

KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B166 Development of Computer Games and Interactive Applications

Escape the Lab VR

Done by:
Eligijus Kiudys
IFF-7/14

Date:
2020-02-17

Kaunas, 2019

Tables of Contents

<i>Tables of Images</i>	3
<i>Table of Tables/functions</i>	4
<i>Work Distribution Table:</i>	5
<i>Description of Your Game</i>	6
<i>Laboratory work #1</i>	7
List of tasks	7
<i>Solution</i>	7
Task #1. Create second level.....	7
Task #2. Implement Virtual Reality player movement (in this case, teleportation)	13
Task #3. Decorate the level with at least 20 GameObjects, 5 Lights and 5 materials	16
Task #4. Make a new GameObject, give it a Collider, use onTriggerEnter to track when the Player touches it, then destroy it and print out a message that says “Player touched me”	18
Task #5. Create flask breaking onCollisionEnter	19
Task #6. Create a GameObject and make that object starts hover by pressing spacebar and pressing space second time it come down (defense task)	22
<i>Laboratory work #2</i>	25
Task #7. Experiment with Polybrush / SpriteShapes / ProBuilder / Tilemap Palette tools.	25
<i>Literature list</i>	45
<i>ANNEX</i>	46

Tables of Images

Figure 1. Second level scene with some used prefabs below	8
Figure 2. Objects inside of a shelf	9
Figure 3. Timer trigger	10
Figure 4. Correct teleportation	14
Figure 5. Incorrect teleportation	14
Figure 6. VRTK Teleportation hierarchy	15
Figure 7. Main teleportation component inspector	15
Figure 8. Scene decoration with various GameObjects	16
Figure 9. Objects used in the scene #1	16
Figure 10. Objects used in the scene #2	17
Figure 11. Objects used in the scene #3	17
Figure 12. Lamp model #1	17
Figure 13. Lamp model #2	18
Figure 14. Capsule standign still	23
Figure 15. Hovering capsule	23
Figure 16. Created terrain	25
Figure 17 Created terrain	25
Figure 18 Open hands	26
Figure 19 Using grip	26
Figure 20 Using index finger with trigger	26
Figure 21 Running flask	27
Figure 22 Flickering lights	27
Figure 23 Driving car	28
Figure 24 Water particles	28
Figure 25 Liquid chemical particles	29
Figure 26 Flask smash particle	29
Figure 27 Outlet particles	30
Figure 28 Dry ice reaction	30
Figure 29 Editor woth open scene with night skybox	31
Figure 30 Bouncy pen	31
Figure 31 bouncines material	31
Figure 32 Flask with friction	32
Figure 33 Friction material	32
Figure 34 Slipery mouse	32
Figure 35 Slipery material	32
Figure 36 Objects set to static	37
Figure 37 Objcets set to static	37
Figure 38 Camera settings	38
Figure 39 In game statistics	38
Figure 40 profiler results	39
Figure 41 Baked light map	39
Figure 42 profiler results with baked light map	40
Figure 43 Material exampl	40
Figure 44 Material example	41
Figure 45 Material example	41
Figure 46 Frame debugger	42
Figure 47 Low quality	42
Figure 48 High Quality	43
Figure 49 Ultra Quality	43

Table of Tables/functions

Table 1. Timeleft.cs script component.....	11
Table 2. StopTime.cs script component.....	11
Table 3. LockUnlockWithKey.cs script component.....	13
Table 4. destroyontouch script component	18
Table 5. BottleSmah.cs code	22
Table 6. Hovering.cs script.....	24
Table 7 Liquid absorber code	34
Table 8 On trigger enter note.....	35
Table 9 OnTrigger enter timer.....	35
Table 10 Doors on trigger enter.....	36

Work Distribution Table:

<i>Name/Surname</i>	<i>Description of game development part</i>
<i>Airidas Janonis</i>	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
<i>Eligijus Kiudys</i>	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 Your Game

