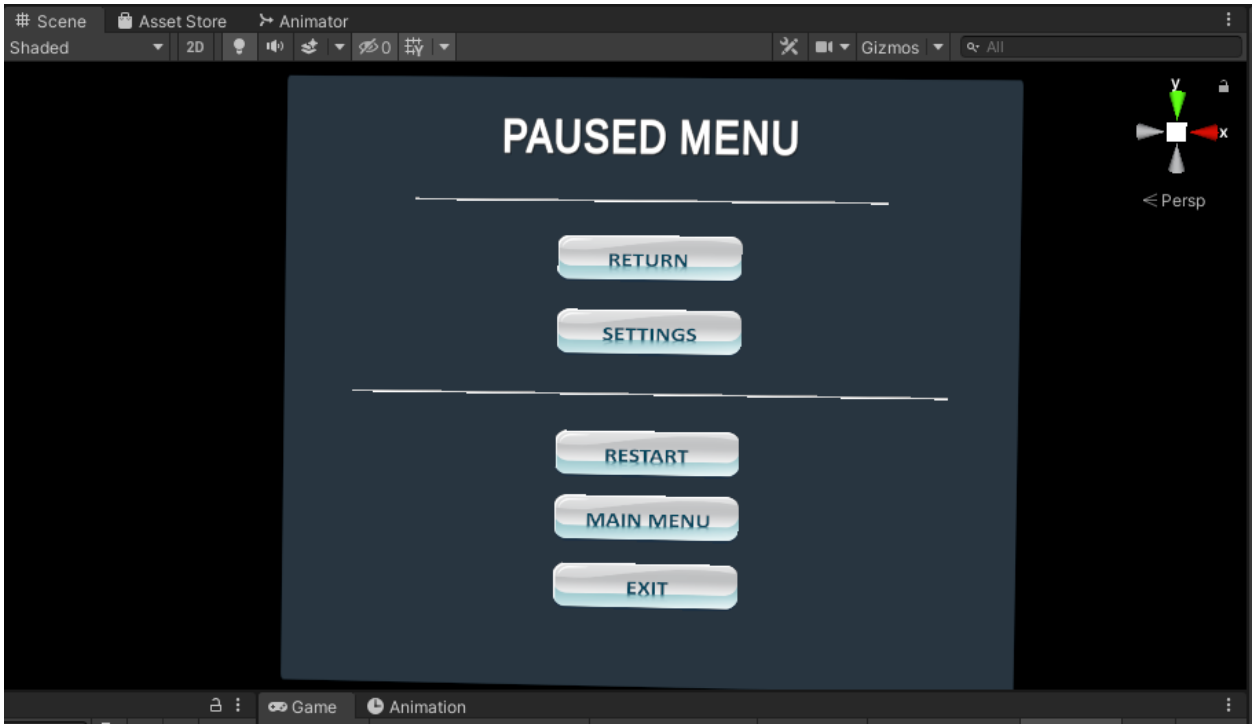

Figure 69 Paused menu

```
public class InteractableCollisionUI : MonoBehaviour
{
    [SerializeField] float sliderOffset = 0.5f;
    [SerializeField] float thicknes = 0.01f;
    [SerializeField] bool buttonPressWithTrigger = false;
    [SerializeField] GameObject triggerGo;
    BoxCollider colliderForPosition;
    List<UiButtonClick> buttons;
    List<UiSliderControll> sliders;
    List<UiDropdownControll> dropdowns;
    RectTransform uiTransform;
    static float SliderOffset = 0;
    void Start()
    {
        if (triggerGo != null)
        {
            if (buttonPressWithTrigger)
            {
                triggerGo.SetActive(true);
            }
            else
            {
                triggerGo.SetActive(false);
            }
        }
        buttons = new List<UiButtonClick>();
        sliders = new List<UiSliderControll>();
        dropdowns = new List<UiDropdownControll>();
        gameObject.AddComponent<Rigidbody>();
        gameObject.AddComponent<BoxCollider>();
        uiTransform = GetComponent<RectTransform>();
        colliderForPosition = gameObject.GetComponent<BoxCollider>();
        colliderForPosition.size = new Vector3(uiTransform.sizeDelta.x,
uiTransform.sizeDelta.y, thicknes);
        colliderForPosition.isTrigger = true;
        Rigidbody rb = GetComponent<Rigidbody>();
        rb.isKinematic = true;
        RecursiveChilds(gameObject.transform);
        SliderOffset = sliderOffset;
    }
    void RecursiveChilds(Transform transformGo)
    {
        for (int i = 0; i < transformGo.childCount; i++)
        {
            if (transformGo.GetChild(i).GetComponent<Button>() != null)
            {
                transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                UiButtonClick btn =
transformGo.GetChild(i).gameObject.AddComponent<UiButtonClick>();
                buttons.Add(btn);
                btn.SetUiController(this);
                btn.UseButtonWithTrigger(buttonPressWithTrigger);
                RectTransform uiTransformTemp =
transformGo.GetChild(i).gameObject.GetComponent<RectTransform>();
                BoxCollider colliderForPositionTemp =
transformGo.GetChild(i).gameObject.GetComponent<BoxCollider>();
                float tempThicknes = thicknes + 0.02f;
                colliderForPositionTemp.size = new
Vector3(uiTransformTemp.sizeDelta.x, uiTransformTemp.sizeDelta.y, tempThicknes);
                colliderForPositionTemp.center = new Vector3(0, 0, 0);
            }
            if (transformGo.GetChild(i).GetComponent<Slider>() != null)
            {
                transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
```

```
                UiSliderControll slider =
transformGo.GetChild(i).gameObject.AddComponent<UiSliderControll>();
                sliders.Add(slider);
                slider.SetUiController(this);
                slider.UseButtonWithTrigger(buttonPressWithTrigger);
                RectTransform uiTransformTemp =
transformGo.GetChild(i).gameObject.GetComponent<RectTransform>();
                BoxCollider colliderForPositionTemp =
transformGo.GetChild(i).gameObject.GetComponent<BoxCollider>();
                float tempThicknes = thicknes + 0.02f;
                colliderForPositionTemp.size = new
Vector3(uiTransformTemp.sizeDelta.x, uiTransformTemp.sizeDelta.y, tempThicknes);
                colliderForPositionTemp.center = new Vector3(0, 0, 0);
            }
            if (transformGo.GetChild(i).GetComponent<TMP_Dropdown>() != null ||
transformGo.GetChild(i).GetComponent<Dropdown>() != null)
            {
                transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                UiDropdownControll dropdown =
transformGo.GetChild(i).gameObject.AddComponent<UiDropdownControll>();
                dropdowns.Add(dropdown);
                dropdown.SetUiController(this);
                dropdown.UseButtonWithTrigger(buttonPressWithTrigger);
                RectTransform uiTransformTemp =
transformGo.GetChild(i).gameObject.GetComponent<RectTransform>();
                BoxCollider colliderForPositionTemp =
transformGo.GetChild(i).gameObject.GetComponent<BoxCollider>();
                float tempThicknes = thicknes + 0.02f;
                colliderForPositionTemp.size = new
Vector3(uiTransformTemp.sizeDelta.x, uiTransformTemp.sizeDelta.y, tempThicknes);
                colliderForPositionTemp.center = new Vector3(0, 0, 0);
            }
            if (transformGo.GetChild(i).GetComponent<Scrollbar>() != null)
            {
                BoxCollider bx =
transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                ScrollRect sr =
transformGo.GetChild(i).parent.GetComponent<ScrollRect>();
                bx.center = new Vector3(-
(bx.gameObject.GetComponent<RectTransform>().sizeDelta.x / 2), -
(sr.GetComponent<RectTransform>().sizeDelta.y / 2), 0);
                bx.size = new
Vector3(bx.gameObject.GetComponent<RectTransform>().sizeDelta.x,
sr.GetComponent<RectTransform>().sizeDelta.y, 0.1f);
                transformGo.GetChild(i).gameObject.AddComponent<VerticalScroll>();
                VerticalScroll temp =
sr.gameObject.transform.GetChild(sr.gameObject.transform.childCount -
1).gameObject.GetComponent<VerticalScroll>();
                temp.UseButtonWithTrigger(buttonPressWithTrigger);
            }
            RecursiveChilds(transformGo.GetChild(i).transform);
        }
    }
    public static float SliderOffsetReturn()
    {
        return SliderOffset;
    }
    public bool buttonWithTrigger()
    {
        return buttonPressWithTrigger;
    }
}
```

Table 13 InteractableCollisionUI.cs script component

```
public class UiButtonControll : MonoBehaviour
{
    [SerializeField] GameObject loadingScreen;
    [SerializeField] AudioMixer volumeControll;
    ButtonClick menu;
    private void OnEnable()
    {
        menu = gameObject.transform.parent.parent.GetComponent<ButtonClick>();
    }
    public void LoadLevel(int index)
    {
        StartCoroutine(LoadAsynchronously(index));
    }
    public void RestartLevel()
    {

StartCoroutine(LoadAsynchronously(SceneManager.GetActiveScene().buildIndex));
    }
    IEnumerator LoadAsynchronously(int sceneIndex)
    {
        loadingScreen.SetActive(true);
        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);
        while (!operation.isDone)
        {
            float progress = Mathf.Clamp01(operation.progress / .9f);
            Debug.Log(progress);
            yield return null;
        }
    }
    public void LoadExit()
    {
        Debug.Log("EXIT");
        Application.Quit();
    }
    public void BackToPlay()
    {
        menu.ButtonPress();
    }
    public void MasterVolumeSlider(float masterVolume)
    {
        volumeControll.SetFloat("MasterVolume", masterVolume);
    }
    public void FXVolumeSlider(float fxVolume)
    {
        volumeControll.SetFloat("SoundFxVolume", fxVolume);
    }
    public void MusicVolumeSlider(float musicVolume)
    {
        volumeControll.SetFloat("MusicVolume", musicVolume);
    }
}
```

Table 14 UiButtonControll.cs script component

Task #25. Add OPTIONS (2 separate sound bars for music and effects) (Eligijus and Airidas)

In the Settings window player can view his game Controls, adjust Audio or Graphics settings.

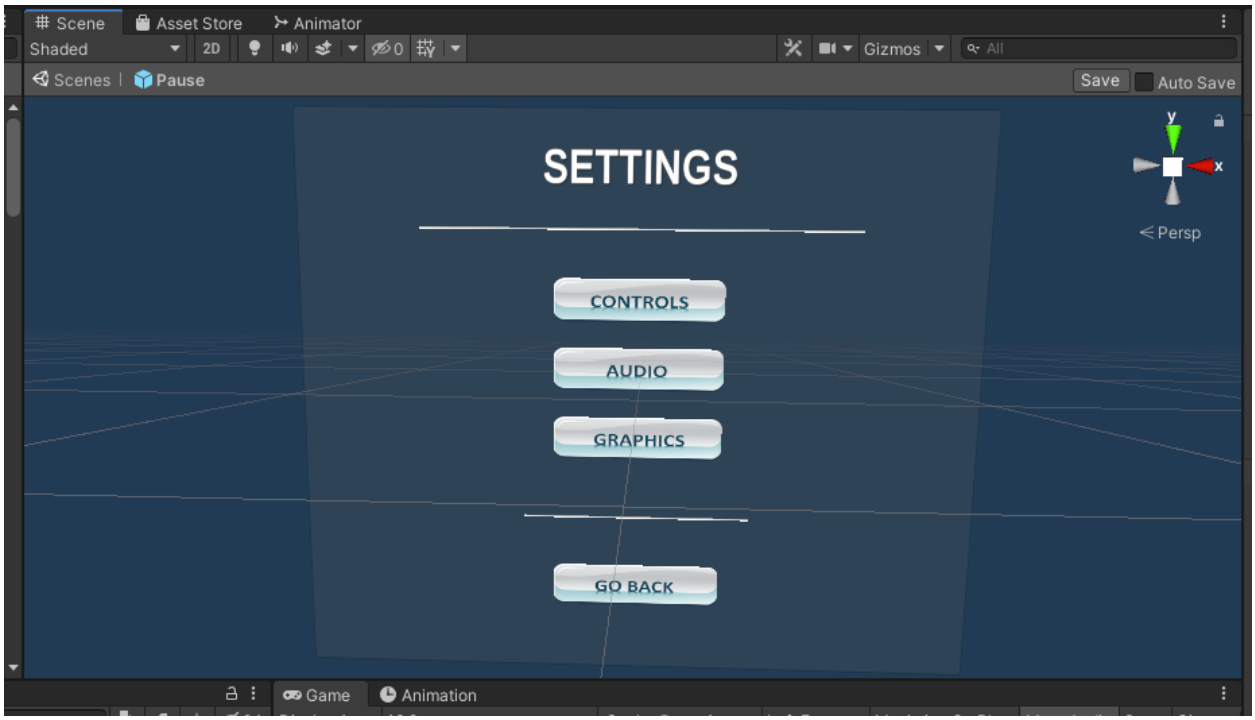

Figure 70 Settings menu

The audio settings screen includes Master, Music or Sound effects volume adjustments.

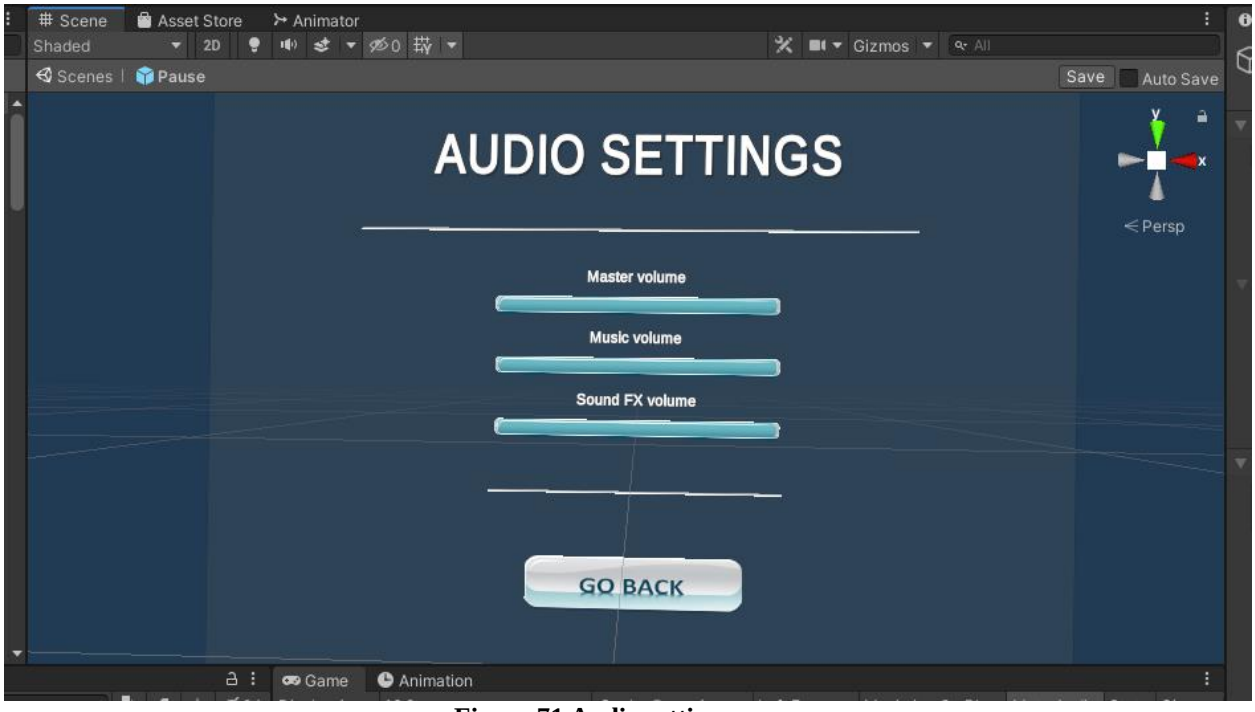

Figure 71 Audio settings menu

Controls window consists of game movement controls that are based on the virtual reality device that the player is using.

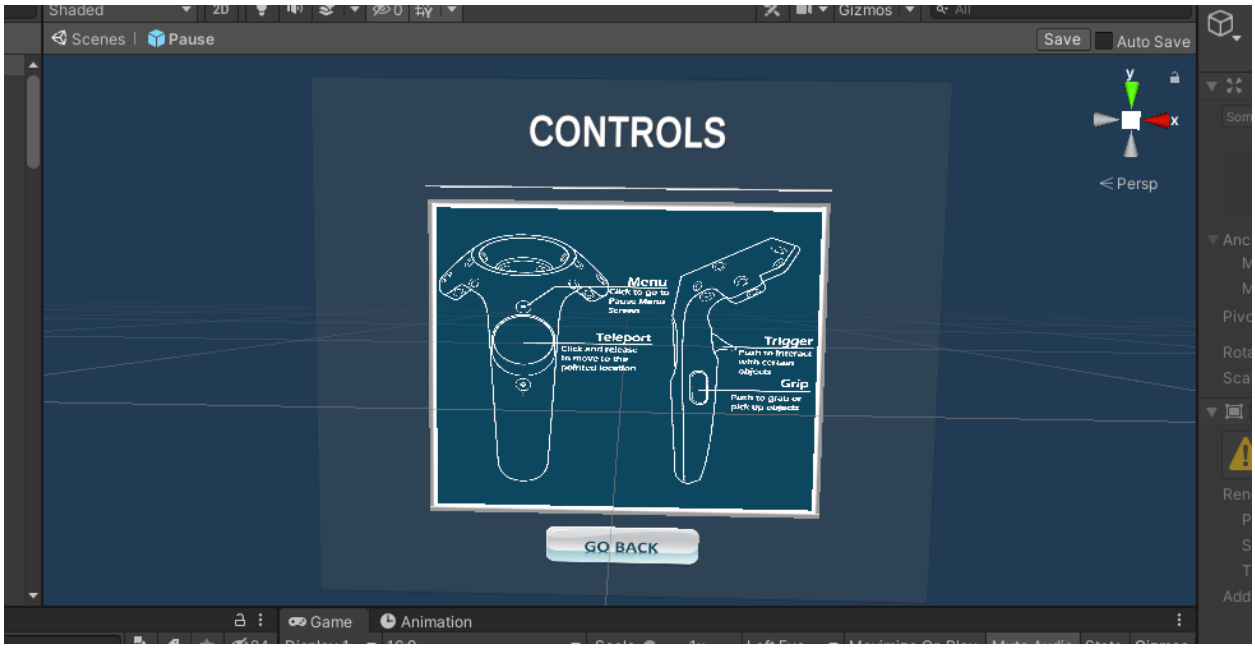

Figure 72 Controls menu

Graphics settings includes different types of adjustments – screen resolution, window mode, Overall quality level.

There is also a possibility to customize the graphics even more with additional settings, such as shadow resolution, shadow cascades, shadow quality, shadow distance, anti-aliasing, enabling or disabling soft particles, or resetting all of the graphics settings back to their defaults.

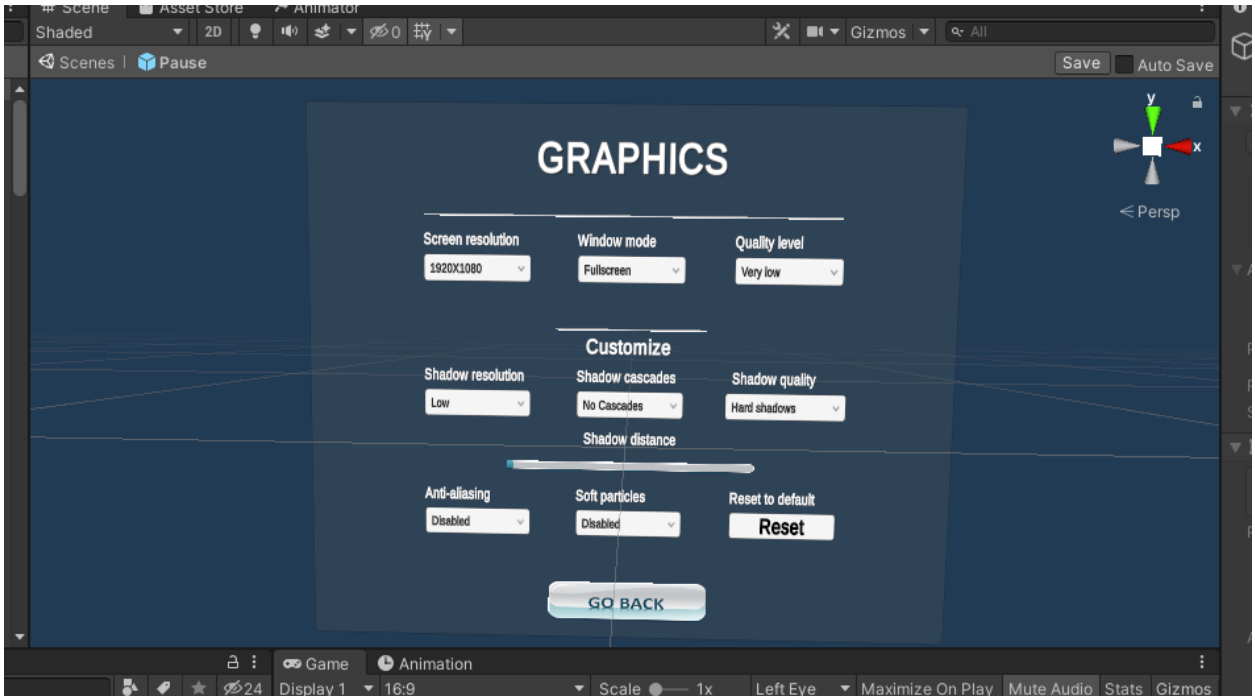

Figure 73 Graphics settings menu

```
public class StartResolutionUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    int indexResolution = 0;
    bool found = false;
    List<Resolution> res;
    public static bool fullScreen = false;
    public bool firstTime = true;
    void OnEnable()
    {
        indexResolution = 0;
        if (firstTime)
        {
            found = false;
            if (PlayerPrefs.HasKey("WindowMode"))
            {
                if (PlayerPrefs.GetInt("WindowMode") == 0)
                {
                    fullScreen = true;
                }
                else
                {
                    fullScreen = false;
                }
            }
            else
            {
                fullScreen = Screen.fullScreen;
            }
            Resolution[] resolutions = Screen.resolutions;
            dropdown = GetComponent<TMP_Dropdown>();
            dropdown.ClearOptions();
            List<string> options = new List<string>();
            res = new List<Resolution>();
            foreach (Resolution item in resolutions)
            {
                res.Add(item);
                options.Add(item.width + "X" + item.height);
                if (!PlayerPrefs.HasKey("ResolutionSave"))
                {
                    if (item.width != Screen.width && item.height != Screen.height
&& !found)
                    {
                        indexResolution++;
                    }
                    else if (item.width == Screen.width && item.height ==
Screen.height && !found)
                    {
                        found = true;
                    }
                }
                else
                {
                    if (item.width + "X" + item.height !=
PlayerPrefs.GetString("ResolutionSave") && !found)
                    {
                        indexResolution++;
                    }
                    else if (item.width + "X" + item.height ==
PlayerPrefs.GetString("ResolutionSave") && !found)
                    {
                        found = true;
                    }
                }
            }
```

```
            dropdown.AddOptions(options);
            firstTime = false;
        }
        else
        {
            found = false;
            for (int i = 0; i < res.Count; i++)
            {
                if (!PlayerPrefs.HasKey("ResolutionSave"))
                {
                    if (res[i].width == Screen.width && res[i].height ==
Screen.height && !found)
                    {
                        found = true;
                        indexResolution = i;
                    }
                }
                else
                {
                    if (res[i].width + "X" + res[i].height ==
PlayerPrefs.GetString("ResolutionSave") && !found)
                    {
                        found = true;
                        indexResolution = i;
                    }
                }
            }
        }
        dropdown.value = indexResolution;
        ChangeResolution(indexResolution);
    }
    public void ChangeResolution(int index)
    {
        Screen.SetResolution(res[index].width, res[index].height, fullScreen);
        PlayerPrefs.SetString("ResolutionSave", res[index].width + "X" +
res[index].height);
        //UpdateSettings.Save = true;
    }
}
```

Table 15 StartResolutionUi.cs script component

```
public class WindowUI : MonoBehaviour
{
    TMP_Dropdown dropdown;

    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
        }
        if (Screen.fullScreen)
        {
            dropdown.value = 0;
        }
        else
        {
            dropdown.value = 1;
        }
    }

    public void SetFullscreen(int isFullscreen)
    {
        if (isFullscreen == 0)
```

```
        {
            Screen.fullScreen = true;
            StartResolutionUi.fullScreen = true;
        }
        else if (isFullscreen == 1)
        {
            Screen.fullScreen = false;
            StartResolutionUi.fullScreen = false;
        }

        UpdateSettings.Save = true;
    }
}
```

Table 16 WindowUI.cs script component

```
public class StartQualityUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    // Start is called before the first frame update

    private void OnEnable()
    {
        if (dropdown != null)
        {
            dropdown.value = QualitySettings.GetQualityLevel();
        }
    }

    void Start()
    {
        dropdown = GetComponent<TMP_Dropdown>();
        dropdown.ClearOptions();
        List<string> options = new List<string>();

        for (int i = 0; i < QualitySettings.names.Length; i++)
        {
            options.Add(QualitySettings.names[i]);
        }
        dropdown.AddOptions(options);
        dropdown.value = QualitySettings.GetQualityLevel();

    }

    public void ChangeQuality(int index)
    {
        QualitySettings.SetQualityLevel(index, true);
        transform.parent.gameObject.SetActive(false);
        transform.parent.gameObject.SetActive(true);
        UpdateSettings.Save = true;
    }

}
```

Table 17 StartQualityUi.cs script component

```
public class ShadowQualityResUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    // Start is called before the first frame update
    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
        }
        if (QualitySettings.shadowResolution == ShadowResolution.Low)
        {
            dropdown.value = 0;
        }
        else if (QualitySettings.shadowResolution == ShadowResolution.Medium)
        {
            dropdown.value = 1;
        }
        else if (QualitySettings.shadowResolution == ShadowResolution.High)
        {
            dropdown.value = 2;
        }
        else if (QualitySettings.shadowResolution == ShadowResolution.VeryHigh)
        {
            dropdown.value = 3;
        }

    }

    public void ChangeShadowRes(int index)
    {
        if (index == 0)
        {
            QualitySettings.shadowResolution = ShadowResolution.Low;
        }
        else if (index == 1)
        {
            QualitySettings.shadowResolution = ShadowResolution.Medium;
        }
        else if (index == 2)
        {
            QualitySettings.shadowResolution = ShadowResolution.High;
        }
        else if (index == 3)
        {
            QualitySettings.shadowResolution = ShadowResolution.VeryHigh;
        }
        UpdateSettings.Save = true;
    }

}
```

Table 18 ShadowQualityResUi.cs script component

```
public class ShadowCascadesUi : MonoBehaviour
{
    TMP_Dropdown dropdown;

    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
        }
        if (QualitySettings.shadowCascades == 0)
```

```
        {
            dropdown.value = 0;
        }
        else if (QualitySettings.shadowCascades == 2)
        {
            dropdown.value = 1;
        }
        else if (QualitySettings.shadowCascades == 4)
        {
            dropdown.value = 2;
        }
    }

    public void SetShadowCascades(int index)
    {
        if (index == 0)
        {
            QualitySettings.shadowCascades = 0;
        }
        else if (index == 1)
        {
            QualitySettings.shadowCascades = 2;
        }
        else if (index == 2)
        {
            QualitySettings.shadowCascades = 4;
        }
        UpdateSettings.Save = true;
    }
}
```

Table 19 ShadowCascadesUi.cs script component

```
public class ShadowDistanceUi : MonoBehaviour
{
    Slider slider;
    private void OnEnable()
    {
        if (slider == null)
        {
            slider = GetComponent<Slider>();
        }
        slider.value = QualitySettings.shadowDistance;
    }

    public void SetShadowDistance(float index)
    {
        QualitySettings.shadowDistance = slider.value;
        UpdateSettings.Save = true;
    }
}
```

Table 20 ShadowDistanceUi.cs script component

```
public class AntiAliasing : MonoBehaviour
{
    TMP_Dropdown dropdown;

    void Start()
    {
        dropdown = GetComponent<TMP_Dropdown>();
    }

    private void OnEnable()
```

```
    {
        if (dropdown != null)
        {
            switch (QualitySettings.antiAliasing)
            {
                case 0:
                    dropdown.value = 0;
                    break;
                case 2:
                    dropdown.value = 1;
                    break;
                case 4:
                    dropdown.value = 2;
                    break;
                default:
                    dropdown.value = 3;
                    break;
            }
        }
    }

    public void SetAntiAliasing(int index)
    {
        switch (dropdown.value)
        {
            case 0:
                index = 1;
                break;
            case 1:
                index = 2;
                break;
            case 2:
                index = 4;
                break;
            default:
                index = 8;
                break;
        }
        QualitySettings.antiAliasing = index;
        UpdateSettings.Save = true;
    }
}
```

Table 21 AntiAliasing.cs script component

```
public class SoftParticlesUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    // Start is called before the first frame update
    private void OnEnable()
    {
        if (dropdown == null)
            dropdown = GetComponent<TMP_Dropdown>();
        if (QualitySettings.softParticles == false)
        {
            dropdown.value = 0;
        }
        else
            dropdown.value = 1;
    }

    public void SetSoftParticle(int index)
    {
        if (index == 0)
        {
```

```
            QualitySettings.softParticles = false;
        }
        else if (index == 1)
        {
            QualitySettings.softParticles = true;
        }
        UpdateSettings.Save = true;
    }
}
```

Table 22 SoftParticlesUi.cs script component

```
public class ShadowQualityUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
            if (QualitySettings.shadows == ShadowQuality.HardOnly)
            {
                dropdown.value = 0;
            }
            else if (QualitySettings.shadows == ShadowQuality.All)
            {
                dropdown.value = 1;
            }
        }
        else
        {
            if (QualitySettings.shadows == ShadowQuality.HardOnly)
            {
                dropdown.value = 0;
            }
            else if (QualitySettings.shadows == ShadowQuality.All)
            {
                dropdown.value = 1;
            }
        }
    }

    public void ChangeShadowQuality(int index)
    {
        if (index == 0)
        {
            QualitySettings.shadows = ShadowQuality.HardOnly;
        }
        else if (index == 1)
        {
            QualitySettings.shadows = ShadowQuality.All;
        }
        UpdateSettings.Save = true;
    }
}
```

Table 23 ShadowQualityUi.cs script component

Task #26. Game must have a GUI (elements should vary based on your game: health bars, scores, money, resources, button to go back to main menu, etc.) (Airidas and Eligijus)

Pause menu has an option to go to main menu scene, leaderboard sums up all of the gathered points during the course of the game.

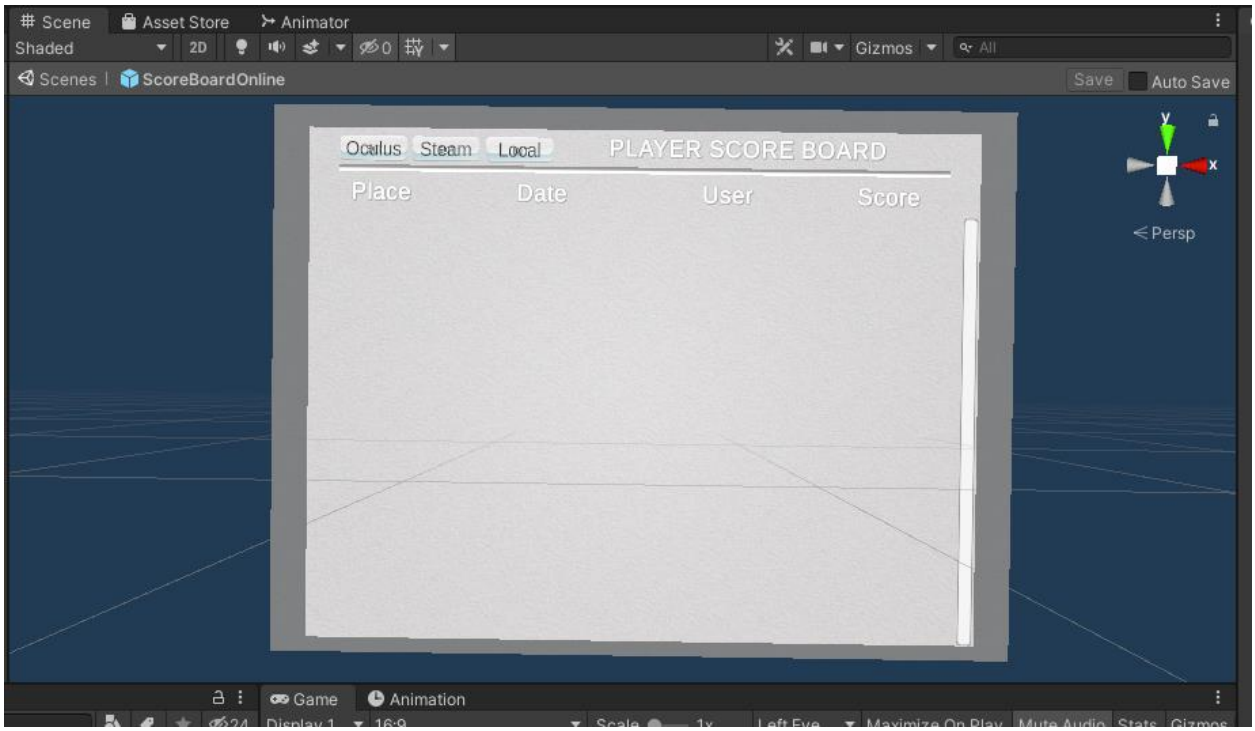

Figure 74 Leaderboard

Task #27. Add attack mechanics (might be replaced with puzzle mechanics, push off the road mechanics (racing game), etc.) (Airidas)

Since this is more of a Sandbox Puzzle title, the game consists of different Puzzle mechanics, such as code locks, timers and so on.