Description of Your 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

Laboratory work #1

List of tasks

1. Create second level
2. Implement Virtual Reality player movement (in this case, teleportation)
3. Decorate the level with at least 20 GameObjects, 5 Lights and 5 materials
4. Make a new GameObject, give it a Collider, use onTriggerEnter to track when the Player touches it, then destroy it and print out a message that says "Player touched me"
5. Create flask breaking onCollisionEnter
6. Create a GameObject and make that object starts hover by pressing spacebar and pressing space second time it come down (defense task)

Solution

Task #1. Create second level

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 1. Second level scene with some used prefabs below

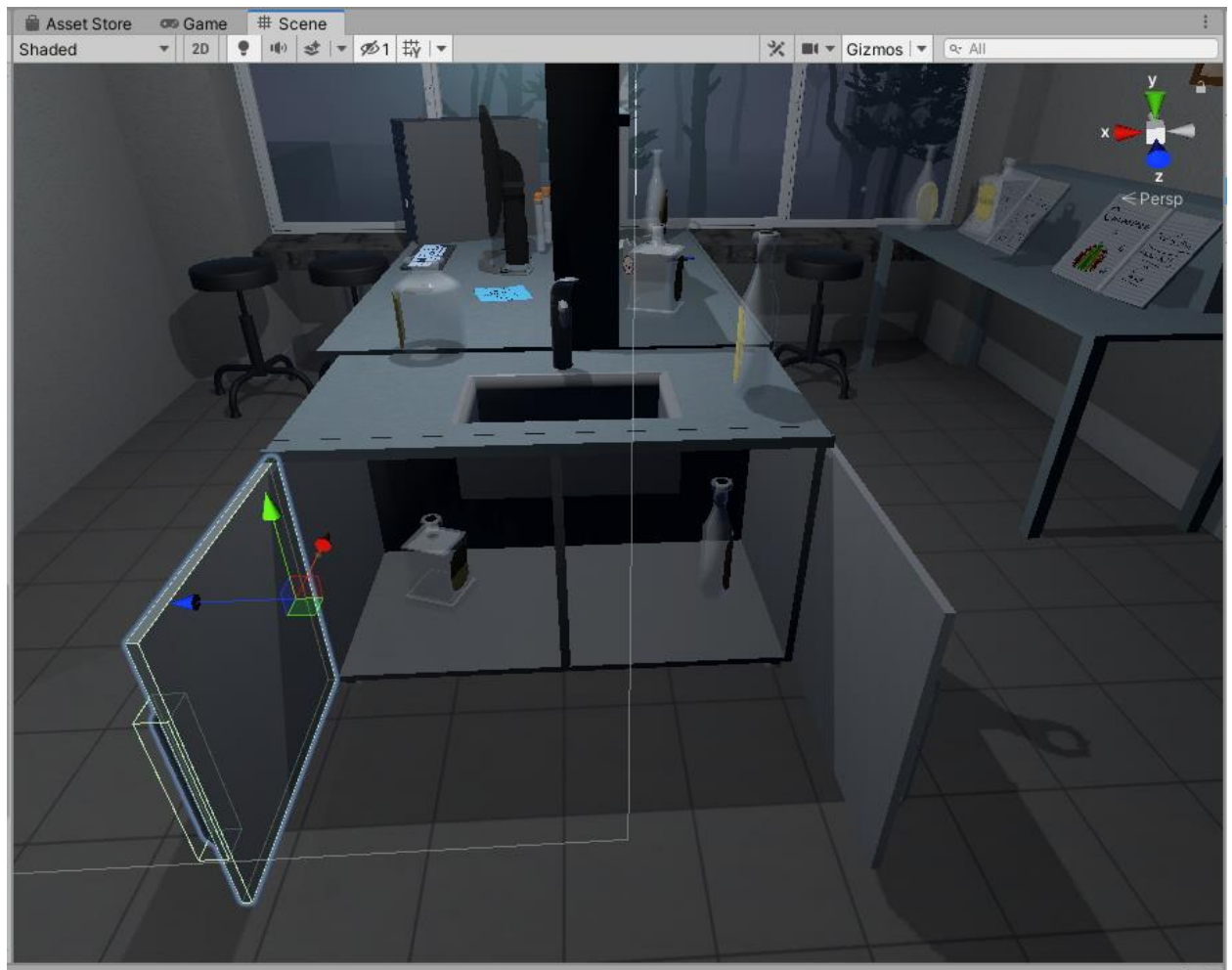


Figure 2. Objects inside of a shelf

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 3. 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 1. 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 2. StopTime.cs script component

Door unlocking workflow:

To pass the level, player has to preforme two experiments to get 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 3. LockUnlockWithKey.cs script component

Task #2. Implement Virtual Reality player movement (in this case, teleportation)

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.

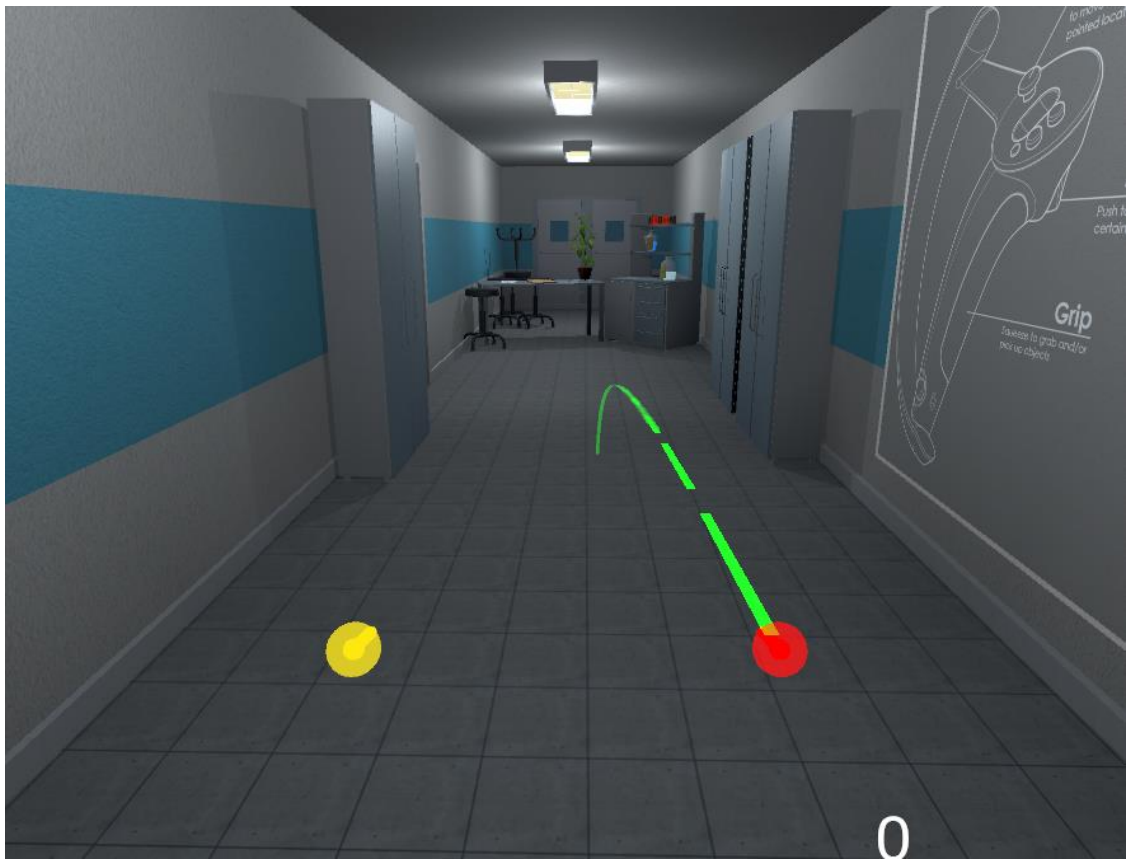


Figure 4. Correct teleportation

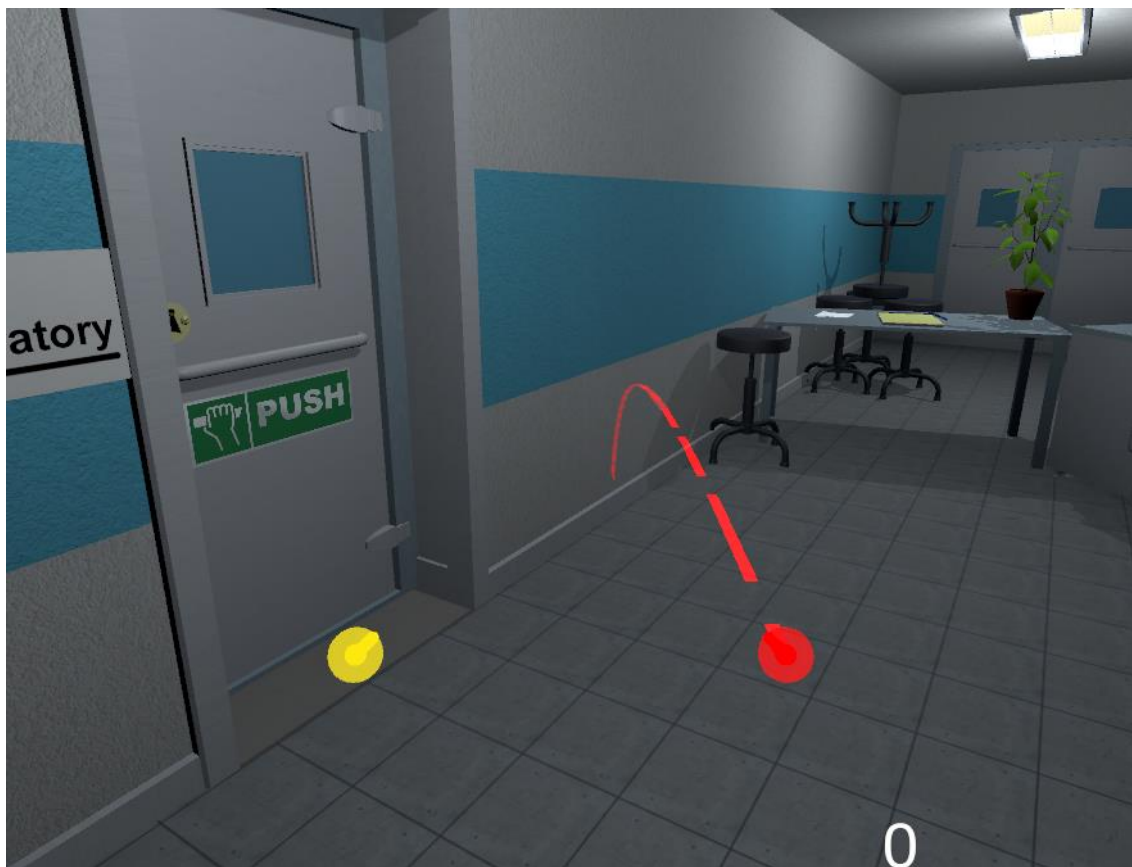


Figure 5. Incorrect teleportation