One of these mechanics include searching for three different parts of code that are written on hidden notes. After picking the note, the player can track his progress by looking at the grey shelf, which gets constantly updated when a number is found.

Figure 75 Code tracking shelf

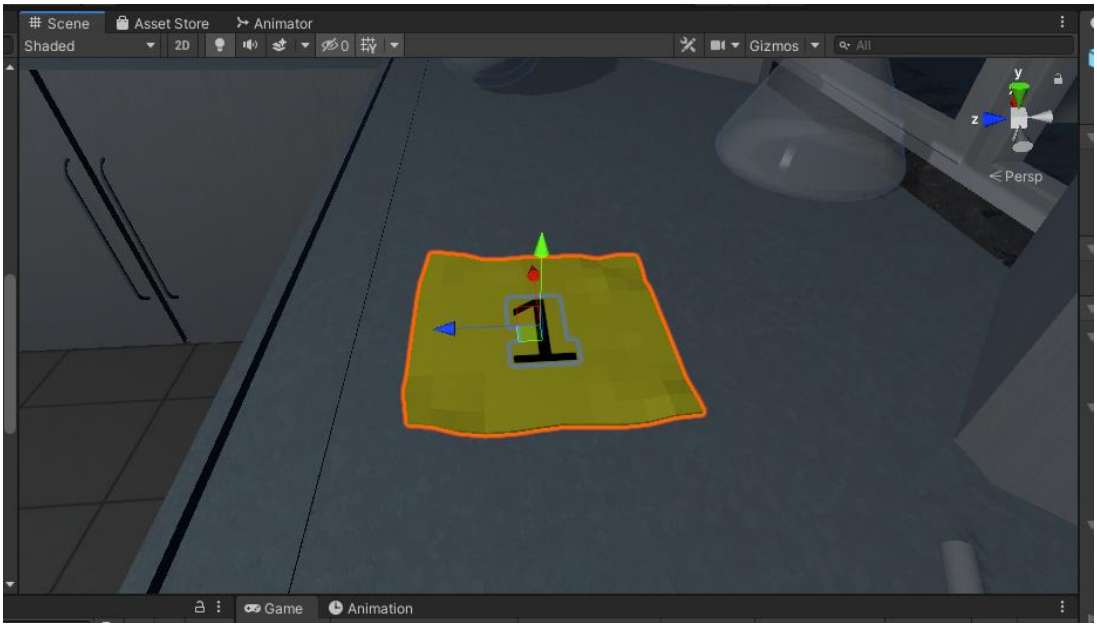

Figure 76 Collectable Note

```
public class NoteBoard : MonoBehaviour
{
    [SerializeField] GameObject printKey;
    bool isGrabbed = false;
    TextMeshPro print;
    TextMeshPro key;
    int count = 0;
       [SerializeField] DelegateChange Task;
    void Start()
    {
        print = printKey.GetComponent<TextMeshPro>();
        key = GetComponentInChildren<TextMeshPro>();
    }

    void FixedUpdate()
    {

        if (isGrabbed)
        {
            print.text = key.text;
                    Task.AddTask();
        }
    }

    public void Grab()
    {
        isGrabbed = true;
    }

    public void UnGrab()
    {
        isGrabbed = false;
    }

}
```

Table 24 NoteBoard.cs script component

After collecting all of the numbers, the player has the possibility to write the code into the code lock, which unlocks the exit door if the code is written in correctly.

Figure 77 Code lock

```
public class LockUnlockWithPin : MonoBehaviour
{
    [SerializeField] GameObject doorsControll;
    [SerializeField] Rigidbody doorsControllRb;
    [SerializeField] int keyLenght = 0;
    [SerializeField] int[] customKey;
    [SerializeField] GameObject[] keyObjects;
    [SerializeField] TimeLeft timer;
    [SerializeField] GroundControll ground;
    [SerializeField] float MinusTimeBad;
    [SerializeField] AccessGranted access;
    [SerializeField] bool x = false;
    [SerializeField] bool y = false;
    [SerializeField] bool z = false;
    AccessDenied accessDenied;
    string tempTMPRO = "";
    StringBuilder stringBuild;
    TextMeshPro text;
    public bool open = false;
    bool fail = true;
    bool equals = true;
    int count = 0;
    int countCheck = 0 ;
    int[] key;
    int[] playerNumbers;
    bool firstTime = true;
       [SerializeField] DelegateChange Task;

    bool correct = false;

    bool saveBool = false;
    // Start is called before the first frame update
    void Start()
    {
```

```
        stringBuild = new StringBuilder();
        text = GetComponent<TextMeshPro>();
        accessDenied = GetComponent<AccessDenied>();
        doorsControll.SetActive(false);
        if (x)
        {
            doorsControllRb.constraints = RigidbodyConstraints.FreezeRotationX;
        }
        else if (y)
        {
            doorsControllRb.constraints = RigidbodyConstraints.FreezeRotationY;
        }
        else if (z)
        {
            doorsControllRb.constraints = RigidbodyConstraints.FreezeRotationZ;
        }
        if (customKey.Length == 0)
        {
            key = new int[keyLenght];
            playerNumbers = new int[keyLenght];
            for (int i = 0; i < keyLenght; i++)
            {
                key[i] = UnityEngine.Random.Range(0, 9);
                keyObjects[i].GetComponentInChildren<TextMeshPro>().text =
key[i].ToString();
                Debug.Log(key[i]);
            }
        }
        else
        {
            keyLenght = customKey.Length;
            key = customKey;
            playerNumbers = new int[keyLenght];
        }


    }

    // Update is called once per frame
    void Update()
    {
        if (firstTime)
        {
            firstTime = false;
        }
        if (countCheck < keyLenght)
        {
            if (key[countCheck] != playerNumbers[countCheck])
            {
                equals = false;
            }

        }
        countCheck++;

        if (equals && count == keyLenght && countCheck >= keyLenght)
        {
            doorsControll.SetActive(true);
            doorsControllRb.constraints = RigidbodyConstraints.None;
            text.text = "Access Granted";
            timer.StopTimer();
            timer.finalStop = true;
                    Task.AddTask();
            GameData.SetTime((int)Math.Truncate(timer.timerTime));
            GameData.SetEnd(true);
```

```
            GameData.SetVictory(true);
            if (ground != null)
            {
                ground.finishOn();
            }
            open = true;
            if (saveBool == false)
            {
                GlobalData.WriteToFile();
                saveBool = true;
            }
        }
        if (!equals && count == keyLenght && !open && countCheck >= keyLenght)
        {
            text.text = "Access Denied";
            accessDenied.StartSound();
            timer.MinusTime(MinusTimeBad);
            count = 0;
            playerNumbers = new int[keyLenght];
            fail = true;
        }

        if (countCheck >= keyLenght && !equals)
        {
            countCheck = 0;
            equals = true;
        }

        if(open && !correct)
        {
            correct = true;
            access.PlaySound();
        }

    }

    public void SetNumber(int val)
    {
        if (count < keyLenght)
        {

            if (fail)
            {
                stringBuild = new StringBuilder();
                stringBuild.Append(tempTMPRO);
                text.text = stringBuild.ToString();
                fail = false;

            }
            if (!open && count >= 0)
            {
                playerNumbers[count] = val;
                stringBuild.Append(" " + val);
                text.text = stringBuild.ToString();
                count++;
            }
            else if (!open && count < 0)
            {
                count = 0;
                playerNumbers[count] = val;
                stringBuild.Append(" " + val);
                text.text = stringBuild.ToString();
                count++;
            }
        }
```

```
    }

    public void Del()
    {
        if (!open && count >= 0)
        {
            if (count == 0)
            {
                stringBuild.Remove(stringBuild.Length - 1, 1);
                text.text = stringBuild.ToString();
            }
            else
            {
                stringBuild.Remove(stringBuild.Length - 2, 2);
                text.text = stringBuild.ToString();
            }
            count--;
            if (count < 0)
            {
                stringBuild = new StringBuilder();
                stringBuild.Append(tempTMPRO);
                text.text = stringBuild.ToString();
            }
        }
    }
}
    }

}
```

Table 25 LockUnlockWithPin.cs script component

Task #28. Add attack mechanics (might be replaced with puzzle mechanics, push off the road mechanics (racing game), etc.) (Eligijus)

Other mechanics include searching for key that unlocks door which is hidden in flask. After breaking flask with key, key spawns in spot where flask was broken.

Figure 78 Flask with key

Figure 79 Spawn key

```
public class KeySpawn : MonoBehaviour
{
    public GameObject key;
    public GameObject Flask;
    public SteamVrSceleton skeleton;
       [SerializeField] DelegateChange Task;

    int count;

    private void Start()
    {
        count = 1;
        skeleton = GetComponent<SteamVrSceleton>();
    }

    // Update is called once per frame
    void Update()
    {
        if(Flask == null && count == 1)
        {
            InstantiateKey();
        }
    }

    private void InstantiateKey()
    {
        GameObject reference = Instantiate(key, transform.position,
transform.rotation);
        //reference.GetComponent<SteamVrSceleton>().LeftHand = skeleton.LeftHand;
        //reference.GetComponent<SteamVrSceleton>().RightHand = skeleton.RightHand;
             Task.AddTask();
        Debug.Log("Key spawned!");
        count++;
    }
```

Table 26 Spawn key script

After finding key, the player has the possibility to unlock door by putting key to keyhole and realizing grabbing button.

Figure 80 unlock door

```
public class LockUnlockWithKey : MonoBehaviour
{
    [SerializeField] GameObject[] Note;
    [SerializeField] GameObject doorsControll;
    [SerializeField] Rigidbody rb;
    [SerializeField] string checkKey;
    [SerializeField] bool x = false;
    [SerializeField] bool y = false;
    [SerializeField] bool z = false;
    bool isSnapped = false;
    IdForUnlock[] unlock;
    public bool Open = false;
    bool oneTime = true;
    bool check = false;
    private int unlockId = 0;
    bool rigidbodyExists = false;
    int index = 0;
       [SerializeField] DelegateChange Task;
    // Start is called before the first frame update
    void Start()
    {
        doorsControll.SetActive(false);
        if (Note.Length > 0)
        {
            foreach (GameObject obj in Note)
            {
                if (obj != null)
                {
                    obj.SetActive(false);
                }
            }
        }
        if (x)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationX;
        }
        if (y)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationY;
        }
        if (z)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationZ;
        }
    }

    // Update is called once per frame
    void Update()
    {

        if (!check && isSnapped)
        {
            unlock = FindObjectsOfType<IdForUnlock>();
            if (index < unlock.Length)
            {
                if (unlock[index] != null && unlock[index].tag == checkKey)
                {
                    int id = Random.Range(1, 20);
                    unlockId = id;
                    unlock[index].SetId(id);
                    check = true;
                }
                else if(unlock[index] != null && unlock[index].tag != checkKey)
                {
                    index++;
```

```
                }

            }
            else if(index >= unlock.Length)
            {
                index = 0;
            }

        }
        else
        {
            if (Open == true && oneTime)
            {

                doorsControll.SetActive(true);
                rb.constraints = RigidbodyConstraints.None;
                foreach (GameObject obj in Note)
                {
                    obj.SetActive(true);
                }
                oneTime = false;
                rigidbodyExists = true;
                           Task.AddTask();
                Destroy(this);
            }
            if (isSnapped && unlock[index].GetId() == unlockId)
            {
                Open = true;
            }
        }


    }

    public bool isChestOpen()
    {
        if (rigidbodyExists)
            return Open;
        else
            return false;
    }

    public void Snap()
    {
        isSnapped = true;
    }
    public void UnSnap()
    {
        isSnapped = false;
    }

}
```

Table 27 LockUnlockWithKey.cs script component

Task #29. Implement Opponents (enemy moves towards you and tries to kill you, might be replaced with random puzzles, race opponents, etc.) (Airidas)

One of the main puzzles in the game is called Blue flame experiment. The main goal of this experiment is to find a part of Aluminium, two liquids and mix these components into a barrel. After mixing everything up, the player has to find a lighter and use it to light up the top of the barrel.

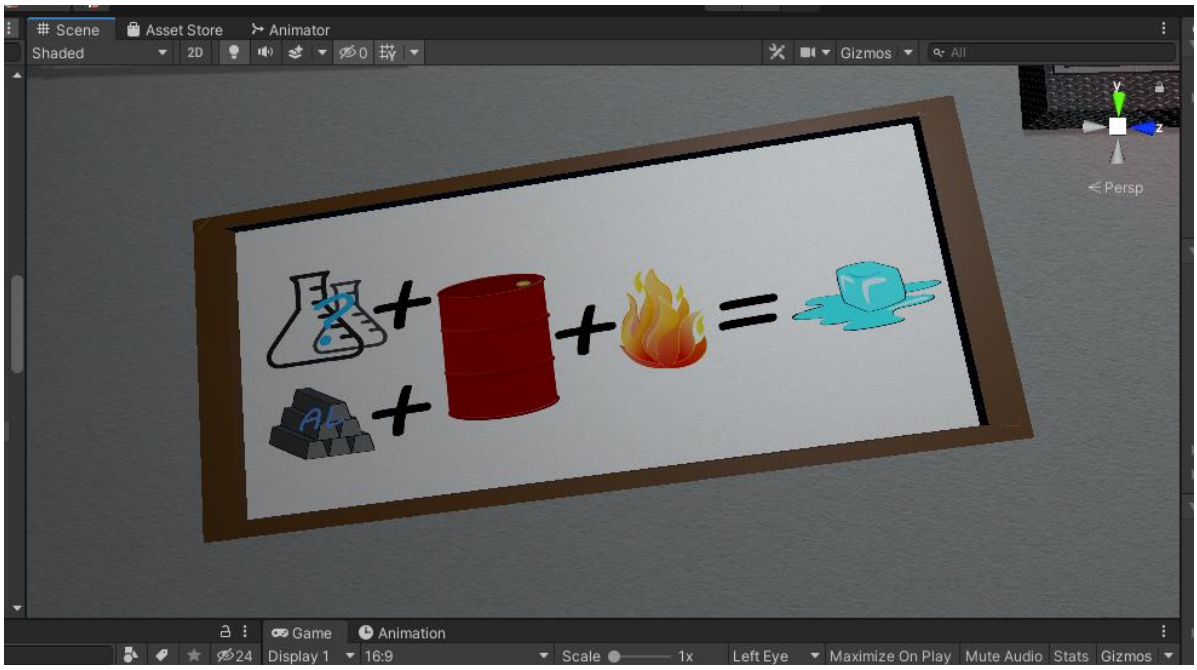

Figure 81 Blue flame experiment hint

To find out what liquids are the correct ones, the player has to look for a hint note. The correct hint note displays a combination of numbers – which are atomic numbers from Chemical period table. The top line A1 displays the first liquid, which is CuSO4 and the second one A2 is HCl.

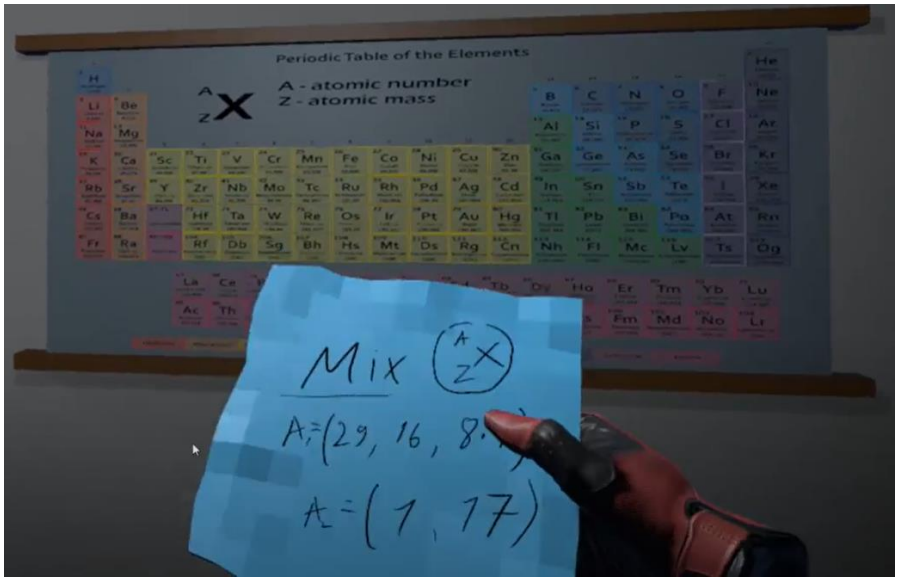

Figure 82 Chemical period table hint

After being lit up, the barrel shoots a blue flame upwards that melts away the ice, which had a frozen hint note inside of it.

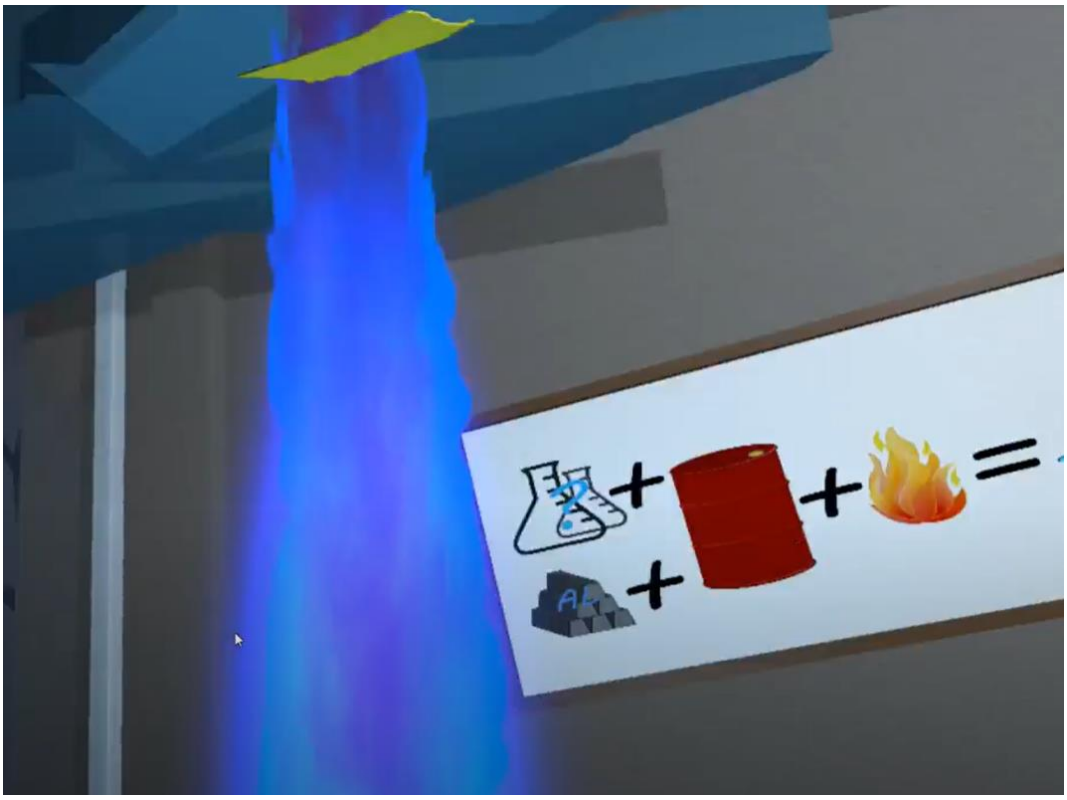

Figure 83 Blue flame experiment

```
public class MixingScript : MonoBehaviour
{
    [SerializeField] string mixedTag = "Mixed";
    [SerializeField] int specialLimit = 0;
    [SerializeField] string fire;
    [SerializeField] List<string> ignoreCollision;
    [SerializeField] GameObject ps;
    [SerializeField] List<string> LiquidTags;
    [SerializeField] int[] absorve;
    [SerializeField] bool special;
    List<string> tags;
    public bool mix = true;
    int[] liquidAbsorve;
    int MixSpec = 0;
    int check = 0;
    int cnt = 0;
    int indexCheckTag = 0;
    bool checkForMixed = false;
    public bool mixed = false;

    void Start()
    {
        liquidAbsorve = new int[LiquidTags.Count];
        tags = new List<string>();
    }

    void OnParticleCollision(GameObject other)
    {

        if (fire != other.tag && other.tag != "Untagged" &&
!ignoreCollision.Contains(other.tag) && other.tag != mixedTag)
        {
```

```
            if (!tags.Contains(other.tag))
            {
                tags.Add(other.tag);
            }
            int i = 0;
            foreach (string tag in LiquidTags)
            {
                if (other.tag == tag && absorve[i] > liquidAbsorve[i])
                {
                    liquidAbsorve[i]++;
                    break;
                }
                i++;
            }
        }
        else if (other.tag == mixedTag && special)
        {
            MixSpec++;

        }

    }
    void Update()
    {
        if (tags.Count >= LiquidTags.Count)
        {
            if (indexCheckTag < tags.Count)
            {
                checkForMixed = false;
                string tag = tags[indexCheckTag];
                if (!LiquidTags.Contains(tag))
                {
                    mix = false;
                }
                indexCheckTag++;
            }
            else if (indexCheckTag >= tags.Count)
            {
                indexCheckTag = 0;
                checkForMixed = true;
            }

            if (checkForMixed)
            {
                if (!mixed)
                {
                    if (cnt < liquidAbsorve.Length)
                    {
                        if (liquidAbsorve[cnt] >= absorve[cnt])
                        {
                            check++;
                        }
                        cnt++;
                    }
                    else
                    {
                        cnt = 0;
                        check = 0;
                    }
                    if ((check == liquidAbsorve.Length && tags.Count ==
LiquidTags.Count && mix) || (special && specialLimit <= MixSpec && mix))
                    {
                        mixed = true;
                        ps.tag = mixedTag;
                    }
```

```
                }
                else
                {
                    if ((check == liquidAbsorve.Length && tags.Count >
LiquidTags.Count && !mix) || (!mix && special && (tags.Count < LiquidTags.Count)))
                    {
                        mixed = false;
                        ps.tag = "Untagged";
                    }
                }
            }
        }
    }

    public void ResetMix()
    {
        liquidAbsorve = new int[LiquidTags.Count];
        tags = new List<string>();
        checkForMixed = false;
        indexCheckTag = 0;
        mixed = false;
        mix = true;
        cnt = 0;
        check = 0;
        ps.tag = "Untagged";
    }

}
```

Table 28 MixingScript.cs script component

```
public class LightOn : MonoBehaviour
{
    [SerializeField] string TagForFire;
    ParticleSystem ps;
    MixingScript mix;
    [SerializeField] ObjectCheck aluminium;
    public bool Started = false;
    float temptime = 0;
    float time;
       [SerializeField] DelegateChange Task;

    void Start()
    {
        mix = GetComponent<MixingScript>();
        ps =
gameObject.transform.parent.Find("Parent").GetComponent<ParticleSystem>();
        time = ps.main.duration;
    }

    void OnParticleCollision(GameObject other)
    {
        if (TagForFire == other.tag && !Started && mix.mixed && aluminium.added)
        {
            gameObject.transform.parent.Find("Parent").gameObject.SetActive(true);
            ps.Play();
            Started = true;
                    Task.AddTask();
        }
    }

    void Update()
    {
        if (Started && time > temptime && mix.mixed && aluminium.added)
        {
            temptime += Time.deltaTime;
```

```
        }
        else if(Started && time <= temptime && mix.mixed && aluminium.added)
        {
            Started = false;
            temptime = 0;
        }
    }

}
```

Table 29 LightOn.cs script component

Task #30. Add health / powerup mechanics and indication in the GUI (Airidas and Eligijus)

Since Escape the Lab title is a Virtual Reality exploration puzzle title, there are no power ups available. Instead, the players are given various tools and objects that help them to complete the levels.
The timer acts like health in the game – if it hits zero, the player loses and gets dropped into Game Over scene.

Figure 84 Timer

Task #31. Implement Opponents (enemy moves towards you and tries to kill you, might be replaced with random puzzles, race opponents, etc.) (Eligijus)

One of the main puzzles in the game is called chameleon. The main goal of this experiment is to find a part of sugar, two liquids and mix these components into each other. After mixing everything up, the player must wait for reaction.

To find out what liquids are the correct ones, the player must look for a hint notebook. The notebook displays a combination of formulas – which are written on flask. The formula line represents which liquids are needed to mix and last line (sugar) represents that after mixing sugar must be added to mixed liquids.

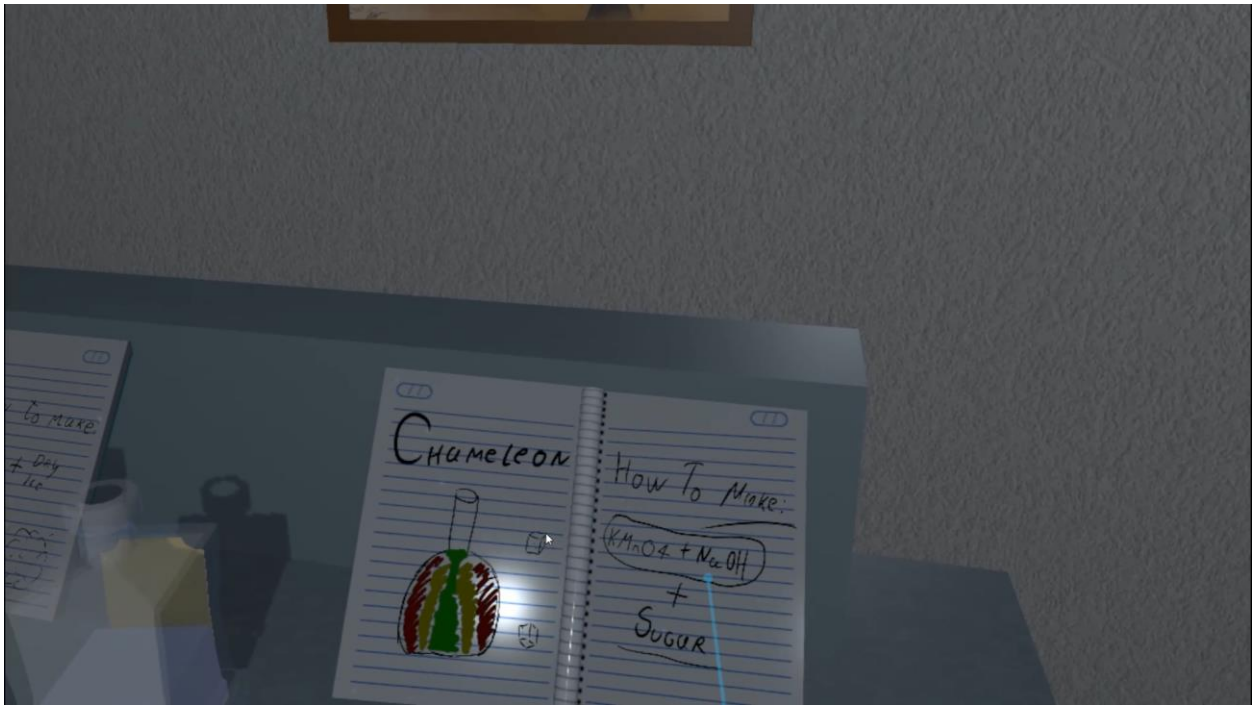

Figure 85 Chameleon experiment hint

After liquids are mixed and sugar added to the liquid, then liquid should start changing its colors.

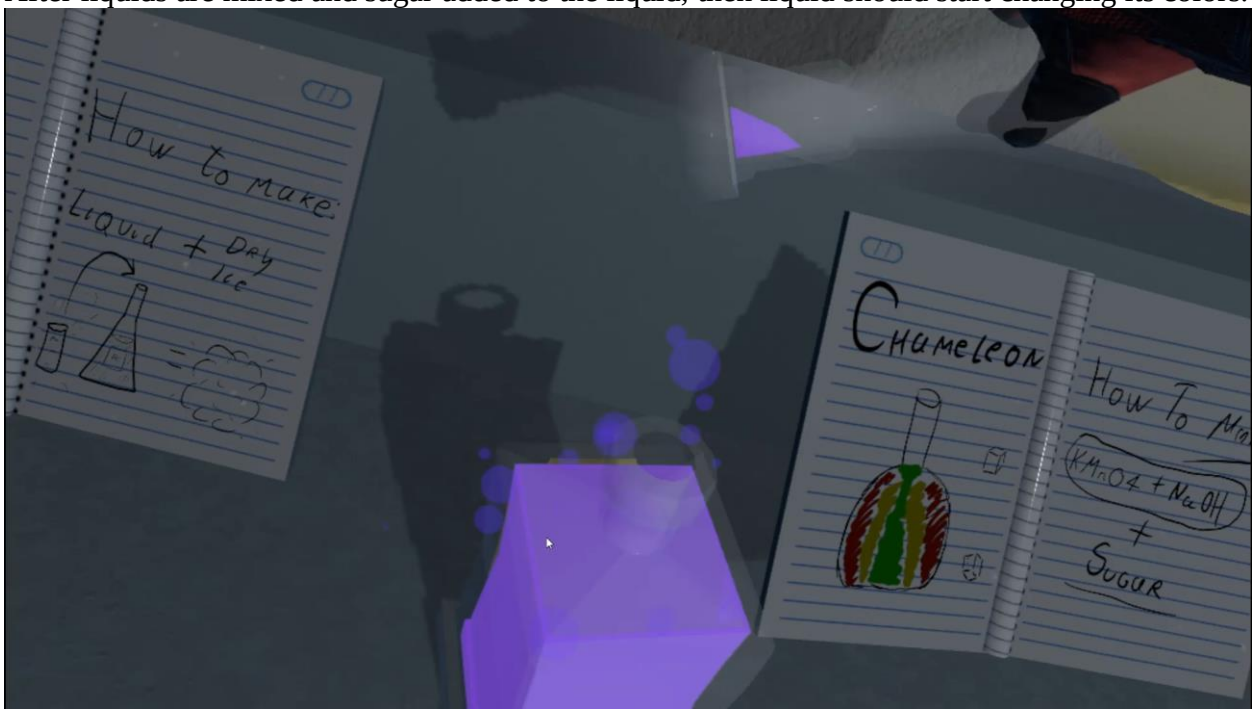

Figure 86 Mix liquids

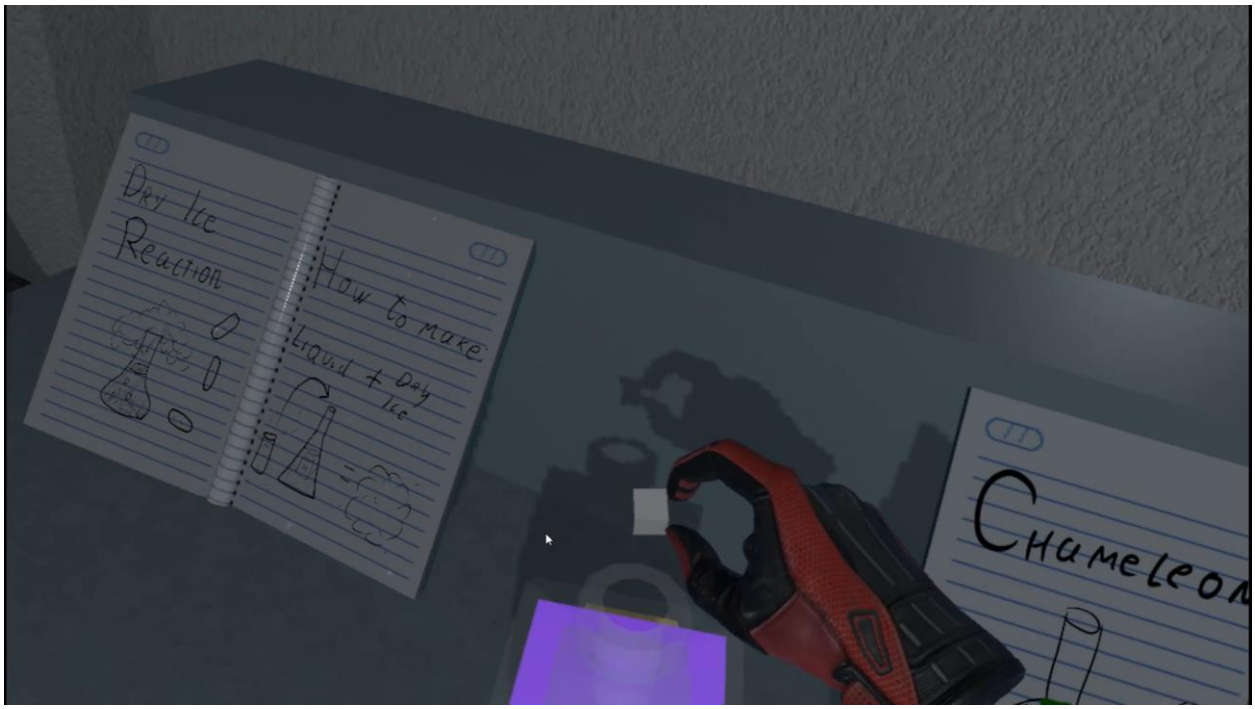

Figure 87 Add sugar to mixed liquid

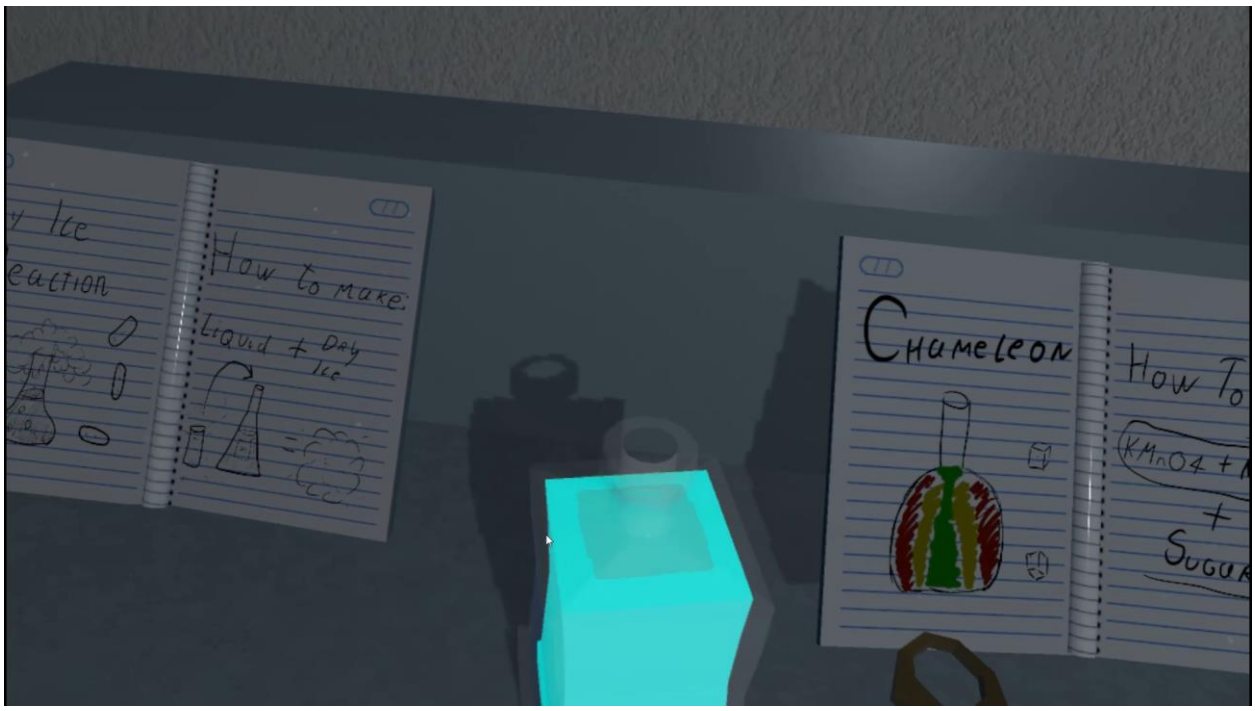

Figure 88 Liquid changing colors

```
public class MixingScript : MonoBehaviour
{
    [SerializeField] string mixedTag = "Mixed";
    [SerializeField] int specialLimit = 0;
    [SerializeField] string fire;
    [SerializeField] List<string> ignoreCollision;
    [SerializeField] GameObject ps;
    [SerializeField] List<string> LiquidTags;
    [SerializeField] int[] absorve;
    [SerializeField] bool special;
    List<string> tags;
    public bool mix = true;
    int[] liquidAbsorve;
    int MixSpec = 0;
    int check = 0;
    int cnt = 0;
    int indexCheckTag = 0;
    bool checkForMixed = false;
    public bool mixed = false;

    void Start()
    {
        liquidAbsorve = new int[LiquidTags.Count];
        tags = new List<string>();
    }

    void OnParticleCollision(GameObject other)
    {

        if (fire != other.tag && other.tag != "Untagged" &&
!ignoreCollision.Contains(other.tag) && other.tag != mixedTag)
        {
            if (!tags.Contains(other.tag))
            {
                tags.Add(other.tag);
            }
            int i = 0;
            foreach (string tag in LiquidTags)
            {
                if (other.tag == tag && absorve[i] > liquidAbsorve[i])
```

```
                {
                    liquidAbsorve[i]++;
                    break;
                }
                i++;
            }
        }
        else if (other.tag == mixedTag && special)
        {
            MixSpec++;

        }

    }
    void Update()
    {
        if (tags.Count >= LiquidTags.Count)
        {
            if (indexCheckTag < tags.Count)
            {
                checkForMixed = false;
                string tag = tags[indexCheckTag];
                if (!LiquidTags.Contains(tag))
                {
                    mix = false;
                }
                indexCheckTag++;
            }
            else if (indexCheckTag >= tags.Count)
            {
                indexCheckTag = 0;
                checkForMixed = true;
            }

            if (checkForMixed)
            {
                if (!mixed)
                {
                    if (cnt < liquidAbsorve.Length)
                    {
                        if (liquidAbsorve[cnt] >= absorve[cnt])
                        {
                            check++;
                        }
                        cnt++;
                    }
                    else
                    {
                        cnt = 0;
                        check = 0;
                    }
                    if ((check == liquidAbsorve.Length && tags.Count ==
LiquidTags.Count && mix) || (special && specialLimit <= MixSpec && mix))
                    {
                        mixed = true;
                        ps.tag = mixedTag;
                    }
                }
                else
                {
                    if ((check == liquidAbsorve.Length && tags.Count >
LiquidTags.Count && !mix) || (!mix && special && (tags.Count < LiquidTags.Count)))
                    {
                        mixed = false;
                        ps.tag = "Untagged";
```

```
                    }
                }
            }
        }
    }

    public void ResetMix()
    {
        liquidAbsorve = new int[LiquidTags.Count];
        tags = new List<string>();
        checkForMixed = false;
        indexCheckTag = 0;
        mixed = false;
        mix = true;
        cnt = 0;
        check = 0;
        ps.tag = "Untagged";
    }

}
```

Table 30 MixingScript.cs script component

```
public class ChameleonCheck : MonoBehaviour
{
    [SerializeField] string changeTagMixed = "SugarNaOHKMnO4";
    [SerializeField] string changeTagNoMixed = "SugarNaOH";
    [SerializeField] string tag;
    [SerializeField] MixingScript mix;
    [SerializeField] ParticleSystem ps;
    [SerializeField] GameObject material;
    [SerializeField] LiquidVolumeAnimator lva;
    public bool check;
       [SerializeField] DelegateChange Task;

    public static bool chameleonActive = false;

    BottleSmash smash;
    bool change = false;
    int cnt = 0;
    GameObject go;
    Color color;

    private void Start()
    {
        chameleonActive = false;
        mix = gameObject.GetComponent<MixingScript>();
        smash = gameObject.transform.parent.GetComponent<BottleSmash>();
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.collider.tag == tag)
        {

            go = collision.gameObject;
            go.GetComponent<Melting>().lva = lva;
            go.GetComponent<Melting>().OneTimeTrigger = true;
            collision.gameObject.GetComponent<BoxCollider>().isTrigger = true;
            color = lva.mats[0].GetColor("_Color");

        }
    }

    private void Update()
    {
```

```
        if (go != null && go.transform.position.y < lva.finalPoint.y && !check &&
mix.mixed)
        {

            ps.tag = changeTagMixed;
            check = true;
        }
        else if (go != null && go.transform.position.y < lva.finalPoint.y && !check
&& !mix.mixed)
        {
            ps.tag = changeTagNoMixed;

        }
        else if (go == null && !check && mix.mixed && ps.tag == changeTagMixed)
        {
            color = lva.mats[0].GetColor("_Color");
            check = true;
        }
        if (check && ps.tag == changeTagMixed)
        {
                    chameleonActive = true;
                    Task.AddTask();
                    if (!change)
            {
                if (cnt == 0)
                {
                    color.r -= 0.01f;
                    if (color.r <= 0)
                    {
                        color.r = 0;
                        change = true;
                        cnt++;
                    }
                }
                else if (cnt == 2)
                {
                    color.b -= 0.01f;
                    if (color.b <= 0)
                    {
                        color.b = 0;
                        change = true;
                        cnt++;
                    }
                }
                else if (cnt == 4)
                {
                    color.g -= 0.01f;
                    if (color.g <= 0)
                    {
                        color.g = 0;
                        change = true;

                    }
                }
            }
            else if (change)
            {
                if (cnt == 1)
                {
                    color.g += 0.01f;
                    if (color.g >= 1)
                    {
                        color.g = 1;
                        change = false;
                        cnt++;
```

```
                    }
                }
                else if (cnt == 3)
                {
                    color.r += 0.01f;
                    if (color.r >= 1)
                    {
                        color.r = 1;
                        change = false;
                        cnt++;
                    }
                }
            }

            smash.color = color;
            smash.ChangedColor();

            if (cnt == 5)
            {
                check = false;
                Destroy(this);
            }


        }
    }

}
```

Table 31 ChameleonCheck.cs script component

Another one of the main puzzles in the game is called dry ice. The main goal of this experiment is to find a dry ice cylinder and some liquid. After adding dry ice to liquid, cloud of smoke is created. There is a notebook with hint that shows how experiment works.

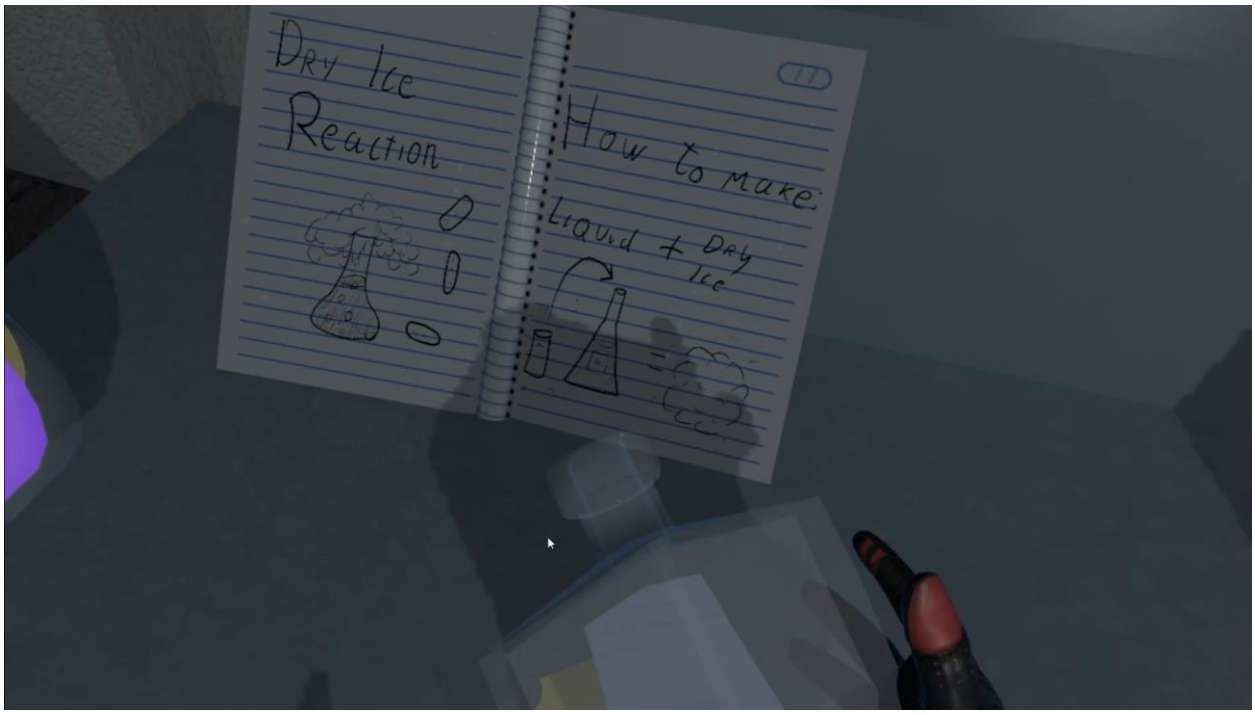

Figure 89 Experiment hint

Figure 90 Dry Ice box

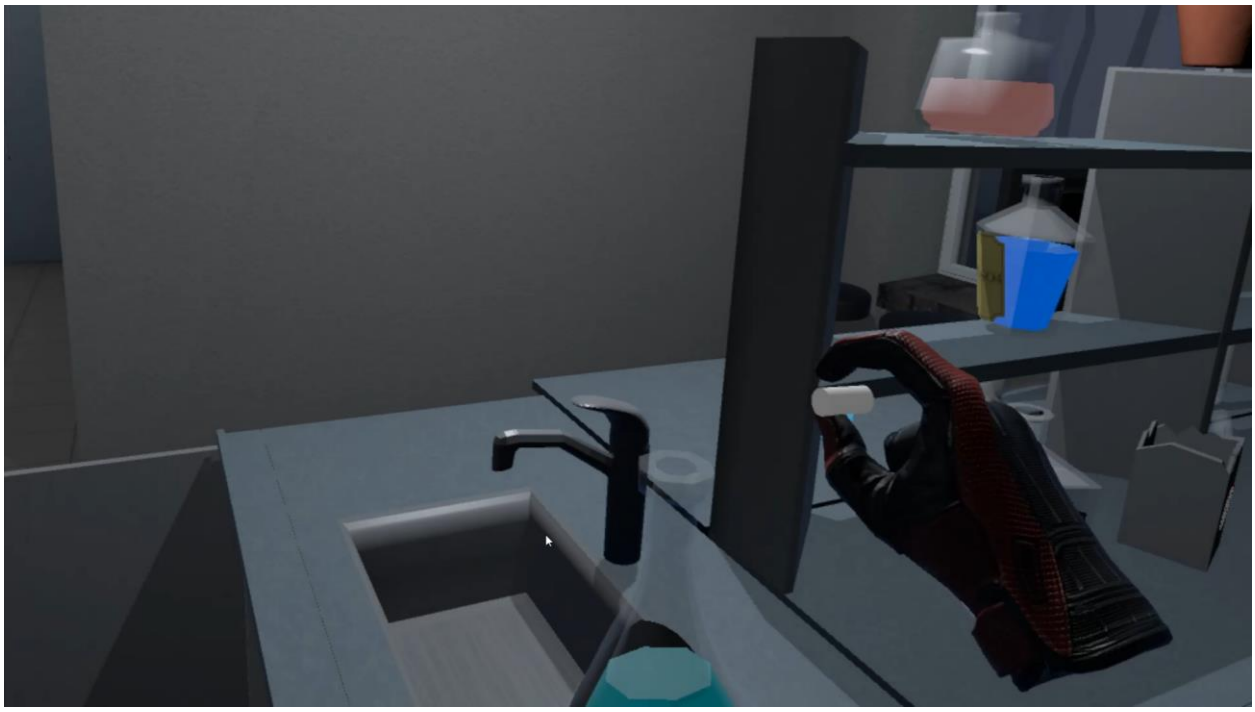

Figure 91 Add dry ice to liquid

```
public class DryIceCheck : MonoBehaviour
{
    [SerializeField] string tag;
    [SerializeField] ParticleSystem ps;
    public LiquidVolumeAnimator lva;
    [SerializeField] BottleSmash bottle;
    [SerializeField] LiquidLevel level;
       [SerializeField] DelegateChange Task;

       public static bool dryIceActive = false;

    private void Start()
    {
```

```
        dryIceActive = false;
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.collider.tag == tag)
        {
                    dryIceActive = true;
            collision.gameObject.GetComponent<Melting>().lva = lva;
            collision.gameObject.GetComponent<Collider>().isTrigger = true;
            collision.gameObject.GetComponent<Melting>().pscheck = ps;
            collision.gameObject.GetComponent<Melting>().collided = true;
            collision.gameObject.GetComponent<Melting>().smash = bottle;
            level.pscheck = ps;
                    Task.AddTask();
        }
    }
}
```

Table 32 DryIce Component

Task #32. Add scoring system (e.g., Easiest enemy – 100, most difficult – 500, powerup – 1000, total game time calculated, etc.) (Eligijus and Airidas)

When the finish is reached, player has the possibility to write his name into the leaderboard and compete with Local, Steam store or Oculus store high scores. The total score is measured by how quickly each level has been finished.

Figure 92 Local leaderboard

Figure 93 Leaderboard name input keyboard

```
public class ScoreBoardDatabase : MonoBehaviour
{
    public static MySqlConnection MySqlConnetionRef;
    [SerializeField] string tableName;
    [SerializeField] bool isLocal = false;
    [SerializeField] int AllLevels = 0;
    [SerializeField] GameObject prefab;
    [SerializeField] GameObject loading;
    [SerializeField] float firstXCoordinate;
    [SerializeField] float firstYCoordinate;
    [SerializeField] float firstZCoordinate;
    [SerializeField] float offSet;
    MySqlConnection connection;
    float tempPosition;
    List<Data> allData;
    WriteReadFile file;
    bool oneTime = true;
    bool dataUpdated = false;
    Thread onlineData;
    Vector2 firstDimmensionsRect = Vector2.zero;
    bool usingSteam = false;

    private void OnEnable()
    {
        if (file == null)
        {
                file = new WriteReadFile("save.data");
        }
        if (isLocal)
        {
            dataUpdated = false;
            allData = new List<Data>();
            allData = file.ReadAllData();
            dataUpdated = true;
        }
        else
        {
            if (onlineData != null)
            {
                onlineData.Join();
            }
                onlineData = new Thread(DataUpdate);
                onlineData.Start();
                dataUpdated = false;
        }
        DestroyAllChild();
        loading.SetActive(true);
    }
    void DataUpdate()
    {
        allData = new List<Data>();
        string connectionInfo =
String.Format("server={0};user={1};database={2};password={3}", "server", "user ",
"db", "lol");
            if (connection == null && MySqlConnetionRef == null)
            {
                connection = new MySqlConnection(connectionInfo);
                MySqlConnetionRef = connection;
            }
            if (MySqlConnetionRef != null && connection == null)
            {
                connection = MySqlConnetionRef;
            }
            using (connection)
            {
```

```
                try
                {
                    connection.Open();
                    string get = String.Format("SELECT * FROM `{0}` WHERE Level={1}
ORDER BY Score DESC", tableName, AllLevels);
                    using (var command = new MySqlCommand(get, connection))
                    {
                        using (var reader = command.ExecuteReader())
                        {
                            while (reader.Read())
                            {
                                Data data = new Data(reader.GetDateTime(1),
reader.GetString(2), reader.GetFloat(3));
                                allData.Add(data);
                            }
                        }
                    }
                    connection.Close();
                }
                catch (Exception ex)
                {
                    Debug.Log(ex.ToString());
                }
            }
            if (usingSteam)
            {
                string name = SteamFriends.GetPersonaName();
                CSteamID steamId = SteamUser.GetSteamID();
                using (connection)
                {
                    try
                    {
                        connection.Open();
                        string get = String.Format("SELECT * FROM `SteamScoreBoard`
WHERE Level={0} AND OwnerKey={1} ORDER BY Score DESC", AllLevels, steamId);
                        using (var command = new MySqlCommand(get, connection))
                        {

                            using (var reader = command.ExecuteReader())
                            {

                                while (reader.Read())
                                {
                                    if (reader.GetString(2) != name)
                                    {
                                        String updateUsername =
String.Format("UPDATE SteamScoreBoard SET Usernam={0} WHERE OwnerKey={1}", name,
steamId);
                                    }
                                }
                            }
                        }
                        connection.Close();
                    }
                    catch (Exception ex)
                    {
                        Debug.Log(ex.ToString());
                    }
                }
        }
        dataUpdated = true;
        onlineData.Abort();
    }
    void FillScoreBoard()
    {
```

```
        tempPosition = 0;
        for (int i = 0; i < allData.Count; i++)
        {
            if (i == 0)
            {
                RectTransform rect = gameObject.GetComponent<RectTransform>();
                GameObject List = Instantiate(prefab);
                rect.sizeDelta = firstDimmensionsRect;
                List.gameObject.SetActive(true);
                List.name = prefab.name + (i + 1).ToString();
                List.transform.SetParent(transform);
                List.GetComponent<RectTransform>().localPosition = new
Vector3(firstXCoordinate, firstYCoordinate, firstZCoordinate);
                List.GetComponent<RectTransform>().localScale = new Vector3(1, 1,
1);
                List.GetComponent<RectTransform>().localRotation =
Quaternion.Euler(0, 0, 0);
                tempPosition = firstYCoordinate;
                List.transform.Find("Place").GetComponent<TextMeshProUGUI>().text =
(i + 1).ToString();
                List.transform.Find("Date").GetComponent<TextMeshProUGUI>().text =
allData[i].DatePlay.ToString("yyyy/MM/dd");
                List.transform.Find("User").GetComponent<TextMeshProUGUI>().text =
allData[i].NickName;
                List.transform.Find("Score").GetComponent<TextMeshProUGUI>().text =
allData[i].Score.ToString();
            }
            if (i > 0)
            {
                RectTransform rect = gameObject.GetComponent<RectTransform>();
                rect.sizeDelta = new Vector2(rect.sizeDelta.x, rect.sizeDelta.y +
Math.Abs(offSet));
                GameObject List = Instantiate(prefab);
                List.gameObject.SetActive(true);
                List.name = prefab.name + (i + 1).ToString();
                List.transform.SetParent(transform);
                float yPosition = tempPosition + offSet;
                List.GetComponent<RectTransform>().localPosition = new
Vector3(firstXCoordinate, yPosition, firstZCoordinate);
                List.GetComponent<RectTransform>().localScale = new Vector3(1, 1,
1);
                List.GetComponent<RectTransform>().localRotation =
Quaternion.Euler(0, 0, 0);
                tempPosition = yPosition;
                List.transform.Find("Place").GetComponent<TextMeshProUGUI>().text =
(i + 1).ToString();
                List.transform.Find("Date").GetComponent<TextMeshProUGUI>().text =
allData[i].DatePlay.ToString("yyyy/MM/dd");
                List.transform.Find("User").GetComponent<TextMeshProUGUI>().text =
allData[i].NickName;
                List.transform.Find("Score").GetComponent<TextMeshProUGUI>().text =
allData[i].Score.ToString();
            }
        }
    }
    void Start()
    {
        usingSteam = SteamManager.Initialized;
        RectTransform rect = gameObject.GetComponent<RectTransform>();
        firstDimmensionsRect = rect.sizeDelta;
    }
    void Update()
    {
        if (GameData.End == true && oneTime && GameEnd.Inserted)
        {
```

```
            if (isLocal)
            {
                dataUpdated = false;
                allData = new List<Data>();
                allData = file.ReadAllData();
                dataUpdated = true;
            }
            else
            {
                if (onlineData != null)
                {
                    onlineData.Join();
                }
                onlineData = new Thread(DataUpdate);
                onlineData.Start();
                dataUpdated = false;
            }
            loading.SetActive(true);
            DestroyAllChild();
            oneTime = false;
        }
        if (dataUpdated)
        {
            FillScoreBoard();
            loading.SetActive(false);
            dataUpdated = false;
        }
    }
    void DestroyAllChild()
    {
        int i = 0;
        foreach (Transform objects in transform)
        {
            if (i > 1)
            {
                Destroy(objects.gameObject);
            }
            i++;
        }
    }
}
```

Table 33 ScoreBoardDatabase.cs script component

Task #33. Add "Game over" condition (once the game ends you should display a high score and reload/main menu buttons (Eligijus)

As mentioned before, when the timer reaches 00:00, the player loses and gets teleported into the Lose scene.

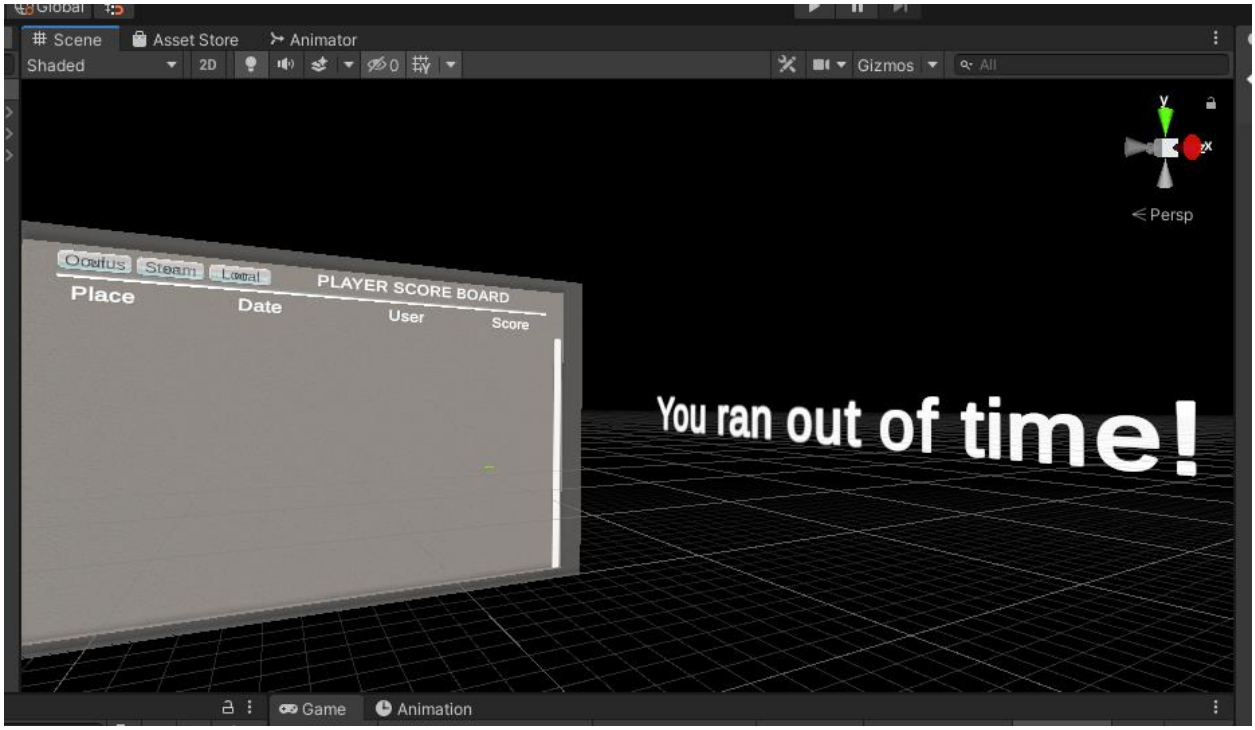

Figure 94 Lose scene with leaderboard

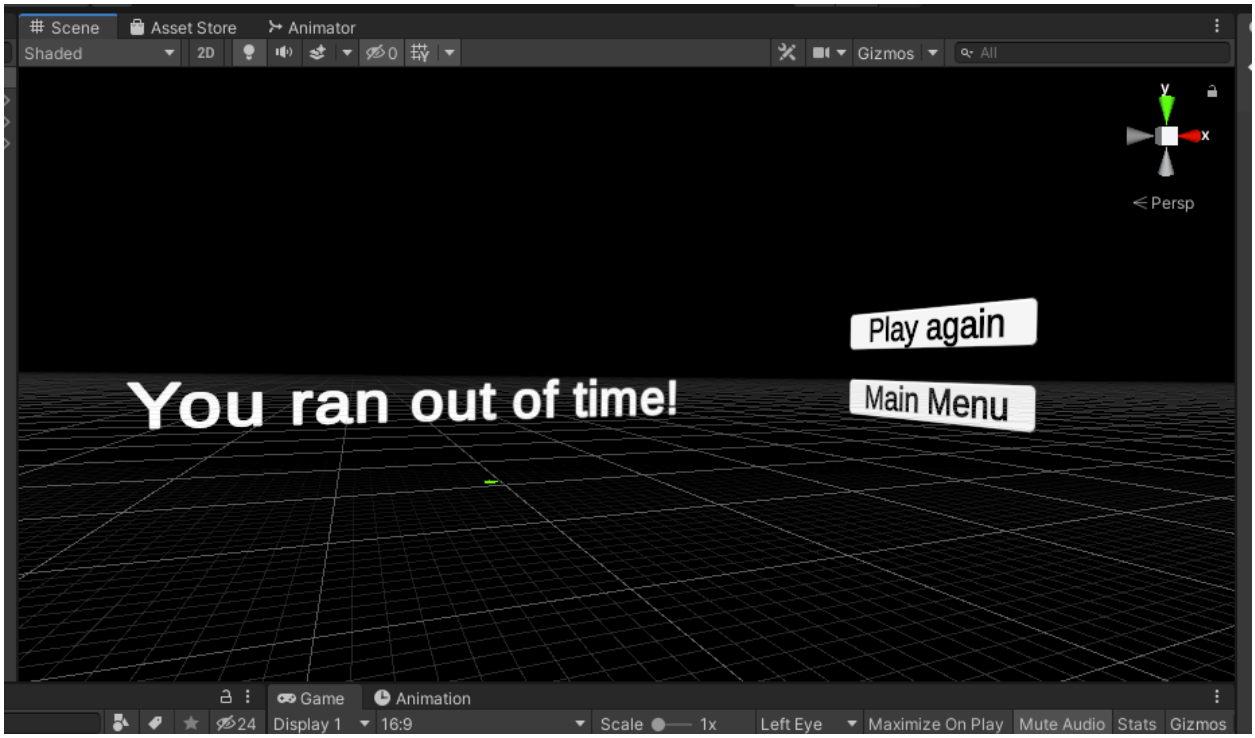

Figure 95 Lose scene menu

```
public class Teleports : MonoBehaviour
{
    TimeLeft timeleft;
    [SerializeField] bool loadPreviuos = false;
    [SerializeField] int loseSceneIndex = 0;
    bool saveFile = false;

    private void Start()
    {
        timeleft = GetComponent<TimeLeft>();
    }

    // Update is called once per frame
    void Update()
    {
        if (GameData.End == true && GameData.Victory == false)
        {
            if (saveFile == false)
            {
                GlobalData.WriteToFile();
                           GlobalData.ResetStatic();
                           BoundryController.ResetCount();
                           saveFile = true;
            }
            GameData.SetPreviuosScene(SceneManager.GetActiveScene().buildIndex);
            loseScene();
            GameData.Cear();
        }
    }

    void loseScene()
    {
        SceneManager.LoadScene(loseSceneIndex);
    }

    void restartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    void LoadPreviuos()
    {
        SceneManager.LoadScene(GameData.PreviuosSceneId);
    }
}
```

Table 34 Teleports.cs script component

Task #34. Add "Post Processing" (Blurring, blood screen, etc.) (Airidas)

A slight bloom, color grading and blueish eye adaptation added to enhance the light blue colors of the game scene.