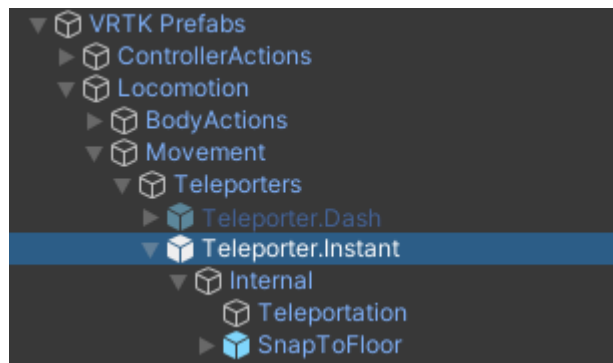


Figure 6. VRTK Teleportation hierarchy

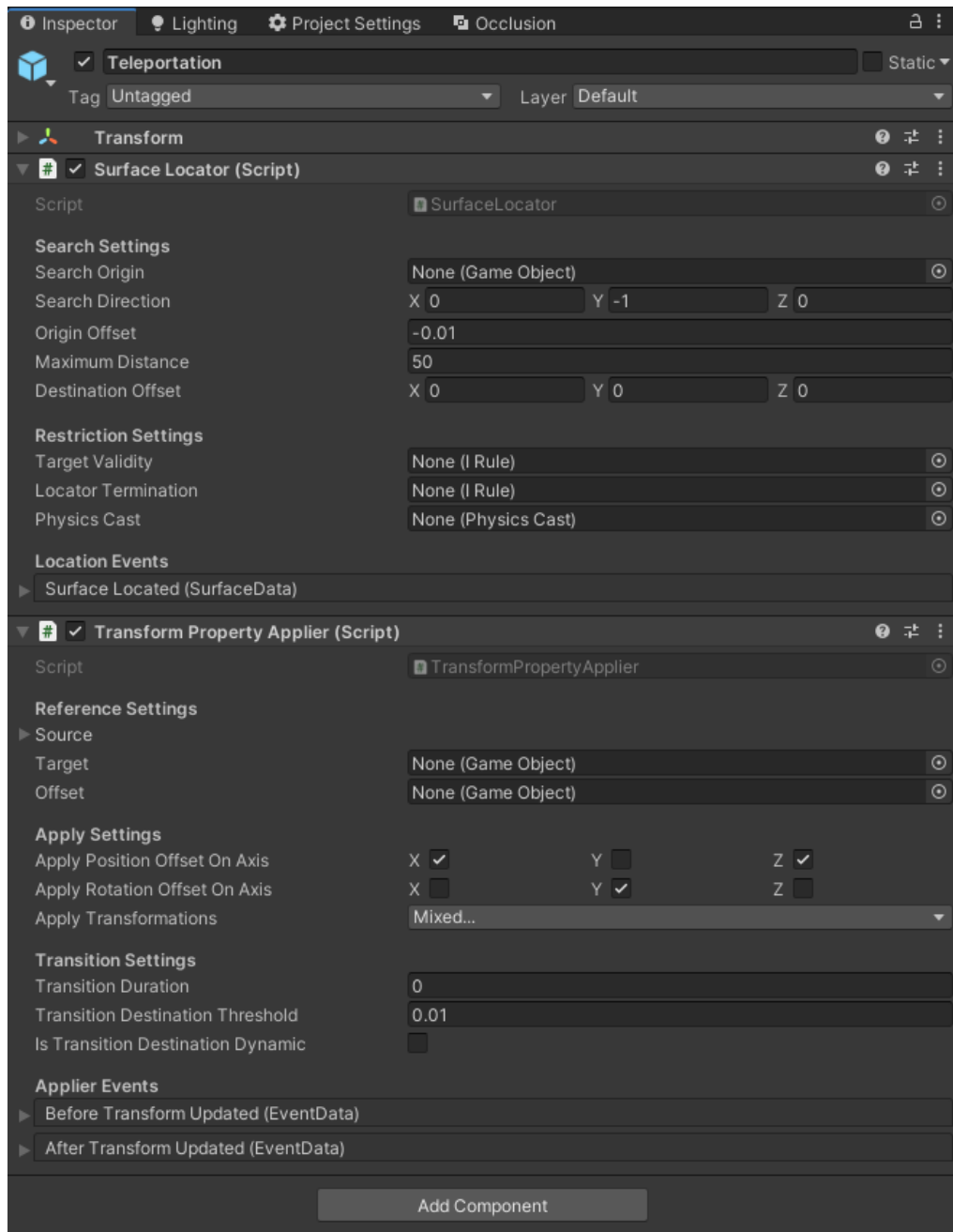


Figure 7. Main teleportation component inspector

Task #3. Decorate the level with at least 20 GameObjects, 5 Lights and 5 materials

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 8. Scene decoration with various GameObjects