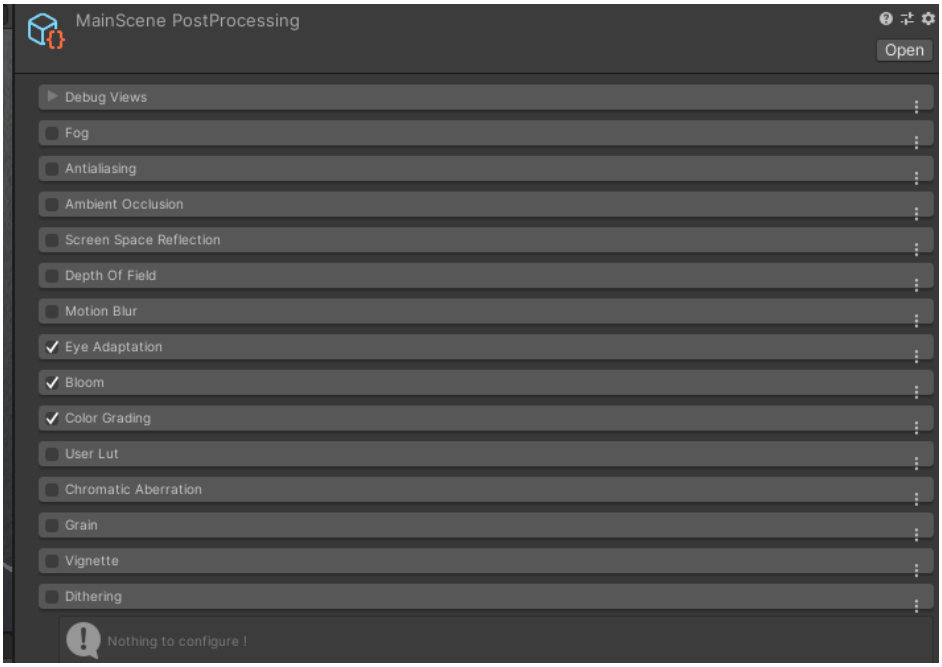

Figure 96 Main scene post processing

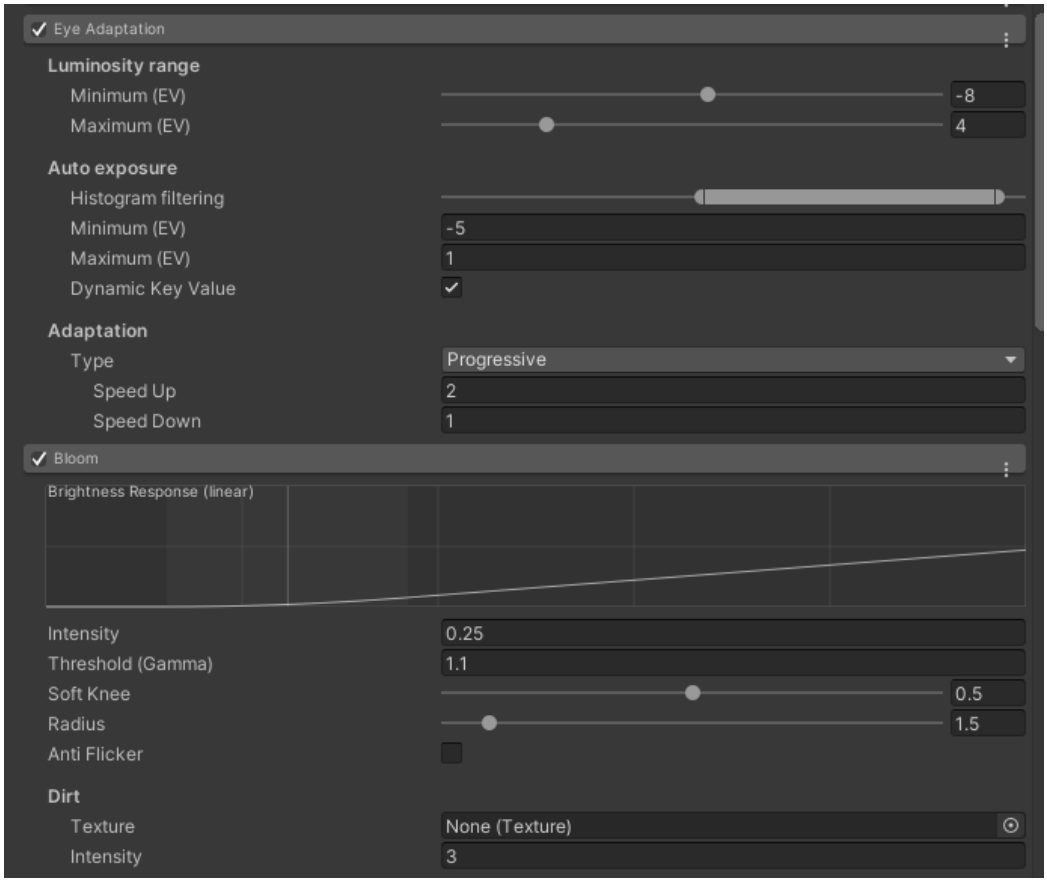

Figure 97 Main scene post processing Eye Adaptation and Bloom

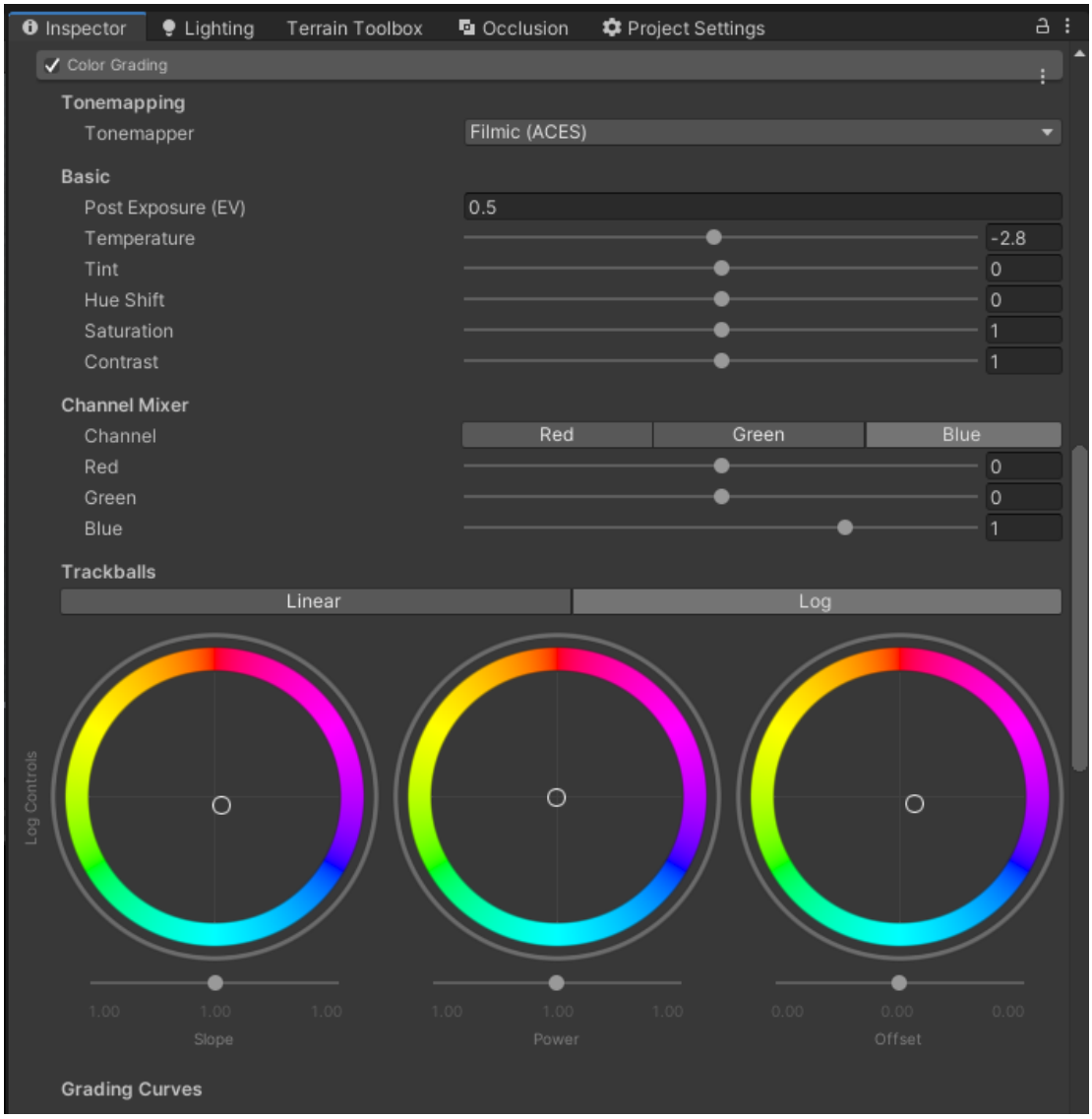

Figure 98 Main scene post processing Color Grading

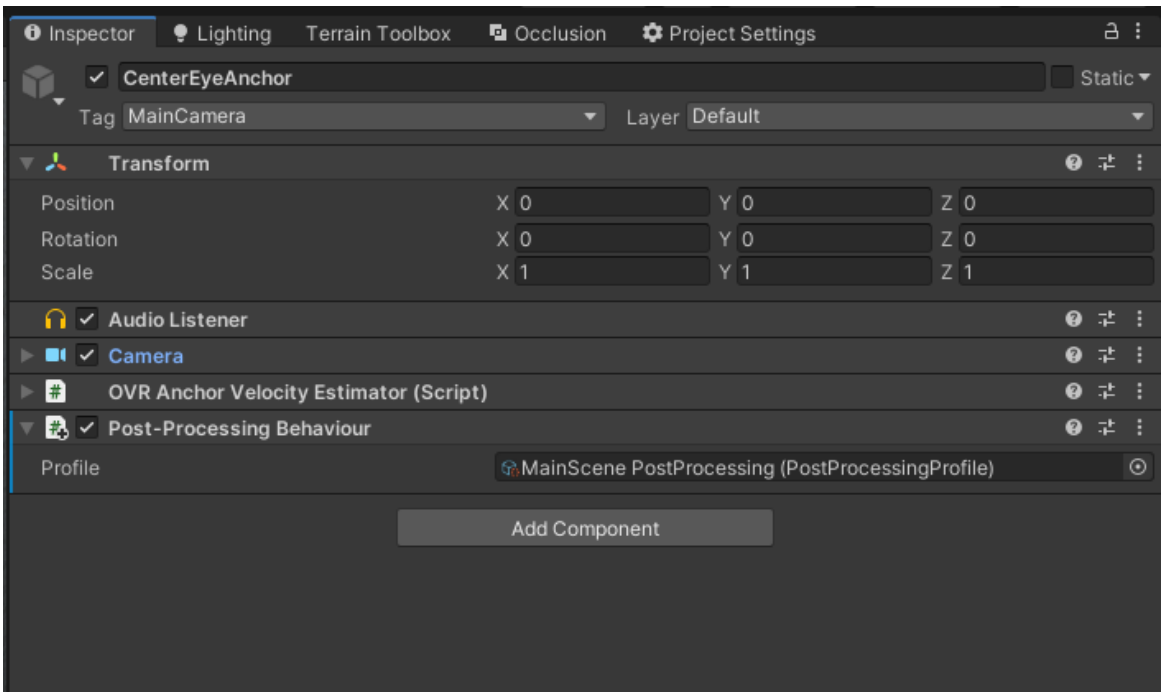

Figure 99 Applied Main scene post processing

Task #35. Add Interactive sounds (Sound of step, shoot, die, scream, etc. - minimum 10 different) and music (1 for menu + minimum 3 for in-game) (Airidas)

Created an empty Game Object, gave it a script with DontDestroyOnLoad behavior, so that the music could be played across the levels seamlessly.

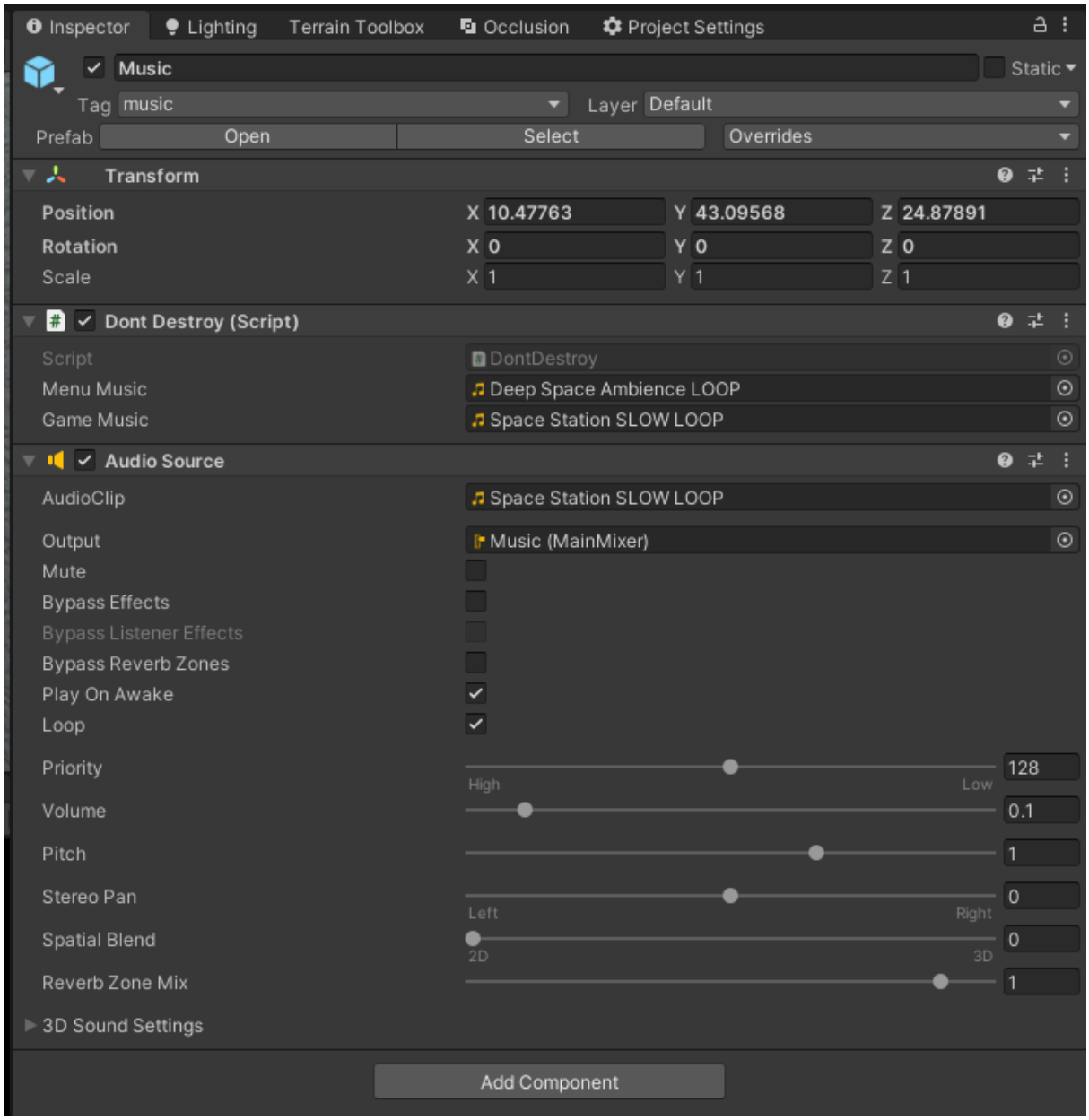

Figure 100 Background music Game Object

```
public class DontDestroy : MonoBehaviour
{
    private static DontDestroy instance;
    private AudioSource aud;

    public AudioClip menuMusic;
    public AudioClip gameMusic;

    private void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    private void Awake()
    {
        if (instance != null && instance != this)
        {
            Destroy(this.gameObject);
        }
        else
            instance = this;

        DontDestroyOnLoad(gameObject);
    }

    void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (scene.name == "MainMenu")
        {
            if (aud == null)
                return;
            aud.Stop();
            aud.clip = menuMusic;
            aud.Play();
        }
        else if (scene.name == "MainGame")
        {
            if (aud == null)
                return;
            aud.Stop();
            aud.clip = gameMusic;
            aud.Play();
        }
    }

}
```

Table 35 DontDestroy.cs script component

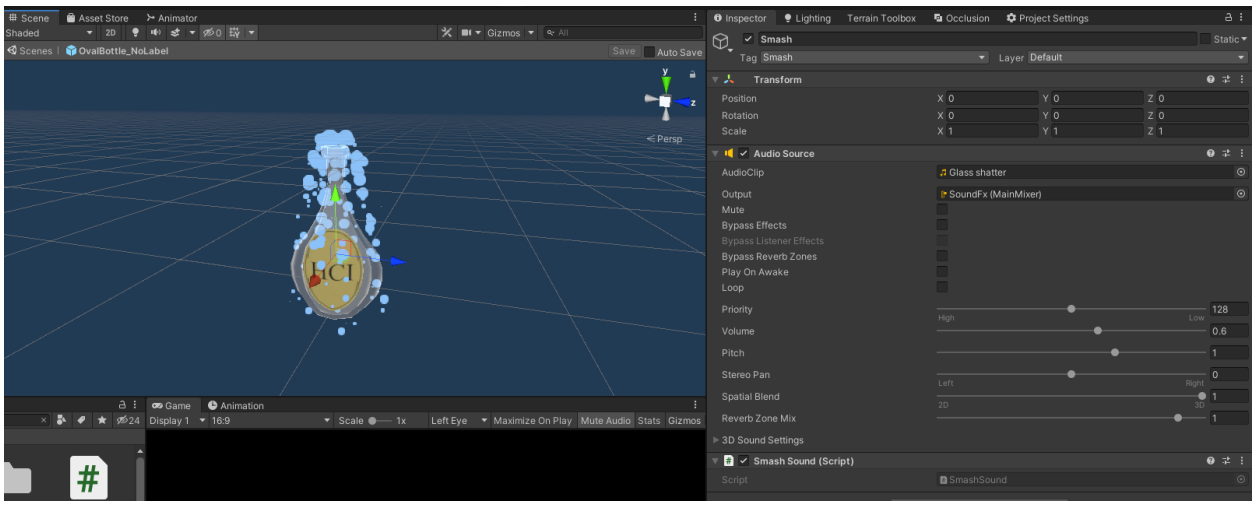

Figure 101 Smash sound effect Game Object

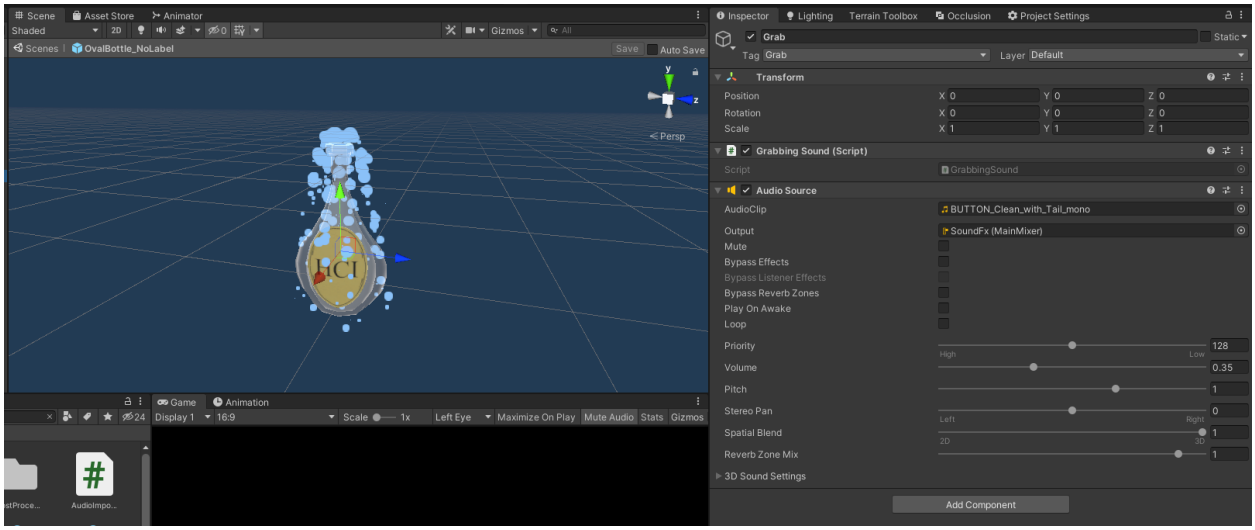

Figure 102 Grabbing sound effect Game Object

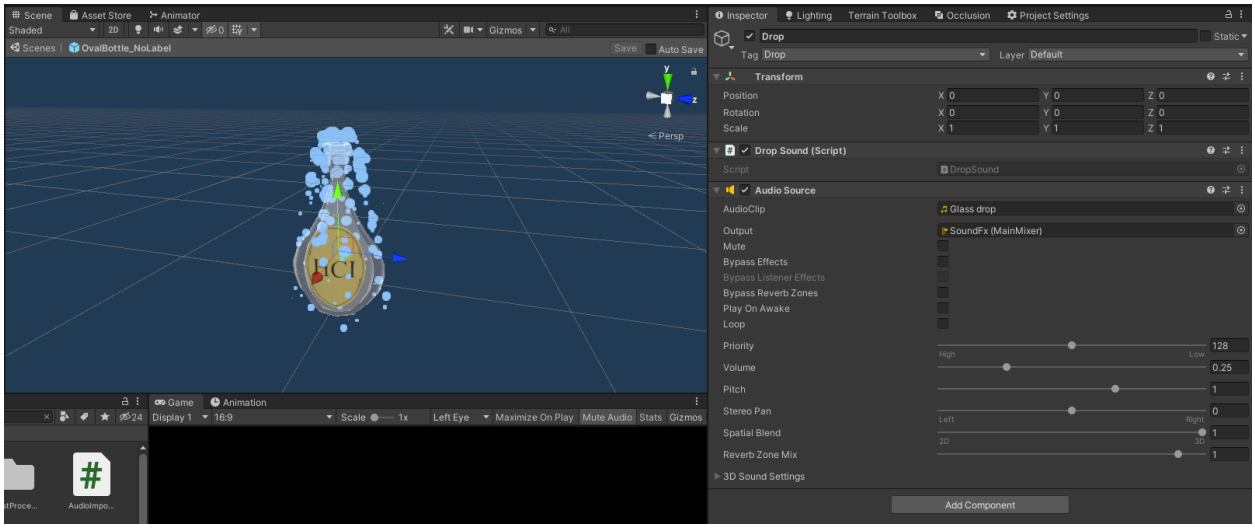

Figure 103 Dropping sound effect Game Object

```
public class DropSound : MonoBehaviour
{
    int cnt = 0;
    AudioSource aud;
    // Start is called before the first frame update
    void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    public void PlaySound()
    {
        if (!aud.isPlaying)
        {
            aud.Play();
        }
    }
}
```

Table 36 DropSound.cs script component

```
public class GrabbingSound : MonoBehaviour
{
    bool isGrabbed = false;
    bool play = true;
    AudioSource sound;
    // Start is called before the first frame update
    void Start()
    {
        sound = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {

        if (isGrabbed && play)
        {
            sound.Play();
            play = false;
        }
        else if (!isGrabbed)
        {
            play = true;
        }
    }

    public void Grab()
    {
        isGrabbed = true;
    }
    public void UnGrab()
    {
        isGrabbed = false;
    }
}
```

Table 37 GrabbingSound.cs script component

```
public class SmashSound : MonoBehaviour
{
    BottleSmash smash;
    bool smashed = false;
    AudioSource aud;
    // Start is called before the first frame update
    void Start()
    {
        smash = GetComponentInParent<BottleSmash>();
        aud = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {
        if (smash.broken && !smashed)
        {
            aud.Play();
            smashed = true;
        }
    }
}
```

Table 38 SmashSound.cs script component

Task #36. Add Interactive sounds (Sound of step, shoot, die, scream, etc. - minimum 10 different) and music (1 for menu + minimum 3 for in-game) (Eligijus)

Created an empty Game Object, gave it a script with DontDestroyOnLoad behavior, so that the music could be played across the levels seamlessly.

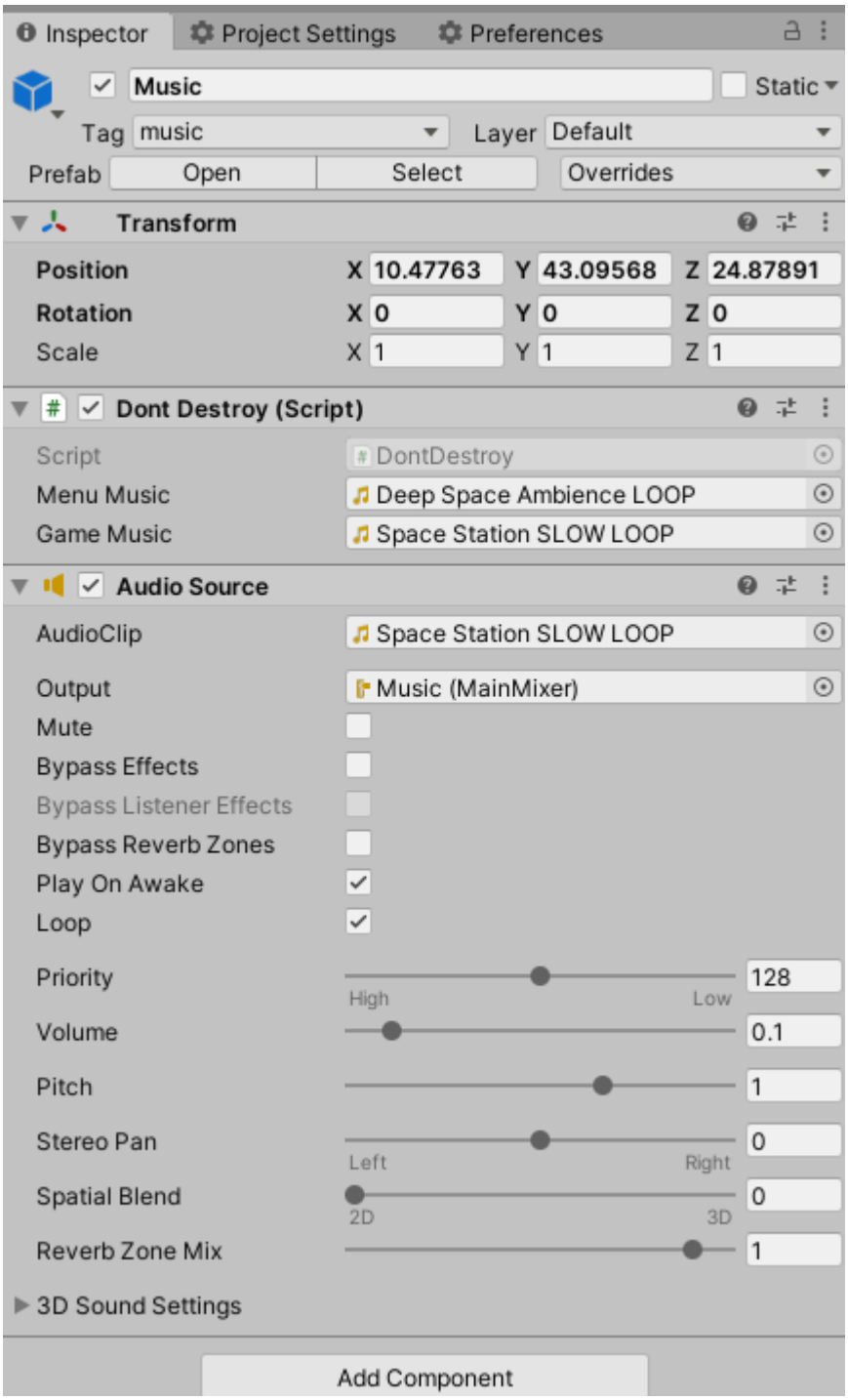

Figure 104 Background music Game Object

```
public class DontDestroy : MonoBehaviour
{
    private static DontDestroy instance;
    private AudioSource aud;

    public AudioClip menuMusic;
    public AudioClip gameMusic;

    private void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    private void Awake()
    {
        if (instance != null && instance != this)
        {
            Destroy(this.gameObject);
        }
        else
            instance = this;

        DontDestroyOnLoad(gameObject);
    }

    void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (scene.name == "MainMenu")
        {
            if (aud == null)
                return;
            aud.Stop();
            aud.clip = menuMusic;
            aud.Play();
        }
        else if (scene.name == "MainGame")
        {
            if (aud == null)
                return;
            aud.Stop();
            aud.clip = gameMusic;
            aud.Play();
        }
    }

}
```

Table 39 DontDestroy.cs script component

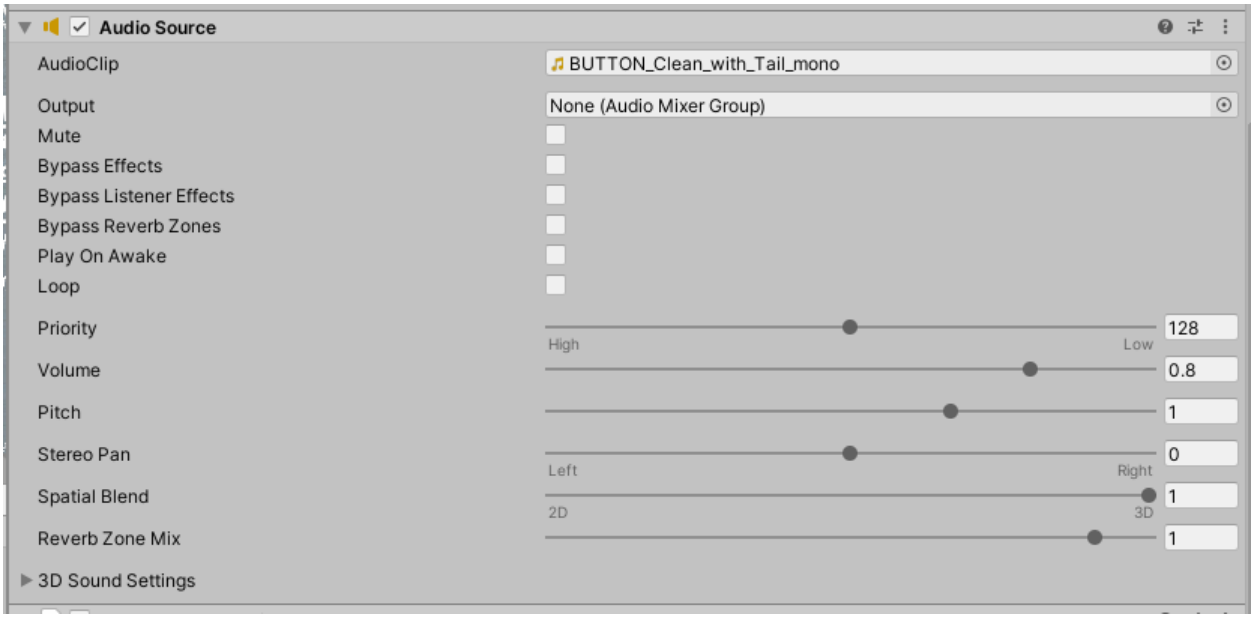

Figure 105 Button Sound

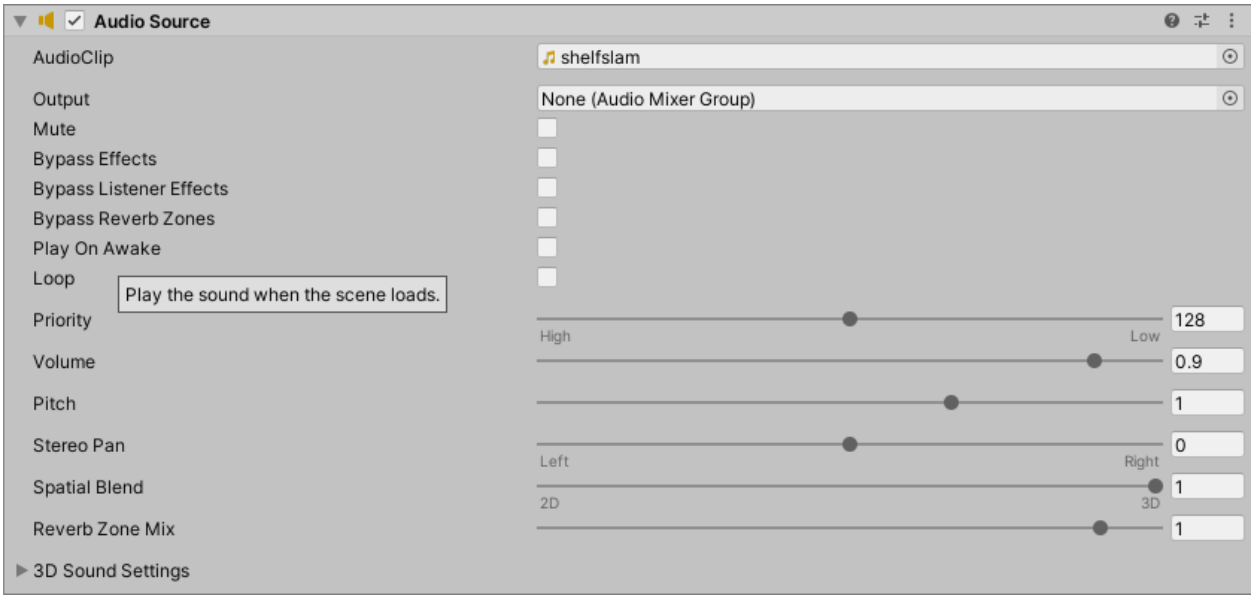

Figure 106 Shelf close sound

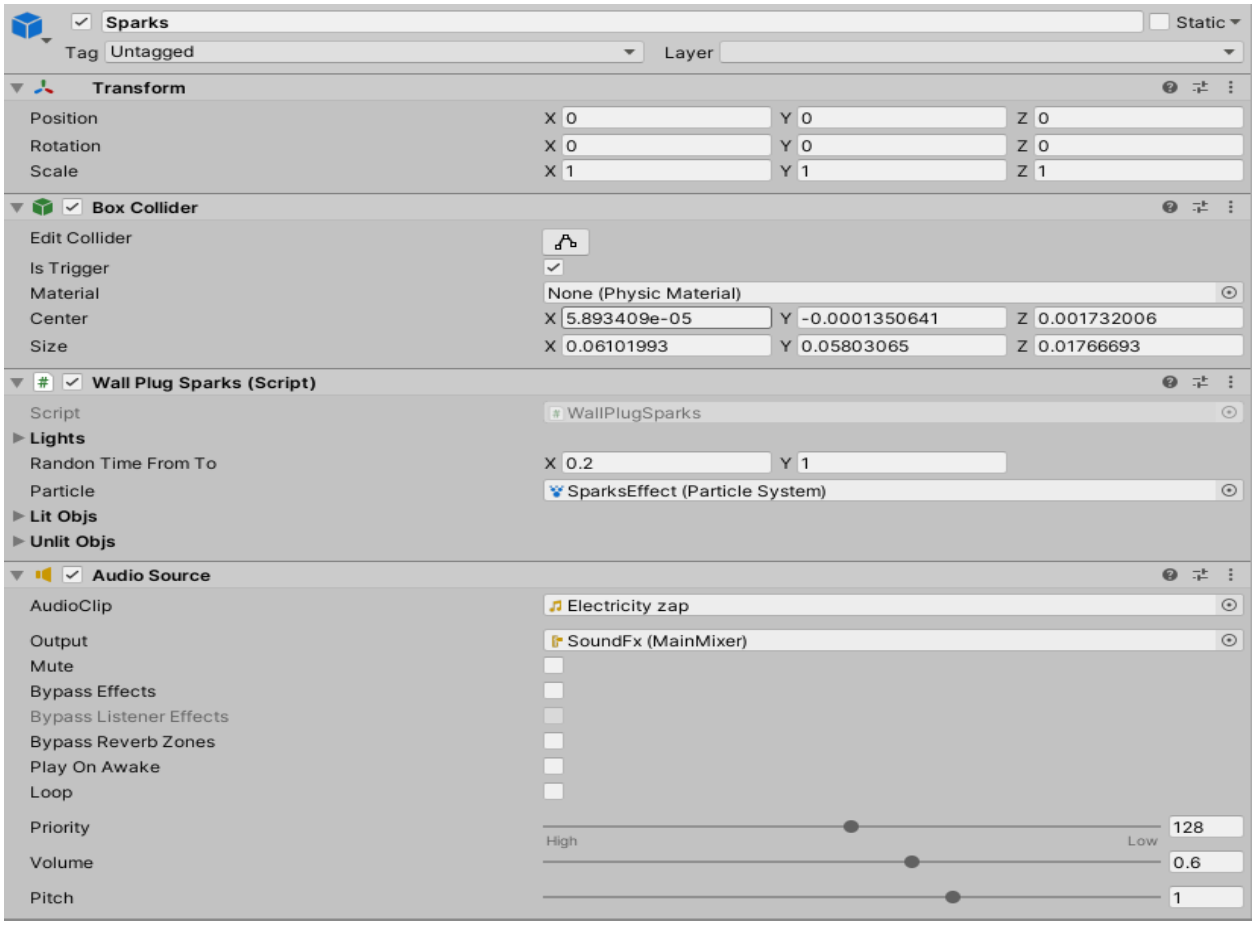

Figure 107 Spark Sound

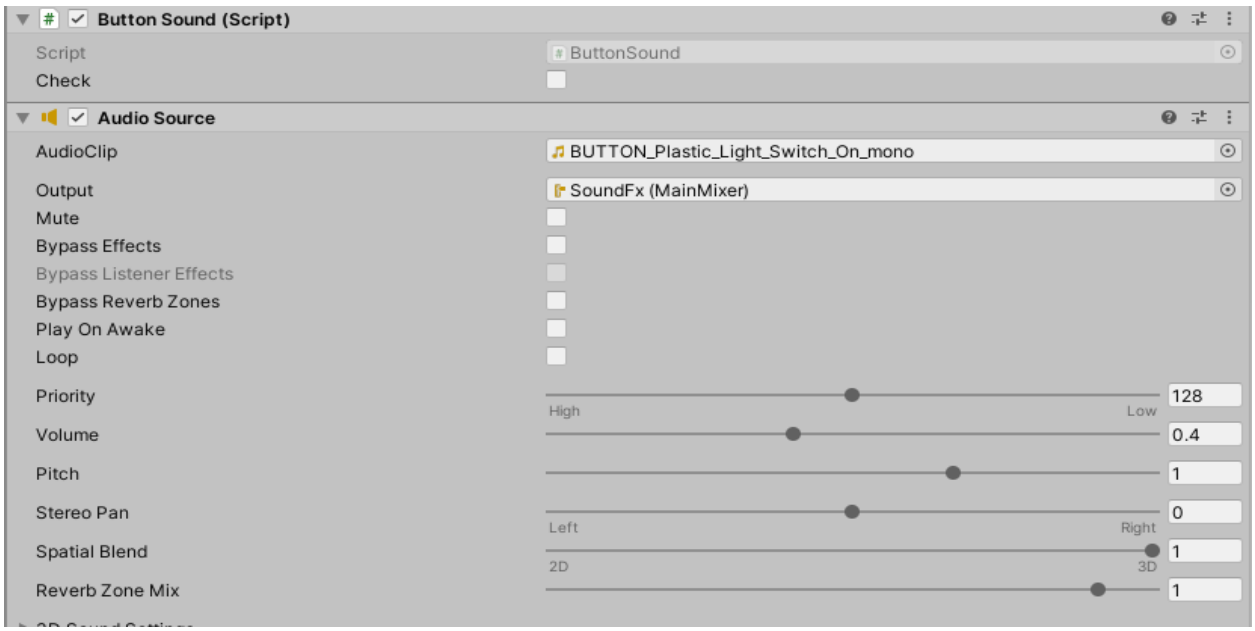

Figure 108 Light switch on sound

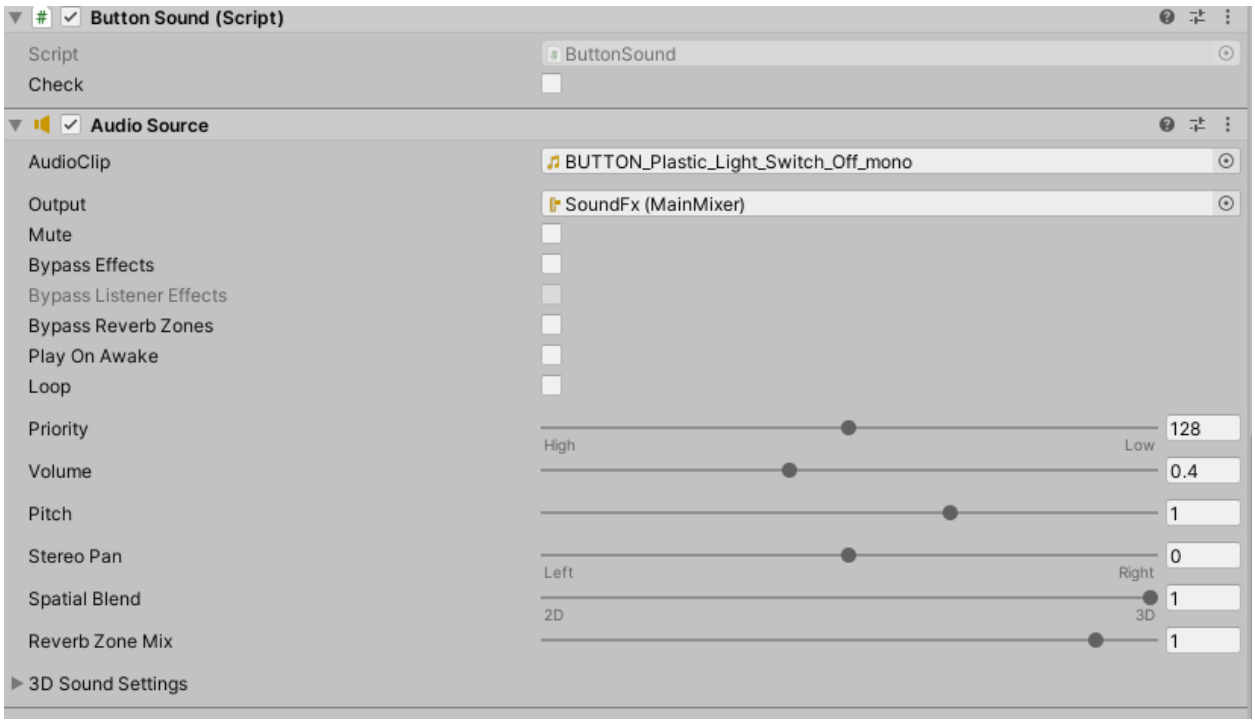

Figure 109 Light switch off sound

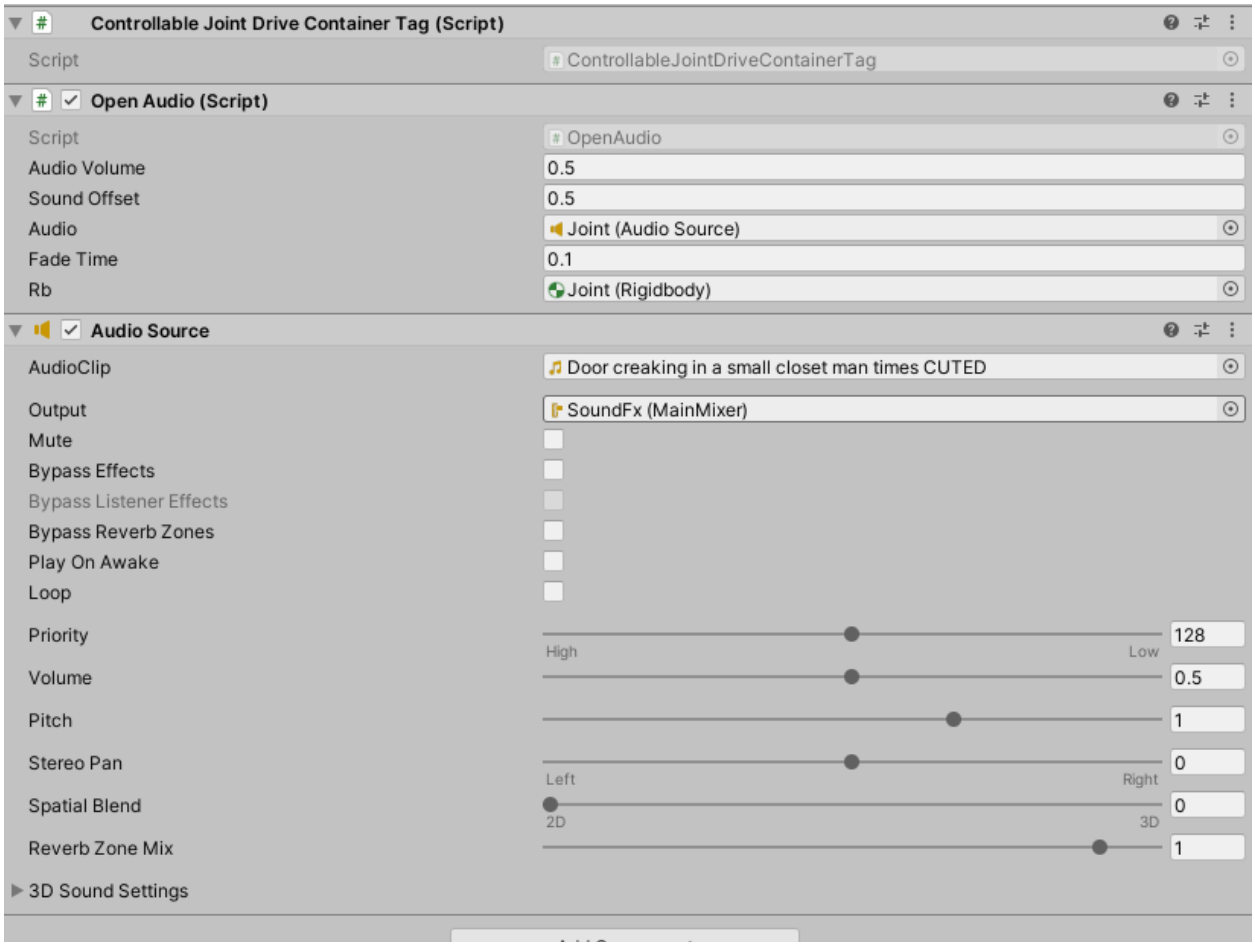

Figure 110 Door creaking sound

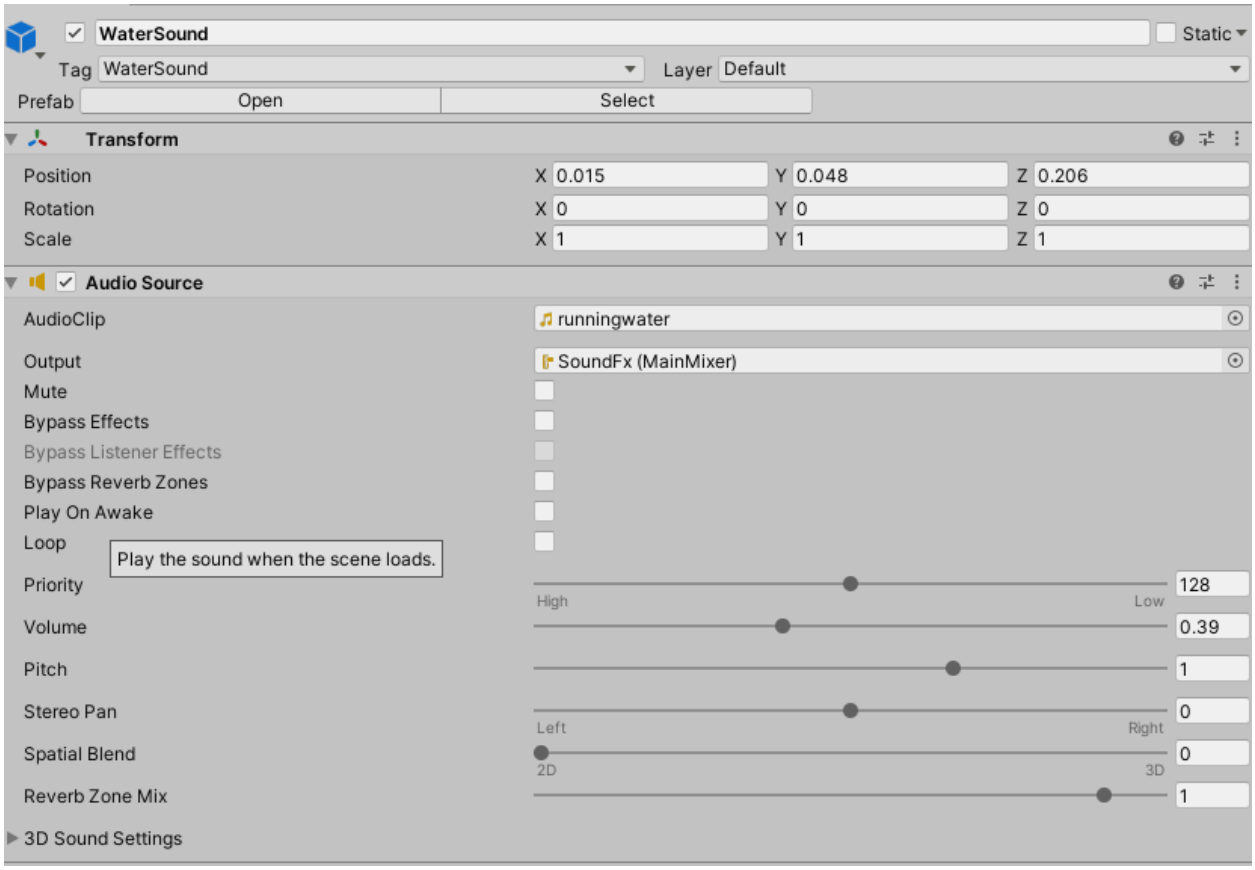

Figure 111 Pouring water sound

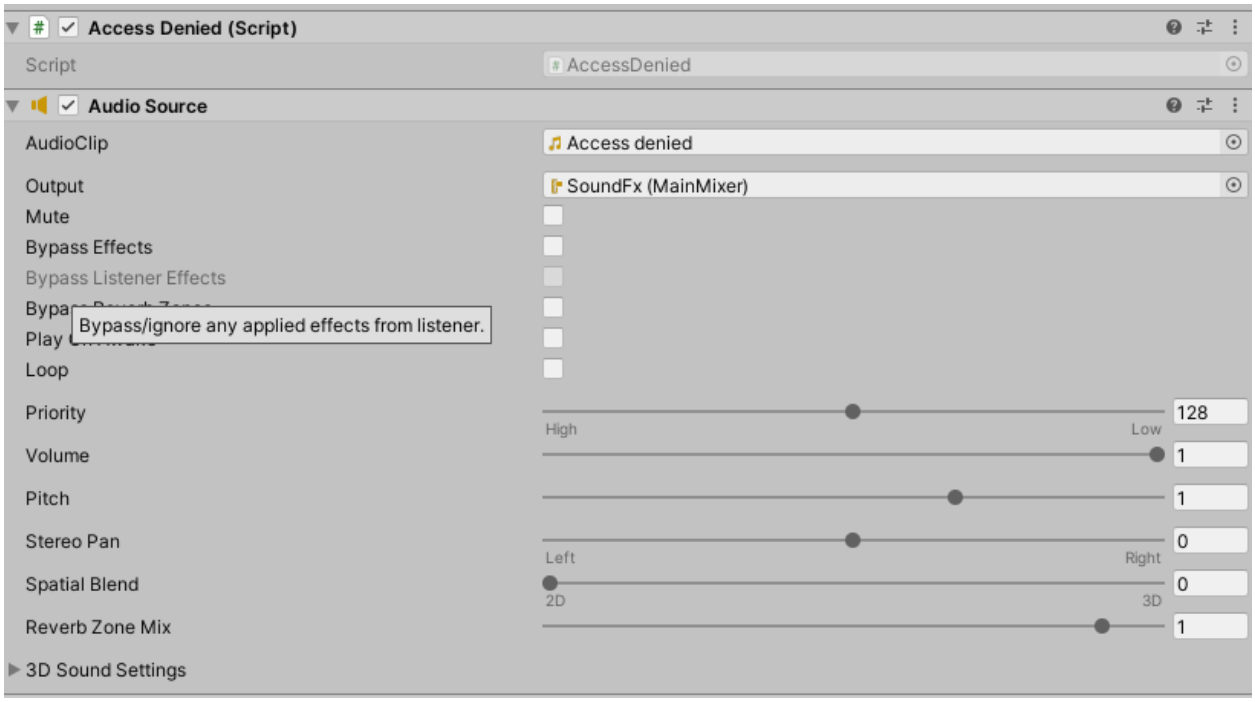

Figure 112 Access denied sound

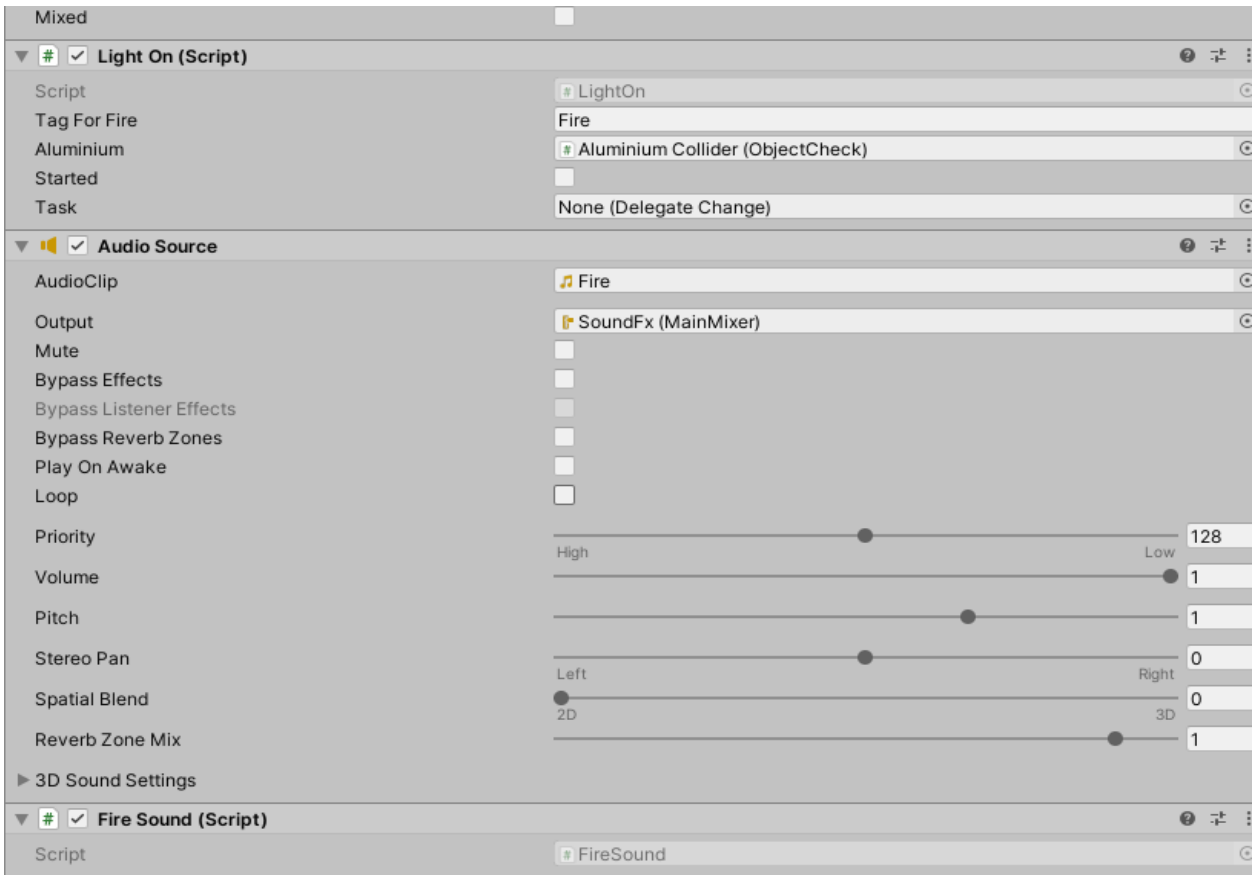

Figure 113 Experiment fire sound

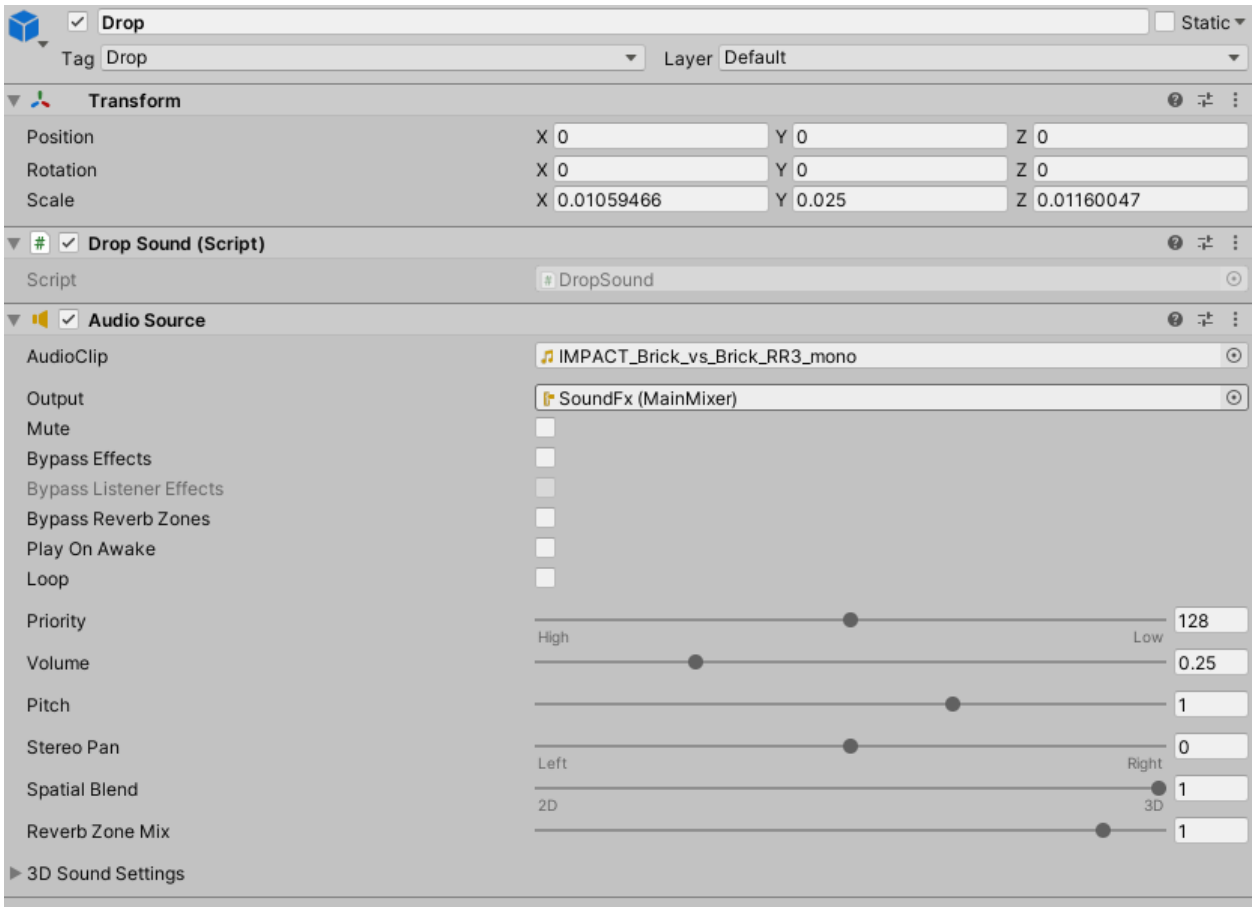

Figure 114 Inpact sound for aluminium

```
public class ButtonSound : MonoBehaviour
{
    AudioSource audio;
    //bool check = false;
    public bool check = false;

    void Start()
    {
        audio = GetComponent<AudioSource>();
    }
    // Update is called once per frame
    void Update()
    {
        if (check)
        {
            audio.Play();
            check = false;
        }
    }

    public void EnableSound()
    {
        check = true;
    }

}
```

Table 40 ButtonSounc.cs component

```
public class ShelfSound : MonoBehaviour
{
    [SerializeField] float audioVolume = 0.35f;

    float firstAngle;
    float fisrstAngleTracking;
    Vector3 angle;
    bool firstAngleCheck;
    bool sound;
    float timer;
    float speeds;
    bool timerbool;
    bool oneTime;
    [SerializeField] AudioSource audio;
    [SerializeField] Rigidbody rg;
    [SerializeField] Transform tf;
    [SerializeField] float speed = 0.1f;
    [SerializeField] bool startFrom360 = false;
    //// Start is called before the first frame update
    void Start()
    {

        StartCoroutine(muteSound());
        sound = false;
        timerbool = true;
        oneTime = false;
        timer = 0.4f;
        speeds = rg.velocity.magnitude;
        firstAngleCheck = true;

    }

    //// Update is called once per frame
    void Update()
    {
        angle = tf.localEulerAngles;
        speeds = rg.velocity.magnitude;
```

```
        //timer -= Time.deltaTime*2;
        //if (timer < 0)
        //{
        //Debug.Log(tf.localEulerAngles.y);
        if (!startFrom360)
        {
            if (speeds > speed && speeds < 1 && (tf.localEulerAngles.y <= 0.01 ||
tf.localEulerAngles.y > 350) && oneTime == false)
            {
                timer = 0.4f;
                timerbool = false;
                sound = true;
                oneTime = true;
            }
            else if (speeds > 0.001 && speeds < 1 && (tf.localEulerAngles.y <= 0.01 ||
tf.localEulerAngles.y > 350) && oneTime == false)
            {
                oneTime = true;
            }
            else if (speeds > speed && tf.localEulerAngles.y > 0.1 &&
(tf.localEulerAngles.y < 200) && oneTime == true)
            {
                timer = 0.4f;
                timerbool = true;
                sound = false;
                oneTime = false;
            }
            //}
            if (sound == true && !audio.isPlaying && timerbool == false)
            {
                audio.Play();
                sound = false;
            }
        }
        else {
            if (speeds > speed && speeds < 1 && (tf.localEulerAngles.y >= 0.01 &&
tf.localEulerAngles.y <= 260) && oneTime == false)
            {
                timer = 0.4f;
                timerbool = false;
                sound = true;
                oneTime = true;
            }
            else if (speeds > 0.001 && speeds < 1 && (tf.localEulerAngles.y >= 0.01 &&
tf.localEulerAngles.y <= 260) && oneTime == false)
            {
                oneTime = true;
            }
            else if (speeds > speed && tf.localEulerAngles.y > 260 &&
(tf.localEulerAngles.y < 360) && oneTime == true)
            {
                timer = 0.4f;
                timerbool = true;
                sound = false;
                oneTime = false;
            }
            //}
            if (sound == true && !audio.isPlaying && timerbool == false)
            {
                audio.Play();
                sound = false;
            }
        }
    }
```

```
    IEnumerator muteSound()
    {
        audio.volume = 0;
        yield return new WaitForSeconds(5);
        audio.volume = audioVolume;
    }

}
```