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

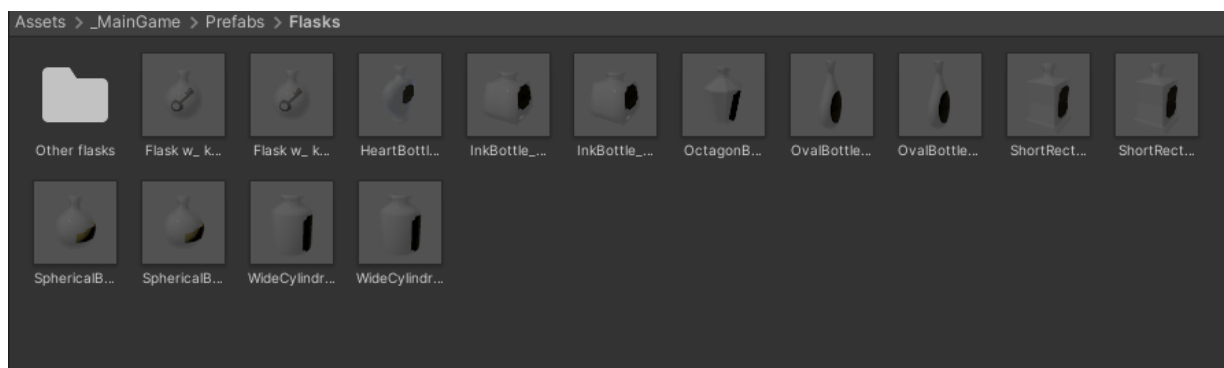


Figure 9. Objects used in the scene #1

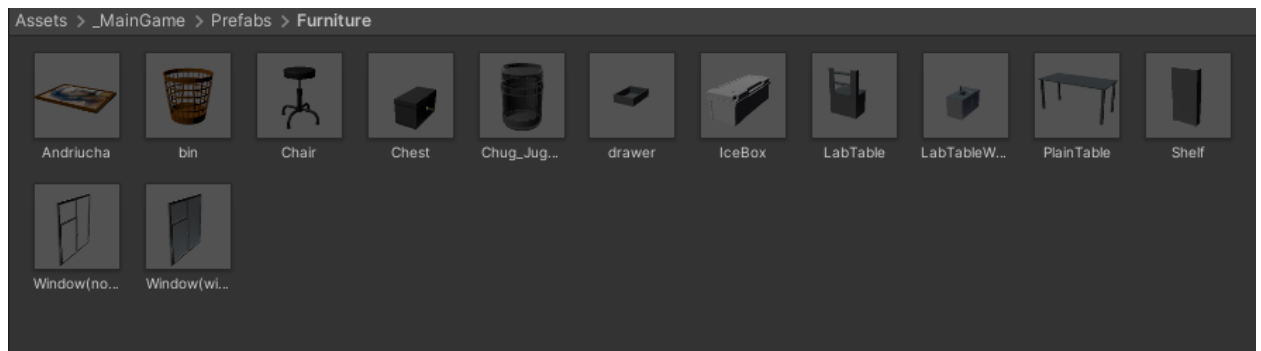


Figure 10. Objects used in the scene #2