Table 41 ShelfSound.cs componenet

```
public class WallPlugSparks : MonoBehaviour
{
    //TODO fix animation component disabling/enabling
    [SerializeField] List<GameObject> lights;
    [SerializeField] Vector2 RandonTimeFromTo = Vector2.zero;
    [SerializeField] ParticleSystem particle;
    public GameObject[] litObjs;
    public GameObject[] unlitObjs;
    List<float> timeToSpark;
    List<float> timeCount;
    List<bool> checkSparkle;
    bool spark = false;
    int lightBlinked = 0;
    AudioSource aud;
    bool isTouched = false;
    private void Start()
    {
        timeToSpark = new List<float>();
        timeCount = new List<float>();
        checkSparkle = new List<bool>();
        for (int i = 0; i < lights.Count; i++)
        {
            timeToSpark.Add(Random.Range(RandonTimeFromTo.x, RandonTimeFromTo.y));
            checkSparkle.Add(false);
            timeCount.Add(0);
        }

        aud = GetComponent<AudioSource>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag != "GameController" && other.name !=
"[VRTK][AUTOGEN][Controller][NearTouch][CollidersContainer]" && other != null)
        {
            particle.Play();
            aud.Play();
            spark = true;
            for (int i = 0; i < timeToSpark.Count; i++)
            {
                timeToSpark[i] = Random.Range(RandonTimeFromTo.x, RandonTimeFromTo.y);
            }
        }
    }

    private void Update()
    {
        if (isTouched && spark == false)
        {
            particle.Play();
            aud.Play();
            spark = true;
            for (int i = 0; i < timeToSpark.Count; i++)
            {
                timeToSpark[i] = Random.Range(RandonTimeFromTo.x, RandonTimeFromTo.y);
```

```
            }
        }
        if (spark)
        {
            for (int i = 0; i < timeToSpark.Count; i++)
            {
                if (timeToSpark[i] >= timeCount[i])
                {
                    litObjs[i].SetActive(false);
                    unlitObjs[i].SetActive(true);
                    lights[i].GetComponent<Light>().enabled = false;
                    timeCount[i] += Time.deltaTime;
                }
                else if (timeToSpark[i] < timeCount[i] && !checkSparkle[i])
                {
                    lightBlinked++;
                    litObjs[i].SetActive(true);
                    unlitObjs[i].SetActive(false);
                    lights[i].GetComponent<Light>().enabled = true;
                    checkSparkle[i] = true;
                }
            }
            if (lightBlinked >= timeToSpark.Count)
            {
                for (int i = 0; i < timeToSpark.Count; i++)
                {
                    if (lightBlinked == timeToSpark.Count)
                    {
                        checkSparkle[i] = false;
                        timeCount[i] = 0;
                    }
                }
                spark = false;
                lightBlinked = 0;
            }

        }
    }

    public void Touch()
    {
        isTouched = true;
    }
    public void Relise()
    {
        isTouched = false;
    }
}
```

Table 42 WallPlugSpark.cs Component

```
public class OpenAudio : MonoBehaviour
{
    [SerializeField] float audioVolume = 0.09f;
    [SerializeField] float soundOffset = 0.5f;
    [SerializeField] AudioSource audio;
    public float fadeTime = 1;
    [SerializeField]Rigidbody rb;
    float timeCnt = 0;
    float stopedAt = 0;
    bool stopped = true;
    float speed = 0;
    Vector3 speedCalc = Vector3.zero;
    float startXPos = 0;
    Vector3 tempx = Vector3.zero;
    float chunksCnt = 0;
```

```
    float chunkSize = 0;
    // Start is called before the first frame update

    private void OnEnable()
    {
        if (rb == null)
        {
            rb = gameObject.GetComponent<Rigidbody>();
        }
    }
    void Start()
    {
        chunksCnt = fadeTime / Time.deltaTime;
        chunkSize = audioVolume / chunksCnt;
        startXPos = rb.gameObject.transform.localPosition.x;
    }

    // Update is called once per frame
    void Update()
    {
        if (tempx != rb.gameObject.transform.localPosition)
        {
            speedCalc = rb.gameObject.transform.localPosition - tempx;
            speedCalc /= Time.deltaTime;
            tempx = rb.gameObject.transform.localPosition;
            speed = speedCalc.magnitude;
        }
        else if (tempx == rb.gameObject.transform.localPosition)
        {
            speed = Vector3.zero.magnitude;
        }


        //Debug.Log(speed);

        if (speed > 0.04 && startXPos > rb.gameObject.transform.localPosition.x &&
stopped)
        {

            audio.volume = audioVolume;
            audio.time = stopedAt;
            audio.Play();
            stopped = false;
            timeCnt = 0;
        }
        else if (speed > 0.04 && startXPos > rb.gameObject.transform.localPosition.x &&
!stopped)
        {
            //Debug.Log("HaHa");
            if (audio.isPlaying && audio.time > audio.clip.length - soundOffset)
            {
                audio.volume = audioVolume;
                audio.time = soundOffset;
                audio.Play();
            }
            else if (!audio.isPlaying)
            {
                audio.volume = audioVolume;
                audio.time = soundOffset;
                audio.Play();
            }
            timeCnt = 0;
        }
        else
        {
```

```
            timeCnt += Time.deltaTime;
            audio.volume -= chunkSize;
            if (timeCnt >= fadeTime && !stopped)
            {
                stopedAt = audio.time;
                audio.Stop();
                stopped = true;
            }

        }
    }
}
```

Table 43 Door open component

```
public class FallingWater : MonoBehaviour
{
    public bool Active = false;
    public GameObject particle;
    [SerializeField] AudioSource audio;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (gameObject.transform.rotation.eulerAngles.z >= 340)
        {
            particle.GetComponent<ParticleSystem>().Play();
            if (!audio.isPlaying)
            {
                audio.Play();
            }
            Active = true;
        }
        else
        {
            particle.GetComponent<ParticleSystem>().Stop();
            audio.Stop();
            Active = false;
        }
    }
}
```

Table 44 Falling water component

```
public class AccessDenied : MonoBehaviour
{
    AudioSource audio;
    bool started = false;

    private void Start()
    {
        audio = GetComponent<AudioSource>();
    }

    public void StartSound()
    {
        if (started && !audio.isPlaying)
        {
            started = false;
        }
```

```
        if (!started)
        {
            audio.Play();
            started = true;
        }

    }
}
```

Table 45 Access denied component

```
public class FireSound : MonoBehaviour
{
    AudioSource audio;
    bool Fire = true;
    LightOn fireScript;
    // Start is called before the first frame update
    void Start()
    {
        audio = GetComponent<AudioSource>();
        fireScript = GetComponent<LightOn>();
    }

    // Update is called once per frame
    void Update()
    {
        if (fireScript.Started && Fire)
        {
            audio.Play();
            Fire = false;
        }
        else if (!fireScript.Started && !Fire)
        {
            Fire = true;
        }
    }
}
```

Table 46 Fire sound component

```
public class DropSound : MonoBehaviour
{
    int cnt = 0;
    AudioSource aud;
    // Start is called before the first frame update
    void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    public void PlaySound()
    {
        if (!aud.isPlaying)
        {
            aud.Play();
        }
    }
}
```

Table 47 Drop sound script component

Game scenario

Main menu

First of all, the player spawns in main menu scene, where he has the possibility to either start the main game, go to a tutorial scene or exit the game.

Figure 115 Main menu scene

Tutorial level

The player is guided through every step in the tutorial level with clipboards that are filled with text. Each clipboard contains important information about gameplay mechanics, tips on how to progress and so on.

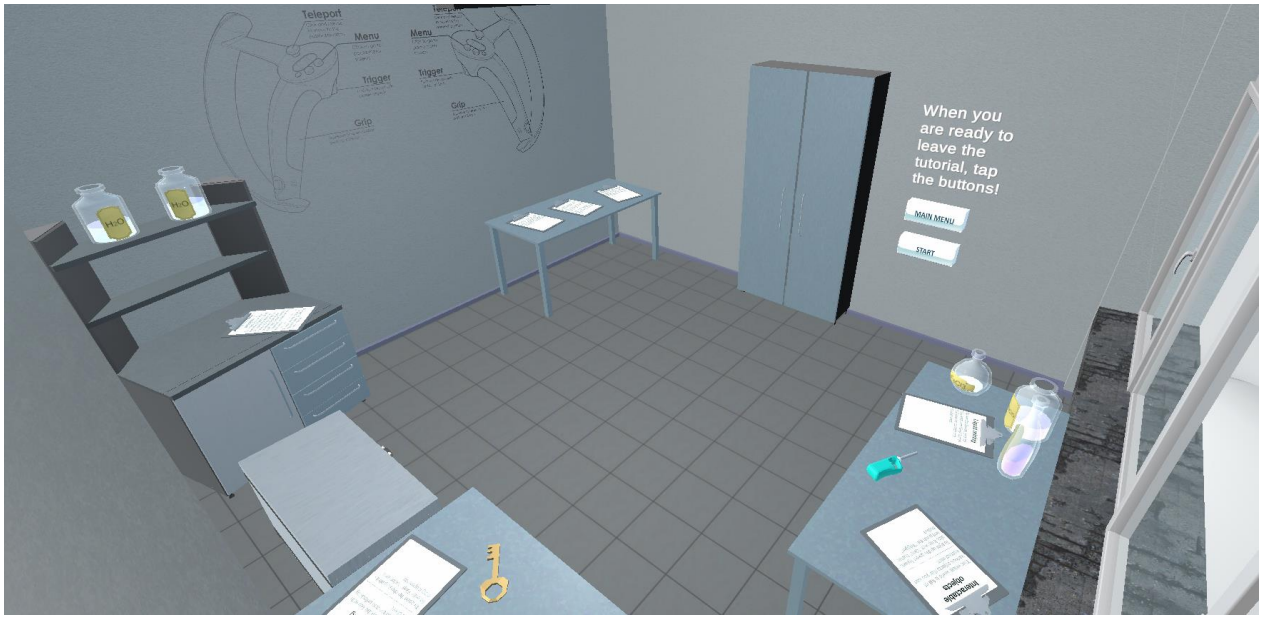

Figure 116 Tutorial level

First level

Goal of the first level is to find a hidden key that unlocks the exit door.

Figure 117 First level

The key is hidden in a flask that is always randomly spawned between several locations, to make the gameplay less repetitive.

Figure 118 Key in a flask

The player has to break the flask by throwing it into a wall or any other static object.

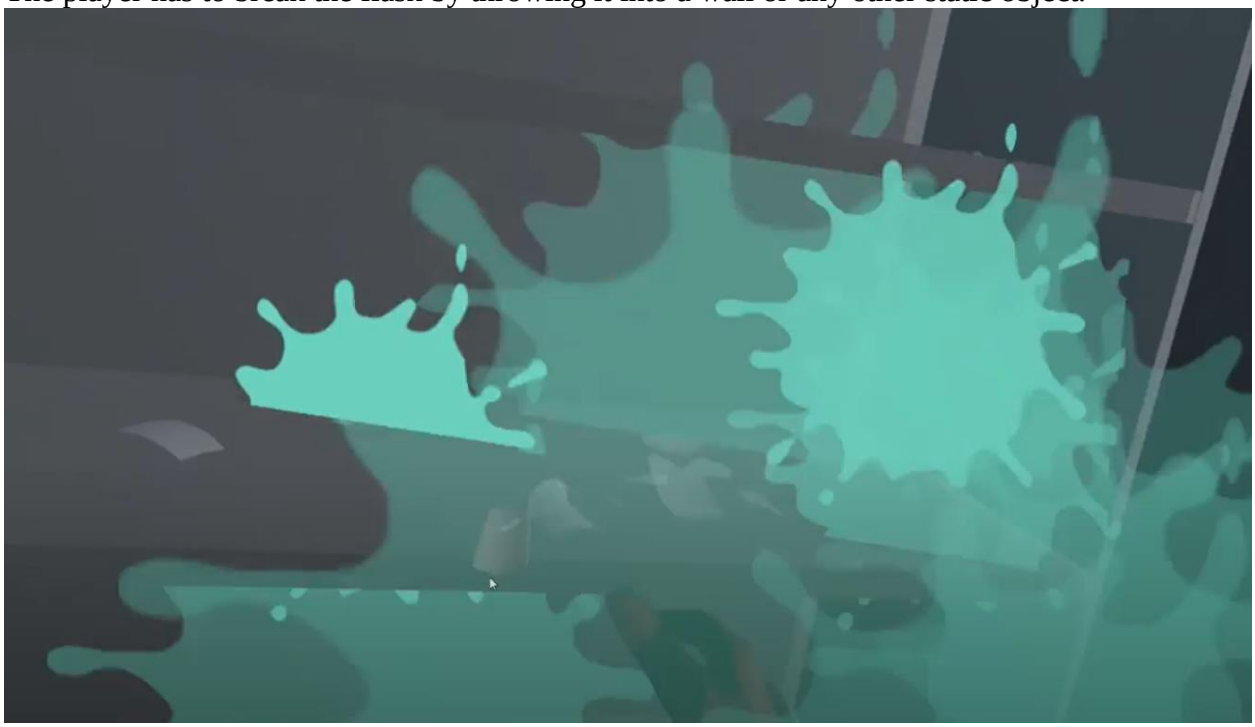

Figure 119 Broken flask in a key

When the flask is shattered, the key falls out.

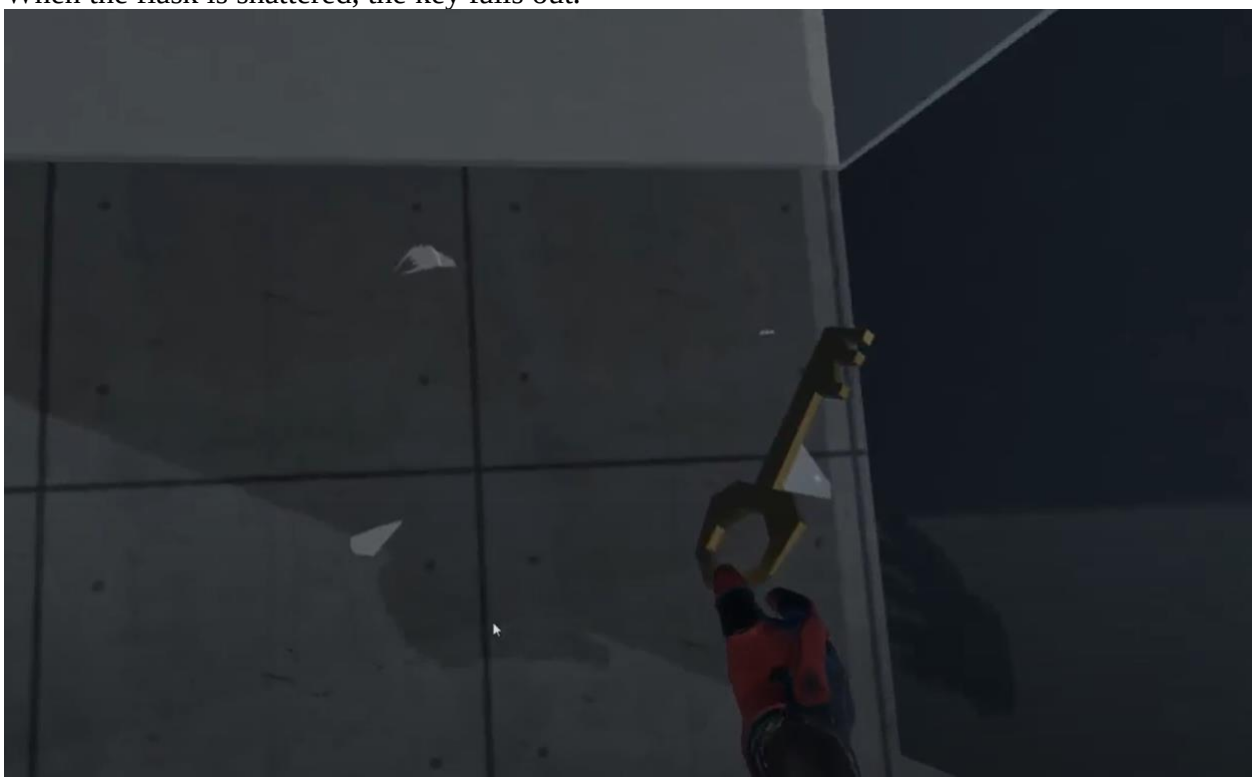

Figure 120 Key from a flask

Once the key is picked up, the player needs to find the exit door and unlock it by snapping the key to the door lock.

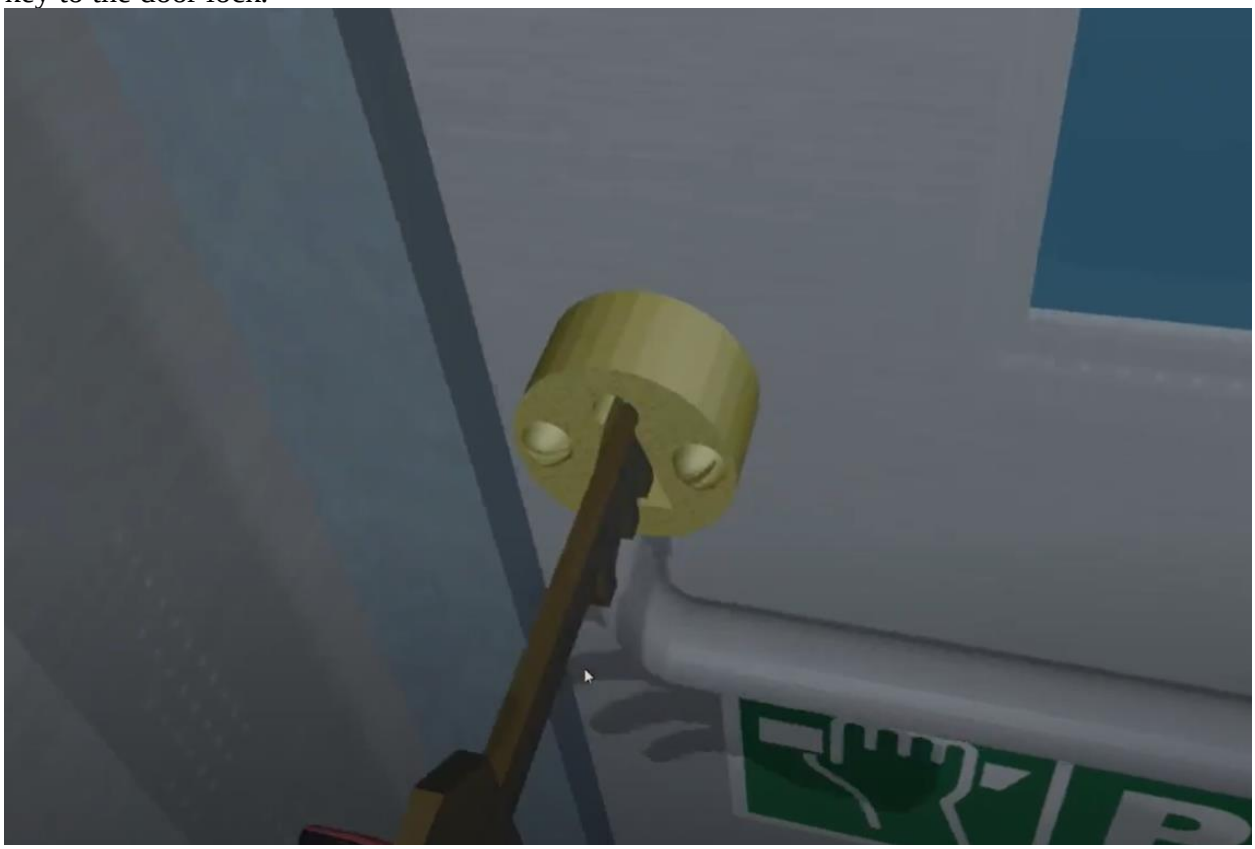

Figure 121 Unlocking the exit door

Second level

Main goal of the second level is to perform two different small experiments. Once both of the experiments are completed, player is provided with a key that unlocks the door to the final level.

Figure 122 Second level

The required experiments are:

- Dry ice experiment – white smoke starts comming out of a liquid
- Chameleon experiment – the mixed liquid changes color for some time

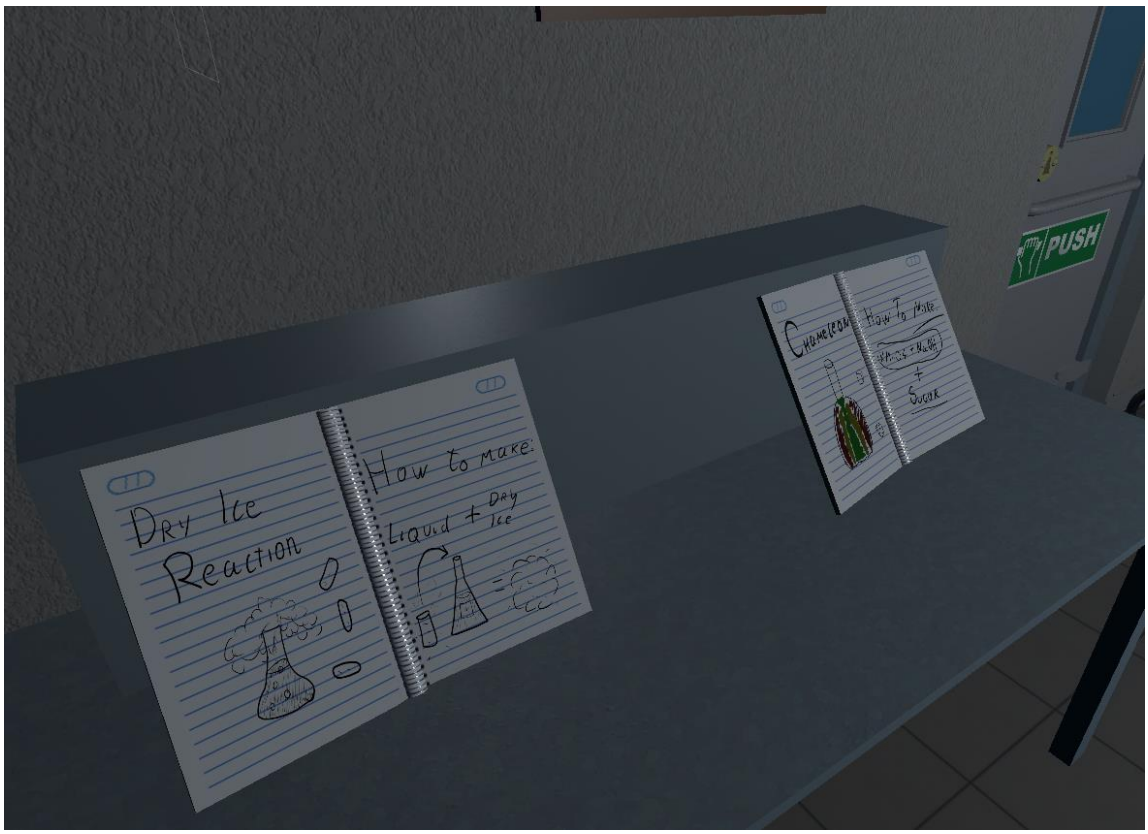

Figure 123 Experiments instructions

To perform the dry ice experiment, the player has to search for dry ice and throw it inside any liquid.

Figure 124 Dry ice box

Figure 125 Dry ice above flask

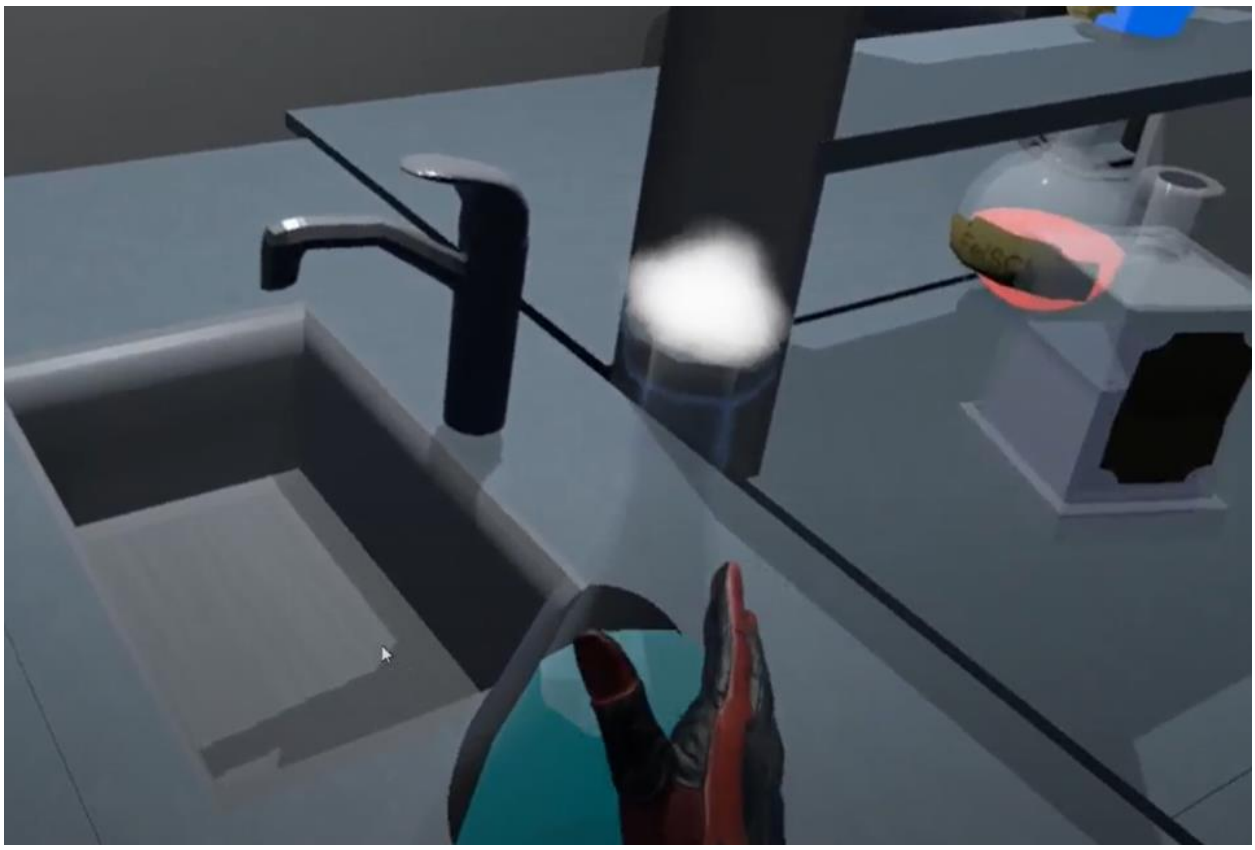

Figure 126 Dry ice thrown inside a flask

To perform the chameleon experiment, the player has to search for KMnO4 liquid and pour it into NaOH liquid.

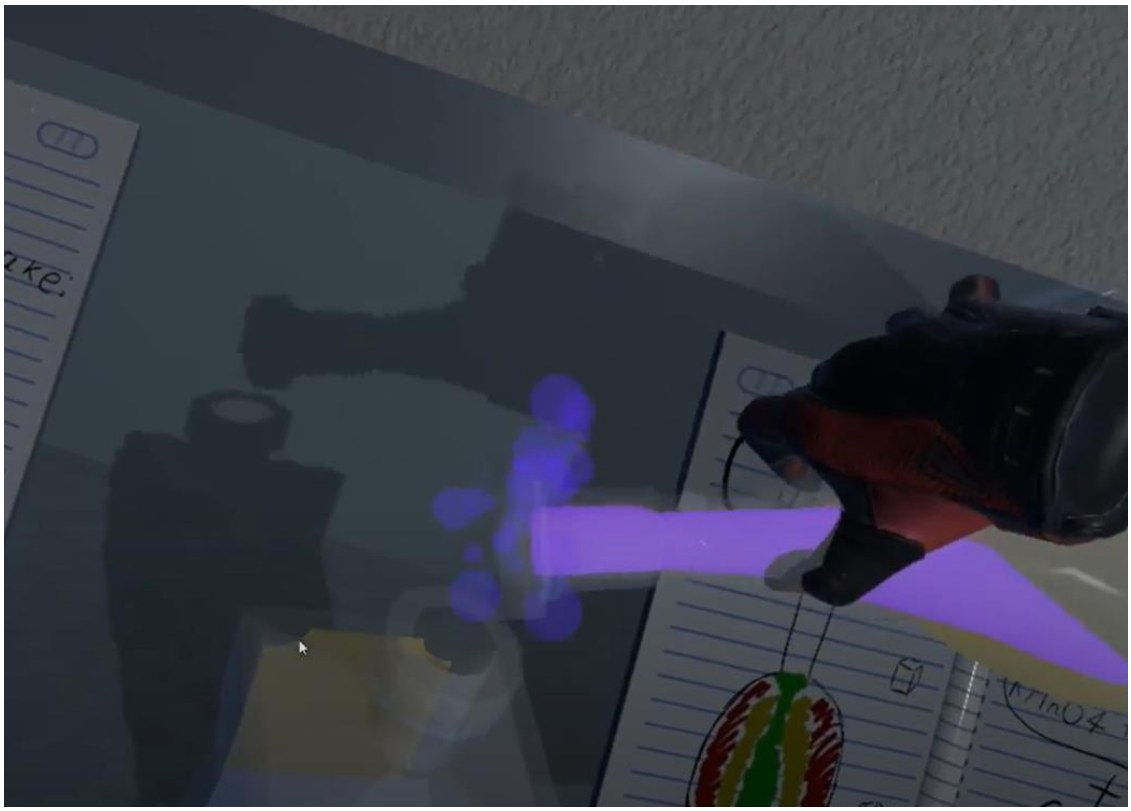

Figure 127 KMnO4 being poured into the NaOH

Then, a cube of sugar must be added into the same mix.

Figure 128 Box of sugar cubes

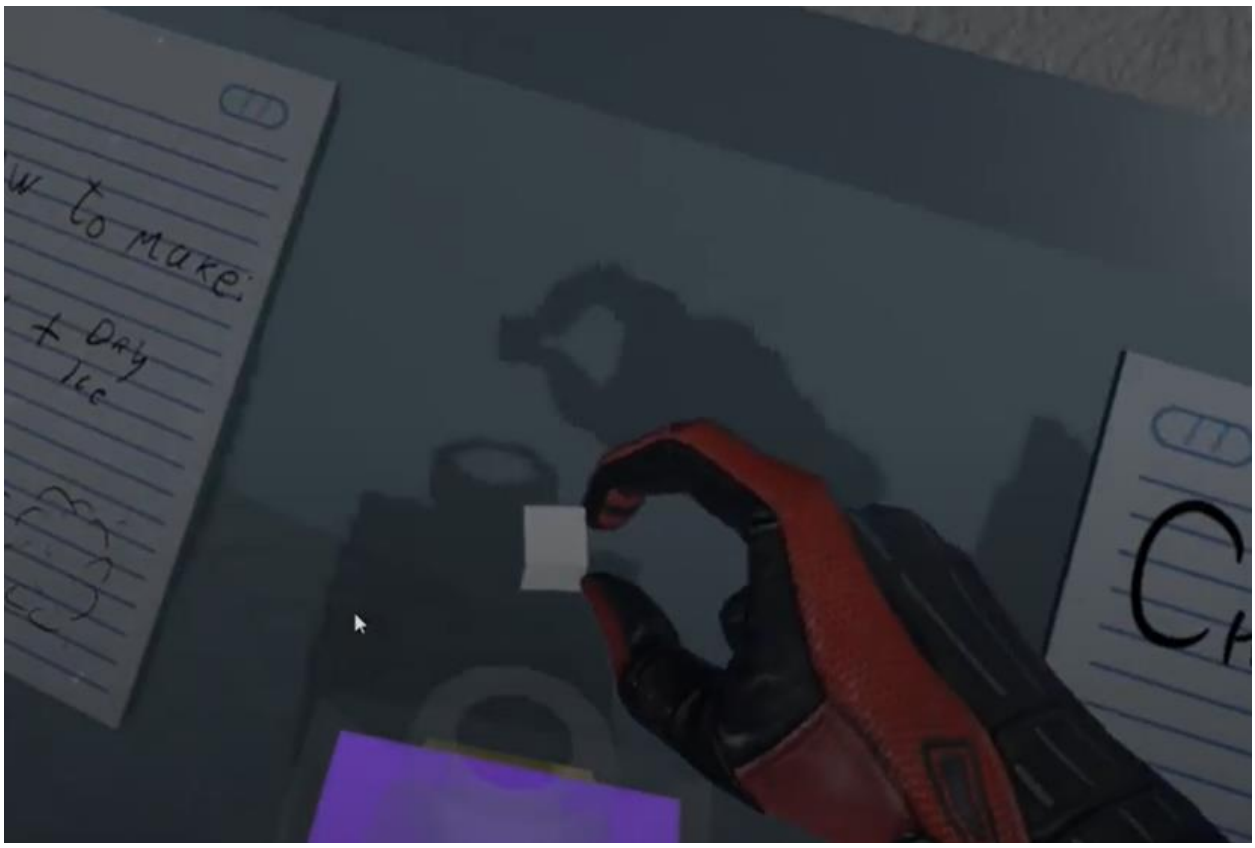

Figure 129 Sugar cube being thrown into mix of KMnO4 and NaOH

Figure 130 Exit door opened

Third level

In the last level the goal is to find three different parts of a code by searching all over the room. One of the note is hidden on top of a block of ice. To reach that note, the player has to perform blue flame experiment, so that the ice melts and the note falls down.

Figure 131 Final level

Figure 132 Final level

Figure 133 Collection of code parts

To perform the blue flame experiment, the player has to search for liquids that must be poured into the huge barrel beneath the ice. There is a Chemical Periodic table hint note that helps the during the search of these liquids.

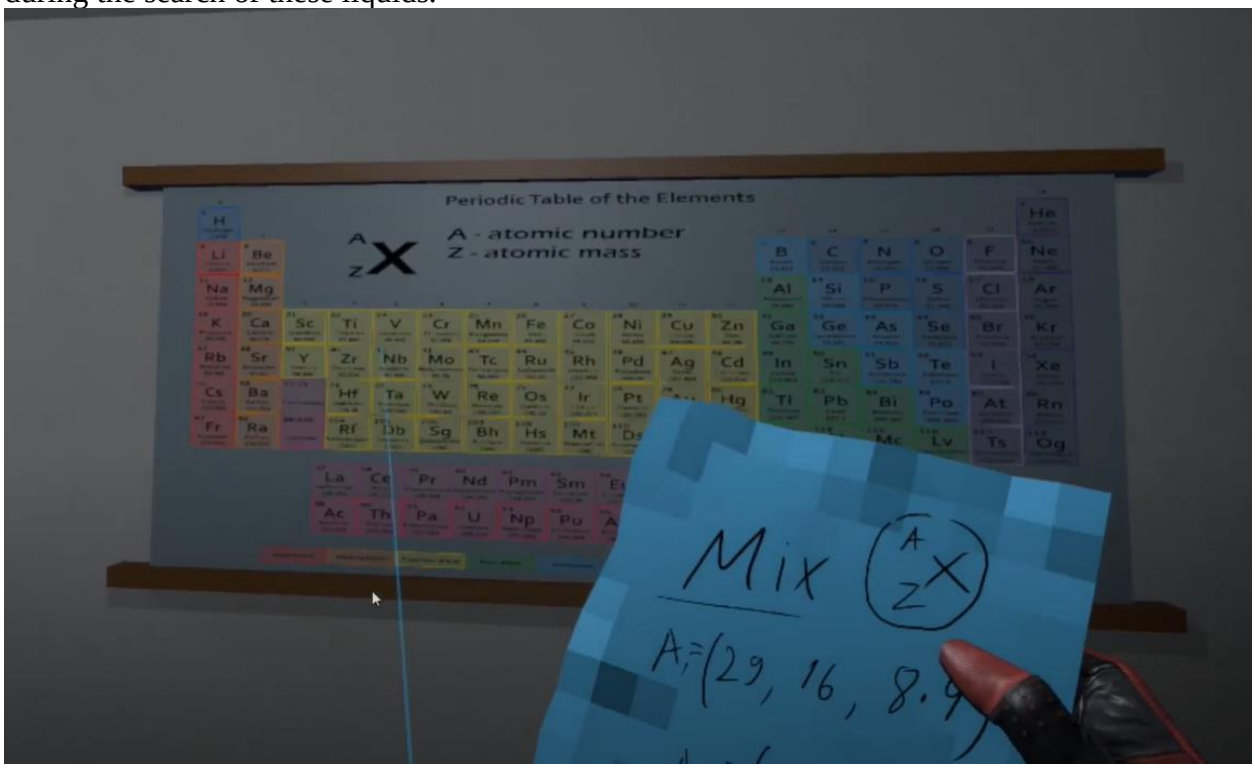

Figure 134 Chemical periodic table puzzle

One of the liquids that should be poured in is HCL.

Figure 135 HCl being poured into a barrel

Another one is CuSO4.

Figure 136 CuSO4 being poured into a barrel

Then, a block of aluminum must be thrown inside the barrel as well.

Figure 137 Aluminium being thrown into a barrel

Finally, the player needs to light the top of the barrel with a lighter. After collision with fire, the blue flame starts emitting.

Figure 138 The barrel is lit up with lighter

Once the flame reaches a certain point, the ice starts melting away and the note falls down.

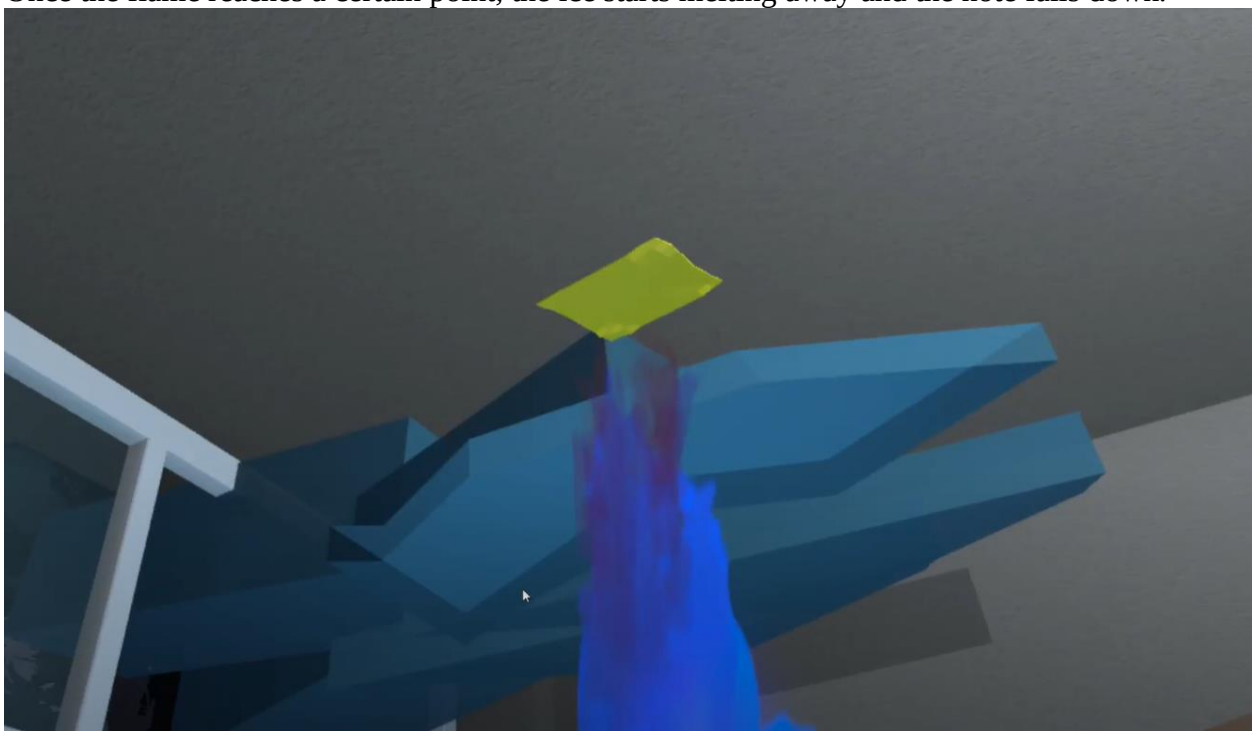

Figure 139 Blue flame experiment

After collecting all of the required code parts, the code is shown on a board that is near the exit door.

Figure 140 Collected code parts

To exit the room, the player has to write this code into the key lock, which unlocks the door, if the code is typed in correctly.

Figure 141 Exit code lock

Finish room

In the finish room, the player has the ability to write his name with the keyboard and save his score into the leaderboard.

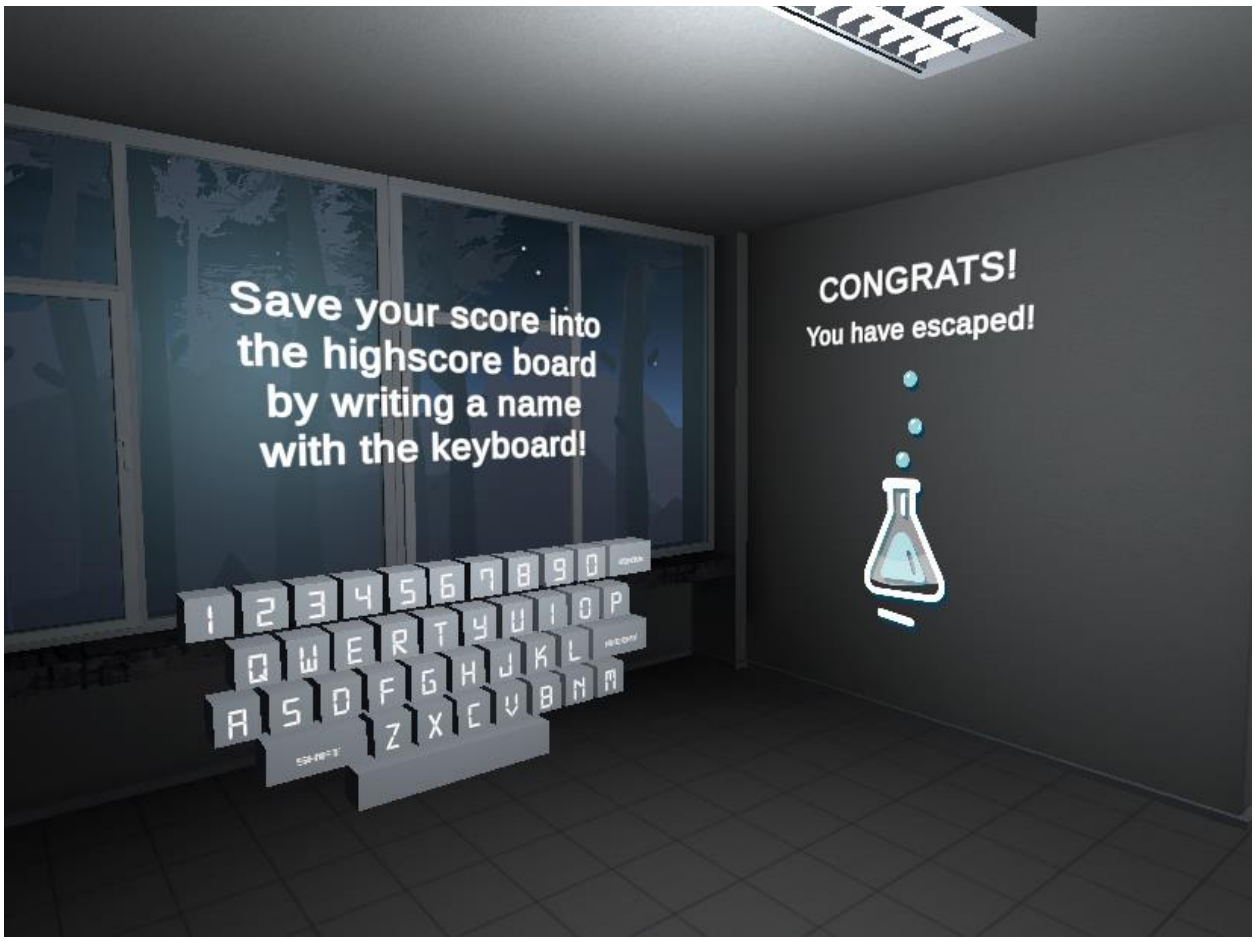

Figure 142 Finish room

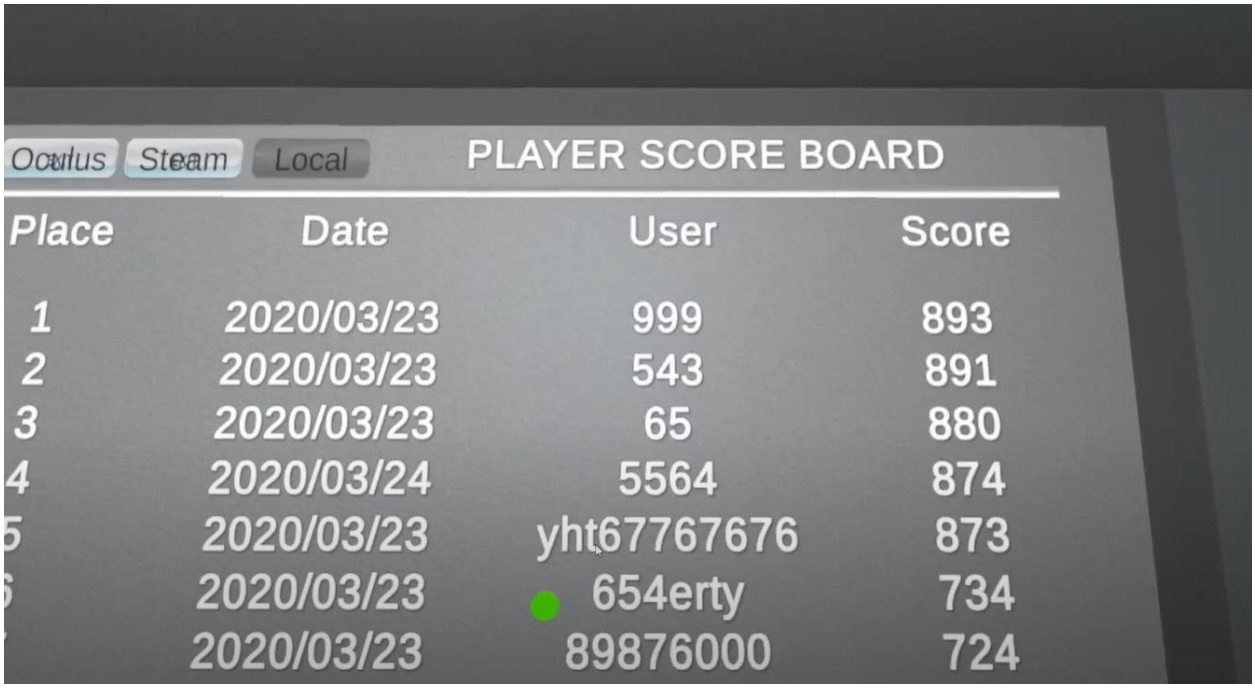

Figure 143 Leaderboard

Literature list

1. Source #1. https://vrtoolkit.readme.io/
2. Source #2. https://assetstore.unity.com/
3. Source #3. https://docs.unity3d.com/Manual/index.html
4. Source #4. https://docs.unity3d.com/ScriptReference/
5. Source #5. https://unity.com/unity/features/vr

ANNEX

All source code is contained in this part.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using VRTK;
using VRTK.Prefabs.Interactions.Controllables;

public class LockUnlockWithKey : MonoBehaviour
{
    [SerializeField] GameObject[] Note;
    [SerializeField] GameObject doorsControll;
    [SerializeField] Rigidbody rb;
    [SerializeField] string checkKey;
    [SerializeField] bool x = false;
    [SerializeField] bool y = false;
    [SerializeField] bool z = false;
    bool isSnapped = false;
    IdForUnlock[] unlock;
    public bool Open = false;
    bool oneTime = true;
    bool check = false;
    private int unlockId = 0;
    bool rigidbodyExists = false;
    int index = 0;
       [SerializeField] DelegateChange Task;
    // Start is called before the first frame update
    void Start()
    {
        doorsControll.SetActive(false);
        foreach ( GameObject obj in Note )
        {
            obj.SetActive(false);

        }
        if (x)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationX;
        }
        else if (y)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationY;
        }
        else if (z)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationZ;
        }
    }

    // Update is called once per frame
    void Update()
    {

        if (!check && isSnapped)
        {
            unlock = FindObjectsOfType<IdForUnlock>();
            if (index < unlock.Length)
            {
                if (unlock[index] != null && unlock[index].tag == checkKey)
                {
```

```
                    int id = Random.Range(1, 20);
                    unlockId = id;
                    unlock[index].SetId(id);
                    check = true;
                }
                else if(unlock[index] != null && unlock[index].tag != checkKey)
                {
                    index++;
                }

            }
            else if(index >= unlock.Length)
            {
                index = 0;
            }

        }
        else
        {
            if (Open == true && oneTime)
            {

                doorsControll.SetActive(true);
                rb.constraints = RigidbodyConstraints.None;
                foreach (GameObject obj in Note)
                {
                    obj.SetActive(true);
                }
                oneTime = false;
                rigidbodyExists = true;
                           Task.AddTask();
                Destroy(this);
            }
            if (isSnapped && unlock[index].GetId() == unlockId)
            {
                Open = true;
            }
        }


    }

    public bool isChestOpen()
    {
        if (rigidbodyExists)
            return Open;
        else
            return false;
    }

    public void Snap()
    {
        isSnapped = true;
    }
    public void UnSnap()
    {
        isSnapped = false;
    }

}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using VRTK;
```

```
using VRTK.Prefabs.Interactions.Controllables;
using VRTK.Prefabs.Interactions.Controllables.ComponentTags;
public class CloseDoors : MonoBehaviour
{
    [SerializeField] GameObject doorController;
    [SerializeField] float doorSpeed = 0.05f;
    [SerializeField] GroundControll controll;
    public Rigidbody rb;
    public bool isClosing = false;
    bool isGrabbed = false;
    //public RotationalJointDrive rotator;
    public GameObject rotator;
    int count = 0;
    // Start is called before the first frame update
    void Start()
    {
        //rotator = doorController.GetComponent<RotationalDriveFacade>();
        //rb = doorController.GetComponent<Rigidbody>();
    }

    private void Update()
    {

        if (doorController.transform.rotation.eulerAngles.y < 0.002 &&
doorController.transform.rotation.eulerAngles.y > 0 && isClosing)
        {
            rb.constraints = RigidbodyConstraints.FreezeRotationY;
            Destroy(this);
            count++;
        }
    }

    public void Grabb()
    {
        isGrabbed = true;
    }
    public void UnGrabb()
    {
        isGrabbed = false;
    }

    private void OnTriggerEnter(Collider other)
    {

        if (other.name == "Head" && count < 1 &&
doorController.transform.rotation.eulerAngles.y > 1)
        {
            isClosing = true;
            ForceToClose();
            Lock();
            count++;
            Debug.Log(count);
        }
    }

    void ForceToClose()
    {
        rb.AddForce(transform.right * doorSpeed);
    }

    void Lock()
    {
        //controll.coridorOff();
        rotator.SetActive(false);
    }
```

```
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StopTime : MonoBehaviour
{
    public LockUnlockWithKey locks;
    [SerializeField] TimeLeft time;
    // Start is called before the first frame update

    // Update is called once per frame
    void Update()
    {
        if (locks.isChestOpen())
        {
            time.StopTimer();
            time.finalStop = true;
                    //time.OneTimeSend();
        }
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using TMPro;

public class TimeLeft : MonoBehaviour
{
    [SerializeField] TextMeshPro timerMinutes;
    [SerializeField] TextMeshPro timerSeconds;
    private float stopTime;
    public float timerTime;

    public bool finalStop = false;
    private bool isRunning = false;

       private bool check = false;

    private void Start()
    {
        StopTimer();
             GlobalData.LevelsTime += timerTime;
       }

    private void Update()
    {
        if (isRunning && !finalStop)
        {
            timerTime = stopTime + (timerTime - Time.deltaTime);
            int minutesInt = (int)timerTime / 60;
            int secondsInt = (int)timerTime % 60;
            timerMinutes.text = (minutesInt < 10) ? "0" + minutesInt :
minutesInt.ToString();
            timerSeconds.text = (secondsInt < 10) ? "0" + secondsInt :
secondsInt.ToString();
            if(timerTime <= 0)
            {
```

```
                GameData.SetEnd(true);
                GameData.SetVictory(false);
                isRunning = false;
            }
        }
             else if(!isRunning && finalStop && !check)
             {
                    OneTimeSend();
                    check = true;
             }
    }

    public void StopTimer()
    {
        isRunning = false;
       }

    public void TimerStart()
    {
        if(!isRunning)
        {
            print("timer is running");
            isRunning = true;
        }
    }

    public void MinusTime(float seconds)
    {
        timerTime -= seconds;
    }

       public void OneTimeSend()
       {
                    //Debug.Log(timerTime.ToString());
                    GlobalData.Timer.Add(timerTime);
       }
}


using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BottleSmash : MonoBehaviour {

       // Use this for initialization
    //all of the required items in order to gie the impression of hte glass breaking.
    [ColorUsageAttribute(true, true, 0f, 8f, 0.125f, 3f)]
    public Color color;
    //to use to find any delta
    [HideInInspector]
    private Color cachedColor;
    //used to update any colors when the value changes, not by polling
    [SerializeField]
    [HideInInspector]
    private List<ColorBase> registeredComponents;

    public GameObject Cork, Liquid, Glass, Glass_Shattered, Label;
    //default despawn time;
    public float DespawnTime = 5.0f;

    public float time = 0.5f;

    float tempTime;
```

```
    float clacTime;

    public float splashLevel = 0.006f;

    public bool colided = false;

    //splash effect.
    public ParticleSystem Effect;
    //3D mesh on hte ground (given a specific height).
    public GameObject Splat;
    //such as the ground layer otherwise the broken glass could be considered the
'ground'
    public LayerMask SplatMask;
    //distance of hte raycast
    public float maxSplatDistance = 5.0f;
    //if the change in velocity is greater than this THEN it breaks
    public float shatterAtSpeed = 2.0f;
    //if the is disabled then it wont shatter by itself.
    public bool allowShattering = true;
    //if it collides with an object then and only then is there a period of 0.2f seconds
to shatter.
    public bool onlyAllowShatterOnCollision = true;
    //for the ability to find the change in velocity.
    [SerializeField]
    [HideInInspector]
    private Vector3 previousPos;
    [SerializeField]
    [HideInInspector]
    private Vector3 previousVelocity;
    [SerializeField]
    [HideInInspector]
    private Vector3 randomRot;
    [SerializeField]
    [HideInInspector]
    private float _lastHitSpeed = 0;
    //dont break if we have already broken, only applies to self breaking logic, not by
calling Smash()
    public bool broken = false;
    //timeout
    float collidedRecently = -1;

    LiquidVolumeAnimator lva;

    SteamVrSceleton skeleton;

       void Start () {
        if (Liquid != null)
        {
            lva = Liquid.GetComponent<LiquidVolumeAnimator>();
        }
        skeleton = GetComponent<SteamVrSceleton>();
        clacTime = time;
        tempTime = time;
        previousPos = transform.position;

       }


       //Smash function so it can be tied to buttons.
    public void RandomizeColor()
    {
        color = new Color(Random.Range(0, 1), Random.Range(0, 1), Random.Range(0, 1), 1);
    }
    void OnCollisionEnter(Collision collision)
    {
```

```
        //set a timer for about 0.2s to be able to be broken
        _lastHitSpeed = collision.impulse.magnitude;
        if (collision.transform.tag != "Liquid" && collision.transform.tag != "Absorver")
        {
            //Debug.Log(collision.transform.name);
            collidedRecently = 0.2f;
        }

    }

    public void AttemptCollision(Collision col)
    {
        OnCollisionEnter(col);
    }

    public void RegisterColorBase(ColorBase cb)
    {
        registeredComponents.Add(cb);
    }

    public void ChangedColor()
    {
        if(cachedColor != color)
        {
            cachedColor = color;

            //update all registered components
            foreach (ColorBase cb in registeredComponents)
            {
                cb.Unify();
            }
        }
    }
    public Vector3 GetRandomRotation()
    {
        return randomRot;
    }
    public void RandomRotation()
    {
        randomRot = (Random.insideUnitSphere + Vector3.forward).normalized;
    }

    public void Smash()
    {

        skeleton.UnGrab();
        broken = true;
        //the Corks collider needs to be turned on;
        if (Cork != null)
        {
            Cork.transform.parent = null;
            Cork.GetComponent<Collider>().enabled = true;
            Cork.GetComponent<Rigidbody>().isKinematic = false;
            Destroy(Cork.gameObject, DespawnTime);
        }
        //the Liquid gets removed after n seconds
        if (Liquid != null)
        {
            float t = 0.0f;
            //if (Effect != null)
            //    t = (Effect.main.startLifetime.constantMin +
Effect.main.startLifetime.constantMax)/2;
            Destroy(Liquid.gameObject, t);
        }
        //particle effect
```

```
        if(Effect != null && lva != null && lva.level > splashLevel)
        {
            Effect.Play();
            Destroy(Effect.gameObject, Effect.main.startLifetime.constantMax);
        }
        else if (Effect != null && lva != null && lva.level < splashLevel)
        {
            Destroy(Effect.gameObject, Effect.main.startLifetime.constantMax);
        }
        else if (Effect != null && lva == null)
        {
            Destroy(Effect.gameObject, Effect.main.startLifetime.constantMax);
        }

        //now the label;
        if (Label != null)
        {
            //Label.transform.parent = null;
            //Label.GetComponent<Collider>().enabled = true;
            //Label.GetComponent<Rigidbody>().isKinematic = false;
            Destroy(Label.gameObject);
        }
        //turn Glass off and the shattered on.
        if (Glass != null)
        {
            Destroy(Glass.gameObject);
        }
        if (Glass_Shattered != null)
        {
            Glass_Shattered.SetActive(true);
            Glass_Shattered.transform.parent = null;
            Destroy(Glass_Shattered, DespawnTime);
        }

        //instantiate the splat.
        RaycastHit info = new RaycastHit();
        if(Splat != null)
        if (Physics.Raycast(transform.position, Vector3.down, out info, maxSplatDistance,
SplatMask))
        {
            GameObject newSplat = Instantiate(Splat);
            newSplat.transform.position = info.point;

        }
        Destroy(transform.gameObject, DespawnTime);

    }
      // Update is called once per frame, for the change in velocity and all that
jazz...
      void FixedUpdate () {
        ChangedColor();
        collidedRecently -= Time.deltaTime;
        Vector3 currentVelocity = (transform.position - previousPos) /
Time.fixedDeltaTime;
        if ((onlyAllowShatterOnCollision && collidedRecently >= 0.0f) |
!onlyAllowShatterOnCollision)
        {
            if (allowShattering)
            {
                if (Vector3.Distance(currentVelocity, previousVelocity) > shatterAtSpeed
|| _lastHitSpeed > shatterAtSpeed)
                {
                    if (!broken)
                        Smash();
                }
```

```
            }
        }
        _lastHitSpeed = 0;

        previousVelocity = currentVelocity;
        previousPos = transform.position;
       }

    public void ResetVelocity()
    {
        previousVelocity = Vector3.zero;
        previousPos = transform.position;
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ColbaNoBreaking : MonoBehaviour
{
    [SerializeField] float time = 0;
    float calTime = 0;
    BottleSmash smash;
    bool grabbed = false;
    // Start is called before the first frame update
    void Start()
    {
        smash = GetComponent<BottleSmash>();
    }

    // Update is called once per frame
    void Update()
    {
        if (grabbed) {
            if (calTime < time)
            {
                smash.enabled = false;
                calTime += Time.deltaTime;
            }
            else {
                smash.ResetVelocity();
                smash.enabled = true;
            }
        }
        else if (!grabbed)
        {
            smash.ResetVelocity();
            smash.enabled = true;
            calTime = 0;
        }
    }

    public void Grab()
    {
        grabbed = true;

    }

    public void Ungrab()
    {
        grabbed = false;
```

```
    }

}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RunningFlask : MonoBehaviour
{
    [SerializeField] Animator anim;
    bool animatorActive = false;

    void Start()
    {
        anim.enabled = false;
    }

    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.name == "HandAvatar" && !animatorActive)
        {
            anim.enabled = true;
        }
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using VRTK;
public class WallPlugSparks : MonoBehaviour
{
    //TODO fix animation component disabling/enabling
    [SerializeField] List<GameObject> lights;
    [SerializeField] Vector2 RandonTimeFromTo = Vector2.zero;
    [SerializeField] ParticleSystem particle;
    public GameObject[] litObjs;
    public GameObject[] unlitObjs;
    List<float> timeToSpark;
    List<float> timeCount;
    List<bool> checkSparkle;
    bool spark = false;
    int lightBlinked = 0;
    AudioSource aud;
    bool isTouched = false;
    private void Start()
    {
        timeToSpark = new List<float>();
        timeCount = new List<float>();
        checkSparkle = new List<bool>();
        for (int i = 0; i < lights.Count; i++)
        {
            timeToSpark.Add(Random.Range(RandonTimeFromTo.x, RandonTimeFromTo.y));
            checkSparkle.Add(false);
            timeCount.Add(0);
        }

        aud = GetComponent<AudioSource>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag != "GameController" && other.name !=
"[VRTK][AUTOGEN][Controller][NearTouch][CollidersContainer]" && other != null)
```

```
    {
        particle.Play();
        aud.Play();
        spark = true;
        for (int i = 0; i < timeToSpark.Count; i++)
        {
            timeToSpark[i] = Random.Range(RandonTimeFromTo.x, RandonTimeFromTo.y);
        }
    }
}

private void Update()
{
    if (isTouched && spark == false)
    {
        particle.Play();
        aud.Play();
        spark = true;
        for (int i = 0; i < timeToSpark.Count; i++)
        {
            timeToSpark[i] = Random.Range(RandonTimeFromTo.x, RandonTimeFromTo.y);
        }
    }
    if (spark)
    {
        for (int i = 0; i < timeToSpark.Count; i++)
        {
            if (timeToSpark[i] >= timeCount[i])
            {
                litObjs[i].SetActive(false);
                unlitObjs[i].SetActive(true);
                lights[i].GetComponent<Light>().enabled = false;
                timeCount[i] += Time.deltaTime;
            }
            else if (timeToSpark[i] < timeCount[i] && !checkSparkle[i])
            {
                lightBlinked++;
                litObjs[i].SetActive(true);
                unlitObjs[i].SetActive(false);
                lights[i].GetComponent<Light>().enabled = true;
                checkSparkle[i] = true;
            }
        }
        if (lightBlinked >= timeToSpark.Count)
        {
            for (int i = 0; i < timeToSpark.Count; i++)
            {
                if (lightBlinked == timeToSpark.Count)
                {
                    checkSparkle[i] = false;
                    timeCount[i] = 0;
                }
            }
            spark = false;
            lightBlinked = 0;
        }

    }
}

public void Touch()
{
    isTouched = true;
}
public void Relise()
```

```
    {
        isTouched = false;
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CollisionDetectionSound : MonoBehaviour
{
    [SerializeField] DropSound drop;
    // Start is called before the first frame update
    void OnCollisionEnter(Collision collision)
    {
        if (collision.relativeVelocity.magnitude > 1)
        {
            drop.PlaySound();
        }
    }
}

public class InteractableCollisionUI : MonoBehaviour
{
    [SerializeField] float sliderOffset = 0.5f;
    [SerializeField] float thicknes = 0.01f;
    [SerializeField] bool buttonPressWithTrigger = false;
    [SerializeField] GameObject triggerGo;
    BoxCollider colliderForPosition;
    List<UiButtonClick> buttons;
    List<UiSliderControll> sliders;
    List<UiDropdownControll> dropdowns;
    RectTransform uiTransform;
    static float SliderOffset = 0;
    void Start()
    {
        if (triggerGo != null)
        {
            if (buttonPressWithTrigger)
            {
                triggerGo.SetActive(true);
            }
            else
            {
                triggerGo.SetActive(false);
            }
        }
        buttons = new List<UiButtonClick>();
        sliders = new List<UiSliderControll>();
        dropdowns = new List<UiDropdownControll>();
        gameObject.AddComponent<Rigidbody>();
        gameObject.AddComponent<BoxCollider>();
        uiTransform = GetComponent<RectTransform>();
        colliderForPosition = gameObject.GetComponent<BoxCollider>();
        colliderForPosition.size = new Vector3(uiTransform.sizeDelta.x,
uiTransform.sizeDelta.y, thicknes);
        colliderForPosition.isTrigger = true;
        Rigidbody rb = GetComponent<Rigidbody>();
        rb.isKinematic = true;
        RecursiveChilds(gameObject.transform);
        SliderOffset = sliderOffset;
    }
    void RecursiveChilds(Transform transformGo)
    {
        for (int i = 0; i < transformGo.childCount; i++)
```

```
        {
            if (transformGo.GetChild(i).GetComponent<Button>() != null)
            {
                transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                UiButtonClick btn =
transformGo.GetChild(i).gameObject.AddComponent<UiButtonClick>();
                buttons.Add(btn);
                btn.SetUiController(this);
                btn.UseButtonWithTrigger(buttonPressWithTrigger);
                RectTransform uiTransformTemp =
transformGo.GetChild(i).gameObject.GetComponent<RectTransform>();
                BoxCollider colliderForPositionTemp =
transformGo.GetChild(i).gameObject.GetComponent<BoxCollider>();
                float tempThicknes = thicknes + 0.02f;
                colliderForPositionTemp.size = new Vector3(uiTransformTemp.sizeDelta.x,
uiTransformTemp.sizeDelta.y, tempThicknes);
                colliderForPositionTemp.center = new Vector3(0, 0, 0);
            }
            if (transformGo.GetChild(i).GetComponent<Slider>() != null)
            {
                transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                UiSliderControll slider =
transformGo.GetChild(i).gameObject.AddComponent<UiSliderControll>();
                sliders.Add(slider);
                slider.SetUiController(this);
                slider.UseButtonWithTrigger(buttonPressWithTrigger);
                RectTransform uiTransformTemp =
transformGo.GetChild(i).gameObject.GetComponent<RectTransform>();
                BoxCollider colliderForPositionTemp =
transformGo.GetChild(i).gameObject.GetComponent<BoxCollider>();
                float tempThicknes = thicknes + 0.02f;
                colliderForPositionTemp.size = new Vector3(uiTransformTemp.sizeDelta.x,
uiTransformTemp.sizeDelta.y, tempThicknes);
                colliderForPositionTemp.center = new Vector3(0, 0, 0);
            }
            if (transformGo.GetChild(i).GetComponent<TMP_Dropdown>() != null ||
transformGo.GetChild(i).GetComponent<Dropdown>() != null)
            {
                transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                UiDropdownControll dropdown =
transformGo.GetChild(i).gameObject.AddComponent<UiDropdownControll>();
                dropdowns.Add(dropdown);
                dropdown.SetUiController(this);
                dropdown.UseButtonWithTrigger(buttonPressWithTrigger);
                RectTransform uiTransformTemp =
transformGo.GetChild(i).gameObject.GetComponent<RectTransform>();
                BoxCollider colliderForPositionTemp =
transformGo.GetChild(i).gameObject.GetComponent<BoxCollider>();
                float tempThicknes = thicknes + 0.02f;
                colliderForPositionTemp.size = new Vector3(uiTransformTemp.sizeDelta.x,
uiTransformTemp.sizeDelta.y, tempThicknes);
                colliderForPositionTemp.center = new Vector3(0, 0, 0);
            }
            if (transformGo.GetChild(i).GetComponent<Scrollbar>() != null)
            {
                BoxCollider bx =
transformGo.GetChild(i).gameObject.AddComponent<BoxCollider>();
                ScrollRect sr =
transformGo.GetChild(i).parent.GetComponent<ScrollRect>();
                bx.center = new Vector3(-
(bx.gameObject.GetComponent<RectTransform>().sizeDelta.x / 2), -
(sr.GetComponent<RectTransform>().sizeDelta.y / 2), 0);
                bx.size = new
Vector3(bx.gameObject.GetComponent<RectTransform>().sizeDelta.x,
sr.GetComponent<RectTransform>().sizeDelta.y, 0.1f);
```

```
                transformGo.GetChild(i).gameObject.AddComponent<VerticalScroll>();
                VerticalScroll temp =
sr.gameObject.transform.GetChild(sr.gameObject.transform.childCount -
1).gameObject.GetComponent<VerticalScroll>();
                temp.UseButtonWithTrigger(buttonPressWithTrigger);
            }
            RecursiveChilds(transformGo.GetChild(i).transform);
        }
    }
    public static float SliderOffsetReturn()
    {
        return SliderOffset;
    }
    public bool buttonWithTrigger()
    {
        return buttonPressWithTrigger;
    }
}



public class UiButtonControll : MonoBehaviour
{
    [SerializeField] GameObject loadingScreen;
    [SerializeField] AudioMixer volumeControll;
    ButtonClick menu;
    private void OnEnable()
    {
        menu = gameObject.transform.parent.parent.GetComponent<ButtonClick>();
    }
    public void LoadLevel(int index)
    {
        StartCoroutine(LoadAsynchronously(index));
    }
    public void RestartLevel()
    {
        StartCoroutine(LoadAsynchronously(SceneManager.GetActiveScene().buildIndex));
    }
    IEnumerator LoadAsynchronously(int sceneIndex)
    {
        loadingScreen.SetActive(true);
        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);
        while (!operation.isDone)
        {
            float progress = Mathf.Clamp01(operation.progress / .9f);
            Debug.Log(progress);
            yield return null;
        }
    }
    public void LoadExit()
    {
        Debug.Log("EXIT");
        Application.Quit();
    }
    public void BackToPlay()
    {
        menu.ButtonPress();
    }
    public void MasterVolumeSlider(float masterVolume)
    {
        volumeControll.SetFloat("MasterVolume", masterVolume);
    }
    public void FXVolumeSlider(float fxVolume)
    {
```

```
        volumeControll.SetFloat("SoundFxVolume", fxVolume);
    }
    public void MusicVolumeSlider(float musicVolume)
    {
        volumeControll.SetFloat("MusicVolume", musicVolume);
    }
}



public class StartResolutionUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    int indexResolution = 0;
    bool found = false;
    List<Resolution> res;
    public static bool fullScreen = false;
    public bool firstTime = true;
    void OnEnable()
    {
        indexResolution = 0;
        if (firstTime)
        {
            found = false;
            if (PlayerPrefs.HasKey("WindowMode"))
            {
                if (PlayerPrefs.GetInt("WindowMode") == 0)
                {
                    fullScreen = true;
                }
                else
                {
                    fullScreen = false;
                }
            }
            else
            {
                fullScreen = Screen.fullScreen;
            }
            Resolution[] resolutions = Screen.resolutions;
            dropdown = GetComponent<TMP_Dropdown>();
            dropdown.ClearOptions();
            List<string> options = new List<string>();
            res = new List<Resolution>();
            foreach (Resolution item in resolutions)
            {
                res.Add(item);
                options.Add(item.width + "X" + item.height);
                if (!PlayerPrefs.HasKey("ResolutionSave"))
                {
                    if (item.width != Screen.width && item.height != Screen.height &&
!found)
                    {
                        indexResolution++;
                    }
                    else if (item.width == Screen.width && item.height == Screen.height
&& !found)
                    {
                        found = true;
                    }
                }
                else
                {
```

```
                    if (item.width + "X" + item.height !=
PlayerPrefs.GetString("ResolutionSave") && !found)
                    {
                        indexResolution++;
                    }
                    else if (item.width + "X" + item.height ==
PlayerPrefs.GetString("ResolutionSave") && !found)
                    {
                        found = true;
                    }
                }
            }
            dropdown.AddOptions(options);
            firstTime = false;
        }
        else
        {
            found = false;
            for (int i = 0; i < res.Count; i++)
            {
                if (!PlayerPrefs.HasKey("ResolutionSave"))
                {
                    if (res[i].width == Screen.width && res[i].height == Screen.height &&
!found)
                    {
                        found = true;
                        indexResolution = i;
                    }
                }
                else
                {
                    if (res[i].width + "X" + res[i].height ==
PlayerPrefs.GetString("ResolutionSave") && !found)
                    {
                        found = true;
                        indexResolution = i;
                    }
                }
            }
        }
        dropdown.value = indexResolution;
        ChangeResolution(indexResolution);
    }
    public void ChangeResolution(int index)
    {
        Screen.SetResolution(res[index].width, res[index].height, fullScreen);
        PlayerPrefs.SetString("ResolutionSave", res[index].width + "X" +
res[index].height);
        //UpdateSettings.Save = true;
    }
}



public class WindowUI : MonoBehaviour
{
    TMP_Dropdown dropdown;

    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
        }
```

```
        if (Screen.fullScreen)
        {
            dropdown.value = 0;
        }
        else
        {
            dropdown.value = 1;
        }
    }

    public void SetFullscreen(int isFullscreen)
    {
        if (isFullscreen == 0)
        {
            Screen.fullScreen = true;
            StartResolutionUi.fullScreen = true;
        }
        else if (isFullscreen == 1)
        {
            Screen.fullScreen = false;
            StartResolutionUi.fullScreen = false;
        }

        UpdateSettings.Save = true;
    }
}



public class StartQualityUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    // Start is called before the first frame update

    private void OnEnable()
    {
        if (dropdown != null)
        {
            dropdown.value = QualitySettings.GetQualityLevel();
        }
    }

    void Start()
    {
        dropdown = GetComponent<TMP_Dropdown>();
        dropdown.ClearOptions();
        List<string> options = new List<string>();

        for (int i = 0; i < QualitySettings.names.Length; i++)
        {
            options.Add(QualitySettings.names[i]);
        }
        dropdown.AddOptions(options);
        dropdown.value = QualitySettings.GetQualityLevel();

    }

    public void ChangeQuality(int index)
    {
        QualitySettings.SetQualityLevel(index, true);
        transform.parent.gameObject.SetActive(false);
        transform.parent.gameObject.SetActive(true);
        UpdateSettings.Save = true;
    }
```

```
}
public class ShadowQualityResUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    // Start is called before the first frame update
    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
        }
        if (QualitySettings.shadowResolution == ShadowResolution.Low)
        {
            dropdown.value = 0;
        }
        else if (QualitySettings.shadowResolution == ShadowResolution.Medium)
        {
            dropdown.value = 1;
        }
        else if (QualitySettings.shadowResolution == ShadowResolution.High)
        {
            dropdown.value = 2;
        }
        else if (QualitySettings.shadowResolution == ShadowResolution.VeryHigh)
        {
            dropdown.value = 3;
        }

    }

    public void ChangeShadowRes(int index)
    {
        if (index == 0)
        {
            QualitySettings.shadowResolution = ShadowResolution.Low;
        }
        else if (index == 1)
        {
            QualitySettings.shadowResolution = ShadowResolution.Medium;
        }
        else if (index == 2)
        {
            QualitySettings.shadowResolution = ShadowResolution.High;
        }
        else if (index == 3)
        {
            QualitySettings.shadowResolution = ShadowResolution.VeryHigh;
        }
        UpdateSettings.Save = true;
    }

}


public class ShadowCascadesUi : MonoBehaviour
{
    TMP_Dropdown dropdown;

    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
        }
        if (QualitySettings.shadowCascades == 0)
```

```
        {
            dropdown.value = 0;
        }
        else if (QualitySettings.shadowCascades == 2)
        {
            dropdown.value = 1;
        }
        else if (QualitySettings.shadowCascades == 4)
        {
            dropdown.value = 2;
        }
    }

    public void SetShadowCascades(int index)
    {
        if (index == 0)
        {
            QualitySettings.shadowCascades = 0;
        }
        else if (index == 1)
        {
            QualitySettings.shadowCascades = 2;
        }
        else if (index == 2)
        {
            QualitySettings.shadowCascades = 4;
        }
        UpdateSettings.Save = true;
    }
}


public class ShadowDistanceUi : MonoBehaviour
{
    Slider slider;
    private void OnEnable()
    {
        if (slider == null)
        {
            slider = GetComponent<Slider>();
        }
        slider.value = QualitySettings.shadowDistance;
    }

    public void SetShadowDistance(float index)
    {
        QualitySettings.shadowDistance = slider.value;
        UpdateSettings.Save = true;
    }
}


public class AntiAliasing : MonoBehaviour
{
    TMP_Dropdown dropdown;

    void Start()
    {
        dropdown = GetComponent<TMP_Dropdown>();
    }

    private void OnEnable()
    {
        if (dropdown != null)
```

```
        {
            switch (QualitySettings.antiAliasing)
            {
                case 0:
                    dropdown.value = 0;
                    break;
                case 2:
                    dropdown.value = 1;
                    break;
                case 4:
                    dropdown.value = 2;
                    break;
                default:
                    dropdown.value = 3;
                    break;
            }
        }
    }

    public void SetAntiAliasing(int index)
    {
        switch (dropdown.value)
        {
            case 0:
                index = 1;
                break;
            case 1:
                index = 2;
                break;
            case 2:
                index = 4;
                break;
            default:
                index = 8;
                break;
        }
        QualitySettings.antiAliasing = index;
        UpdateSettings.Save = true;
    }
}

public class ShadowQualityUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    private void OnEnable()
    {
        if (dropdown == null)
        {
            dropdown = GetComponent<TMP_Dropdown>();
            if (QualitySettings.shadows == ShadowQuality.HardOnly)
            {
                dropdown.value = 0;
            }
            else if (QualitySettings.shadows == ShadowQuality.All)
            {
                dropdown.value = 1;
            }
        }
        else
        {
            if (QualitySettings.shadows == ShadowQuality.HardOnly)
            {
                dropdown.value = 0;
            }
            else if (QualitySettings.shadows == ShadowQuality.All)
```

```
            {
                dropdown.value = 1;
            }
        }
    }

    public void ChangeShadowQuality(int index)
    {
        if (index == 0)
        {
            QualitySettings.shadows = ShadowQuality.HardOnly;
        }
        else if (index == 1)
        {
            QualitySettings.shadows = ShadowQuality.All;
        }
        UpdateSettings.Save = true;
    }
}



public class SoftParticlesUi : MonoBehaviour
{
    TMP_Dropdown dropdown;
    // Start is called before the first frame update
    private void OnEnable()
    {
        if (dropdown == null)
            dropdown = GetComponent<TMP_Dropdown>();
        if (QualitySettings.softParticles == false)
        {
            dropdown.value = 0;
        }
        else
            dropdown.value = 1;
    }

    public void SetSoftParticle(int index)
    {
        if (index == 0)
        {
            QualitySettings.softParticles = false;
        }
        else if (index == 1)
        {
            QualitySettings.softParticles = true;
        }
        UpdateSettings.Save = true;
    }
}



public class NoteBoard : MonoBehaviour
{
    [SerializeField] GameObject printKey;
    bool isGrabbed = false;
    TextMeshPro print;
    TextMeshPro key;
    int count = 0;
       [SerializeField] DelegateChange Task;
    void Start()
    {
```

```
        print = printKey.GetComponent<TextMeshPro>();
        key = GetComponentInChildren<TextMeshPro>();
    }

    void FixedUpdate()
    {

        if (isGrabbed)
        {
            print.text = key.text;
                    Task.AddTask();
        }
    }

    public void Grab()
    {
        isGrabbed = true;
    }

    public void UnGrab()
    {
        isGrabbed = false;
    }

}


public class LockUnlockWithPin : MonoBehaviour
{
    [SerializeField] GameObject doorsControll;
    [SerializeField] Rigidbody doorsControllRb;
    [SerializeField] int keyLenght = 0;
    [SerializeField] int[] customKey;
    [SerializeField] GameObject[] keyObjects;
    [SerializeField] TimeLeft timer;
    [SerializeField] GroundControll ground;
    [SerializeField] float MinusTimeBad;
    [SerializeField] AccessGranted access;
    [SerializeField] bool x = false;
    [SerializeField] bool y = false;
    [SerializeField] bool z = false;
    AccessDenied accessDenied;
    string tempTMPRO = "";
    StringBuilder stringBuild;
    TextMeshPro text;
    public bool open = false;
    bool fail = true;
    bool equals = true;
    int count = 0;
    int countCheck = 0 ;
    int[] key;
    int[] playerNumbers;
    bool firstTime = true;
       [SerializeField] DelegateChange Task;

    bool correct = false;

    bool saveBool = false;
    // Start is called before the first frame update
    void Start()
    {
        stringBuild = new StringBuilder();
        text = GetComponent<TextMeshPro>();
        accessDenied = GetComponent<AccessDenied>();
        doorsControll.SetActive(false);
```

```
        if (x)
        {
            doorsControllRb.constraints = RigidbodyConstraints.FreezeRotationX;
        }
        else if (y)
        {
            doorsControllRb.constraints = RigidbodyConstraints.FreezeRotationY;
        }
        else if (z)
        {
            doorsControllRb.constraints = RigidbodyConstraints.FreezeRotationZ;
        }
        if (customKey.Length == 0)
        {
            key = new int[keyLenght];
            playerNumbers = new int[keyLenght];
            for (int i = 0; i < keyLenght; i++)
            {
                key[i] = UnityEngine.Random.Range(0, 9);
                keyObjects[i].GetComponentInChildren<TextMeshPro>().text =
key[i].ToString();
                Debug.Log(key[i]);
            }
        }
        else
        {
            keyLenght = customKey.Length;
            key = customKey;
            playerNumbers = new int[keyLenght];
        }


    }

    // Update is called once per frame
    void Update()
    {
        if (firstTime)
        {
            firstTime = false;
        }
        if (countCheck < keyLenght)
        {
            if (key[countCheck] != playerNumbers[countCheck])
            {
                equals = false;
            }

        }
        countCheck++;

        if (equals && count == keyLenght && countCheck >= keyLenght)
        {
            doorsControll.SetActive(true);
            doorsControllRb.constraints = RigidbodyConstraints.None;
            text.text = "Access Granted";
            timer.StopTimer();
            timer.finalStop = true;
                    Task.AddTask();
            GameData.SetTime((int)Math.Truncate(timer.timerTime));
            GameData.SetEnd(true);
            GameData.SetVictory(true);
            if (ground != null)
            {
                ground.finishOn();
```

```
            }
            open = true;
            if (saveBool == false)
            {
                GlobalData.WriteToFile();
                saveBool = true;
            }
        }
        if (!equals && count == keyLenght && !open && countCheck >= keyLenght)
        {
            text.text = "Access Denied";
            accessDenied.StartSound();
            timer.MinusTime(MinusTimeBad);
            count = 0;
            playerNumbers = new int[keyLenght];
            fail = true;
        }

        if (countCheck >= keyLenght && !equals)
        {
            countCheck = 0;
            equals = true;
        }

        if(open && !correct)
        {
            correct = true;
            access.PlaySound();
        }

    }

    public void SetNumber(int val)
    {
        if (count < keyLenght)
        {

            if (fail)
            {
                stringBuild = new StringBuilder();
                stringBuild.Append(tempTMPRO);
                text.text = stringBuild.ToString();
                fail = false;

            }
            if (!open && count >= 0)
            {
                playerNumbers[count] = val;
                stringBuild.Append(" " + val);
                text.text = stringBuild.ToString();
                count++;
            }
            else if (!open && count < 0)
            {
                count = 0;
                playerNumbers[count] = val;
                stringBuild.Append(" " + val);
                text.text = stringBuild.ToString();
                count++;
            }
        }

    }

    public void Del()
```

```
    {
        if (!open && count >= 0)
        {
            if (count == 0)
            {
                stringBuild.Remove(stringBuild.Length - 1, 1);
                text.text = stringBuild.ToString();
            }
            else
            {
                stringBuild.Remove(stringBuild.Length - 2, 2);
                text.text = stringBuild.ToString();
            }
            count--;
            if (count < 0)
            {
                stringBuild = new StringBuilder();
                stringBuild.Append(tempTMPRO);
                text.text = stringBuild.ToString();
            }
        }
    }
}
    }

}


public class MixingScript : MonoBehaviour
{
    [SerializeField] string mixedTag = "Mixed";
    [SerializeField] int specialLimit = 0;
    [SerializeField] string fire;
    [SerializeField] List<string> ignoreCollision;
    [SerializeField] GameObject ps;
    [SerializeField] List<string> LiquidTags;
    [SerializeField] int[] absorve;
    [SerializeField] bool special;
    List<string> tags;
    public bool mix = true;
    int[] liquidAbsorve;
    int MixSpec = 0;
    int check = 0;
    int cnt = 0;
    int indexCheckTag = 0;
    bool checkForMixed = false;
    public bool mixed = false;

    void Start()
    {
        liquidAbsorve = new int[LiquidTags.Count];
        tags = new List<string>();
    }

    void OnParticleCollision(GameObject other)
    {

        if (fire != other.tag && other.tag != "Untagged" &&
!ignoreCollision.Contains(other.tag) && other.tag != mixedTag)
        {
            if (!tags.Contains(other.tag))
            {
                tags.Add(other.tag);
            }
            int i = 0;
```

```
            foreach (string tag in LiquidTags)
            {
                if (other.tag == tag && absorve[i] > liquidAbsorve[i])
                {
                    liquidAbsorve[i]++;
                    break;
                }
                i++;
            }
        }
        else if (other.tag == mixedTag && special)
        {
            MixSpec++;

        }

    }
    void Update()
    {
        if (tags.Count >= LiquidTags.Count)
        {
            if (indexCheckTag < tags.Count)
            {
                checkForMixed = false;
                string tag = tags[indexCheckTag];
                if (!LiquidTags.Contains(tag))
                {
                    mix = false;
                }
                indexCheckTag++;
            }
            else if (indexCheckTag >= tags.Count)
            {
                indexCheckTag = 0;
                checkForMixed = true;
            }

            if (checkForMixed)
            {
                if (!mixed)
                {
                    if (cnt < liquidAbsorve.Length)
                    {
                        if (liquidAbsorve[cnt] >= absorve[cnt])
                        {
                            check++;
                        }
                        cnt++;
                    }
                    else
                    {
                        cnt = 0;
                        check = 0;
                    }
                    if ((check == liquidAbsorve.Length && tags.Count == LiquidTags.Count
&& mix) || (special && specialLimit <= MixSpec && mix))
                    {
                        mixed = true;
                        ps.tag = mixedTag;
                    }
                }
                else
                {
                    if ((check == liquidAbsorve.Length && tags.Count > LiquidTags.Count
&& !mix) || (!mix && special && (tags.Count < LiquidTags.Count)))
```

```
                    {
                        mixed = false;
                        ps.tag = "Untagged";
                    }
                }
            }
        }
    }

    public void ResetMix()
    {
        liquidAbsorve = new int[LiquidTags.Count];
        tags = new List<string>();
        checkForMixed = false;
        indexCheckTag = 0;
        mixed = false;
        mix = true;
        cnt = 0;
        check = 0;
        ps.tag = "Untagged";
    }

}


public class LightOn : MonoBehaviour
{
    [SerializeField] string TagForFire;
    ParticleSystem ps;
    MixingScript mix;
    [SerializeField] ObjectCheck aluminium;
    public bool Started = false;
    float temptime = 0;
    float time;
       [SerializeField] DelegateChange Task;

    void Start()
    {
        mix = GetComponent<MixingScript>();
        ps = gameObject.transform.parent.Find("Parent").GetComponent<ParticleSystem>();
        time = ps.main.duration;
    }

    void OnParticleCollision(GameObject other)
    {
        if (TagForFire == other.tag && !Started && mix.mixed && aluminium.added)
        {
            gameObject.transform.parent.Find("Parent").gameObject.SetActive(true);
            ps.Play();
            Started = true;
                    Task.AddTask();
        }
    }

    void Update()
    {
        if (Started && time > temptime && mix.mixed && aluminium.added)
        {
            temptime += Time.deltaTime;
        }
        else if(Started && time <= temptime && mix.mixed && aluminium.added)
        {
            Started = false;
            temptime = 0;
        }
```

```
    }

}


public class ScoreBoardDatabase : MonoBehaviour
{
    public static MySqlConnection MySqlConnetionRef;
    [SerializeField] string tableName;
    [SerializeField] bool isLocal = false;
    [SerializeField] int AllLevels = 0;
    [SerializeField] GameObject prefab;
    [SerializeField] GameObject loading;
    [SerializeField] float firstXCoordinate;
    [SerializeField] float firstYCoordinate;
    [SerializeField] float firstZCoordinate;
    [SerializeField] float offSet;
    MySqlConnection connection;
    float tempPosition;
    List<Data> allData;
    WriteReadFile file;
    bool oneTime = true;
    bool dataUpdated = false;
    Thread onlineData;
    Vector2 firstDimmensionsRect = Vector2.zero;
    bool usingSteam = false;

    private void OnEnable()
    {
        if (file == null)
        {
                file = new WriteReadFile("save.data");
        }
        if (isLocal)
        {
            dataUpdated = false;
            allData = new List<Data>();
            allData = file.ReadAllData();
            dataUpdated = true;
        }
        else
        {
            if (onlineData != null)
            {
                onlineData.Join();
            }
                onlineData = new Thread(DataUpdate);
                onlineData.Start();
                dataUpdated = false;
        }
        DestroyAllChild();
        loading.SetActive(true);
    }
    void DataUpdate()
    {
        allData = new List<Data>();
        string connectionInfo =
String.Format("server={0};user={1};database={2};password={3}", "162.144.18.183",
"r2tcj65r_ETL", "r2tcj65r_EscapeTheLab", "ETLScoreboard");
            if (connection == null && MySqlConnetionRef == null)
            {
                connection = new MySqlConnection(connectionInfo);
                MySqlConnetionRef = connection;
            }
            if (MySqlConnetionRef != null && connection == null)
```

```
            {
                connection = MySqlConnetionRef;
            }
            using (connection)
            {
                try
                {
                    connection.Open();
                    string get = String.Format("SELECT * FROM `{0}` WHERE Level={1} ORDER
BY Score DESC", tableName, AllLevels);
                    using (var command = new MySqlCommand(get, connection))
                    {
                        using (var reader = command.ExecuteReader())
                        {
                            while (reader.Read())
                            {
                                Data data = new Data(reader.GetDateTime(1),
reader.GetString(2), reader.GetFloat(3));
                                allData.Add(data);
                            }
                        }
                    }
                    connection.Close();
                }
                catch (Exception ex)
                {
                    Debug.Log(ex.ToString());
                }
            }
            if (usingSteam)
            {
                string name = SteamFriends.GetPersonaName();
                CSteamID steamId = SteamUser.GetSteamID();
                using (connection)
                {
                    try
                    {
                        connection.Open();
                        string get = String.Format("SELECT * FROM `SteamScoreBoard` WHERE
Level={0} AND OwnerKey={1} ORDER BY Score DESC", AllLevels, steamId);
                        using (var command = new MySqlCommand(get, connection))
                        {

                            using (var reader = command.ExecuteReader())
                            {

                                while (reader.Read())
                                {
                                    if (reader.GetString(2) != name)
                                    {
                                        String updateUsername = String.Format("UPDATE
SteamScoreBoard SET Usernam={0} WHERE OwnerKey={1}", name, steamId);
                                    }
                                }
                            }
                        }
                        connection.Close();
                    }
                    catch (Exception ex)
                    {
                        Debug.Log(ex.ToString());
                    }
                }
            }
            dataUpdated = true;
```

```
        onlineData.Abort();
    }
    void FillScoreBoard()
    {
        tempPosition = 0;
        for (int i = 0; i < allData.Count; i++)
        {
            if (i == 0)
            {
                RectTransform rect = gameObject.GetComponent<RectTransform>();
                GameObject List = Instantiate(prefab);
                rect.sizeDelta = firstDimmensionsRect;
                List.gameObject.SetActive(true);
                List.name = prefab.name + (i + 1).ToString();
                List.transform.SetParent(transform);
                List.GetComponent<RectTransform>().localPosition = new
Vector3(firstXCoordinate, firstYCoordinate, firstZCoordinate);
                List.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);
                List.GetComponent<RectTransform>().localRotation = Quaternion.Euler(0, 0,
0);
                tempPosition = firstYCoordinate;
                List.transform.Find("Place").GetComponent<TextMeshProUGUI>().text = (i +
1).ToString();
                List.transform.Find("Date").GetComponent<TextMeshProUGUI>().text =
allData[i].DatePlay.ToString("yyyy/MM/dd");
                List.transform.Find("User").GetComponent<TextMeshProUGUI>().text =
allData[i].NickName;
                List.transform.Find("Score").GetComponent<TextMeshProUGUI>().text =
allData[i].Score.ToString();
            }
            if (i > 0)
            {
                RectTransform rect = gameObject.GetComponent<RectTransform>();
                rect.sizeDelta = new Vector2(rect.sizeDelta.x, rect.sizeDelta.y +
Math.Abs(offSet));
                GameObject List = Instantiate(prefab);
                List.gameObject.SetActive(true);
                List.name = prefab.name + (i + 1).ToString();
                List.transform.SetParent(transform);
                float yPosition = tempPosition + offSet;
                List.GetComponent<RectTransform>().localPosition = new
Vector3(firstXCoordinate, yPosition, firstZCoordinate);
                List.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);
                List.GetComponent<RectTransform>().localRotation = Quaternion.Euler(0, 0,
0);
                tempPosition = yPosition;
                List.transform.Find("Place").GetComponent<TextMeshProUGUI>().text = (i +
1).ToString();
                List.transform.Find("Date").GetComponent<TextMeshProUGUI>().text =
allData[i].DatePlay.ToString("yyyy/MM/dd");
                List.transform.Find("User").GetComponent<TextMeshProUGUI>().text =
allData[i].NickName;
                List.transform.Find("Score").GetComponent<TextMeshProUGUI>().text =
allData[i].Score.ToString();
            }
        }
    }
    void Start()
    {
        usingSteam = SteamManager.Initialized;
        RectTransform rect = gameObject.GetComponent<RectTransform>();
        firstDimmensionsRect = rect.sizeDelta;
    }
    void Update()
    {
```

```
        if (GameData.End == true && oneTime && GameEnd.Inserted)
        {
            if (isLocal)
            {
                dataUpdated = false;
                allData = new List<Data>();
                allData = file.ReadAllData();
                dataUpdated = true;
            }
            else
            {
                if (onlineData != null)
                {
                    onlineData.Join();
                }
                onlineData = new Thread(DataUpdate);
                onlineData.Start();
                dataUpdated = false;
            }
            loading.SetActive(true);
            DestroyAllChild();
            oneTime = false;
        }
        if (dataUpdated)
        {
            FillScoreBoard();
            loading.SetActive(false);
            dataUpdated = false;
        }
    }
    void DestroyAllChild()
    {
        int i = 0;
        foreach (Transform objects in transform)
        {
            if (i > 1)
            {
                Destroy(objects.gameObject);
            }
            i++;
        }
    }
}


public class Teleports : MonoBehaviour
{
    TimeLeft timeleft;
    [SerializeField] bool loadPreviuos = false;
    [SerializeField] int loseSceneIndex = 0;
    bool saveFile = false;

    private void Start()
    {
        timeleft = GetComponent<TimeLeft>();
    }

    // Update is called once per frame
    void Update()
    {
        if (GameData.End == true && GameData.Victory == false)
        {
            if (saveFile == false)
            {
                GlobalData.WriteToFile();
```

```
                        GlobalData.ResetStatic();
                        BoundryController.ResetCount();
                        saveFile = true;
                }
                GameData.SetPreviuosScene(SceneManager.GetActiveScene().buildIndex);
                loseScene();
                GameData.Cear();
            }
        }

        void loseScene()
        {
            SceneManager.LoadScene(loseSceneIndex);
        }

        void restartLevel()
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        }

        void LoadPreviuos()
        {
            SceneManager.LoadScene(GameData.PreviuosSceneId);
        }
    }



    public class DontDestroy : MonoBehaviour
    {
        private static DontDestroy instance;
        private AudioSource aud;

        public AudioClip menuMusic;
        public AudioClip gameMusic;

        private void Start()
        {
            aud = GetComponent<AudioSource>();
        }

        private void Awake()
        {
            if (instance != null && instance != this)
            {
                Destroy(this.gameObject);
            }
            else
                instance = this;

            DontDestroyOnLoad(gameObject);
        }

        void OnEnable()
        {
            SceneManager.sceneLoaded += OnSceneLoaded;
        }

        void OnSceneLoaded(Scene scene, LoadSceneMode mode)
        {
            if (scene.name == "MainMenu")
            {
                if (aud == null)
                    return;
```

```
            aud.Stop();
            aud.clip = menuMusic;
            aud.Play();
        }
        else if (scene.name == "MainGame")
        {
            if (aud == null)
                return;
            aud.Stop();
            aud.clip = gameMusic;
            aud.Play();
        }
    }

}


public class DropSound : MonoBehaviour
{
    int cnt = 0;
    AudioSource aud;
    // Start is called before the first frame update
    void Start()
    {
        aud = GetComponent<AudioSource>();
    }

    public void PlaySound()
    {
        if (!aud.isPlaying)
        {
            aud.Play();
        }
    }
}



public class GrabbingSound : MonoBehaviour
{
    bool isGrabbed = false;
    bool play = true;
    AudioSource sound;
    // Start is called before the first frame update
    void Start()
    {
        sound = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {

        if (isGrabbed && play)
        {
            sound.Play();
            play = false;
        }
        else if (!isGrabbed)
        {
            play = true;
        }
    }
```

```
    public void Grab()
    {
        isGrabbed = true;
    }
    public void UnGrab()
    {
        isGrabbed = false;
    }
}


public class SmashSound : MonoBehaviour
{
    BottleSmash smash;
    bool smashed = false;
    AudioSource aud;
    // Start is called before the first frame update
    void Start()
    {
        smash = GetComponentInParent<BottleSmash>();
        aud = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {
        if (smash.broken && !smashed)
        {
            aud.Play();
            smashed = true;
        }
    }
}
```