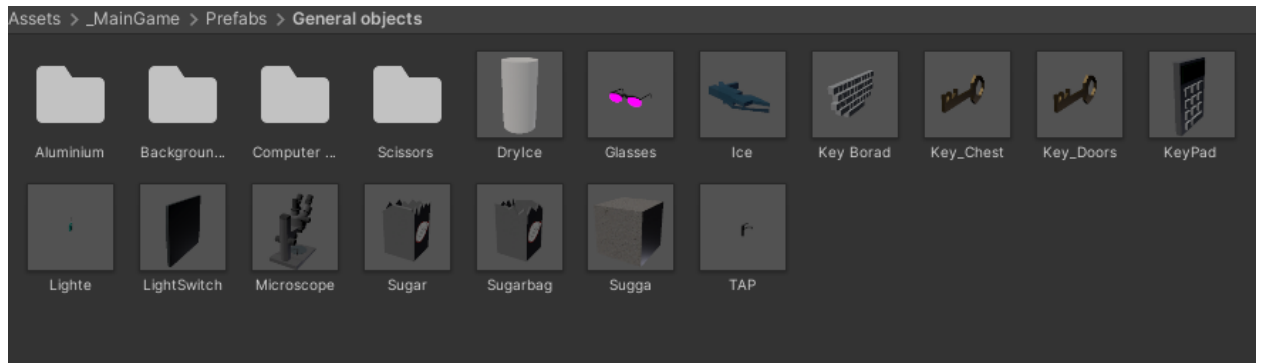


Figure 11. Objects used in the scene #3

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

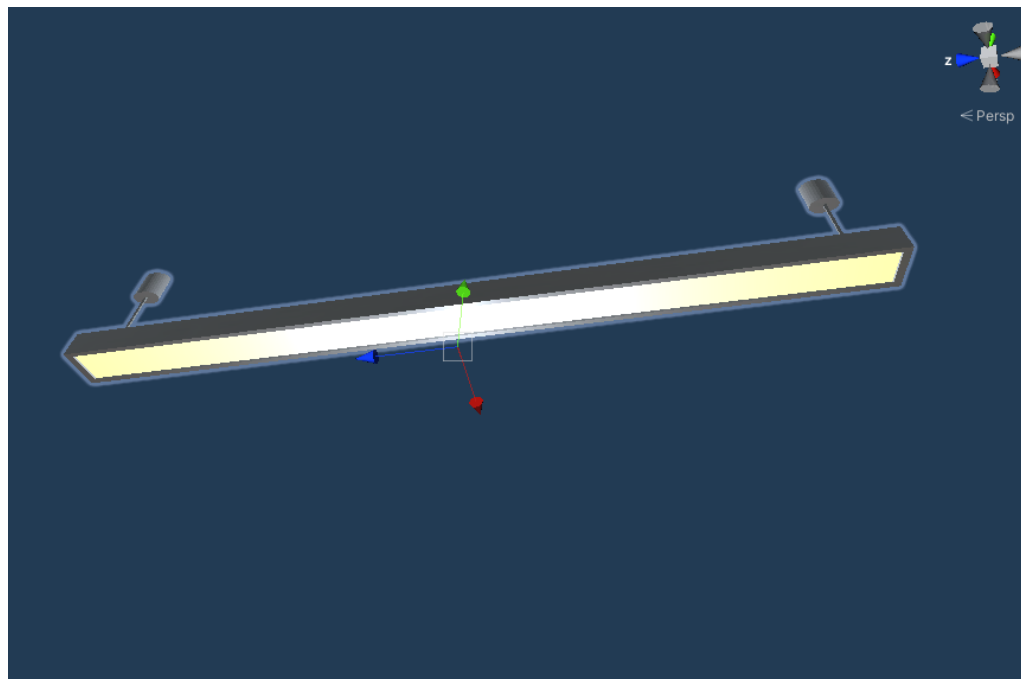


Figure 12. Lamp model #1

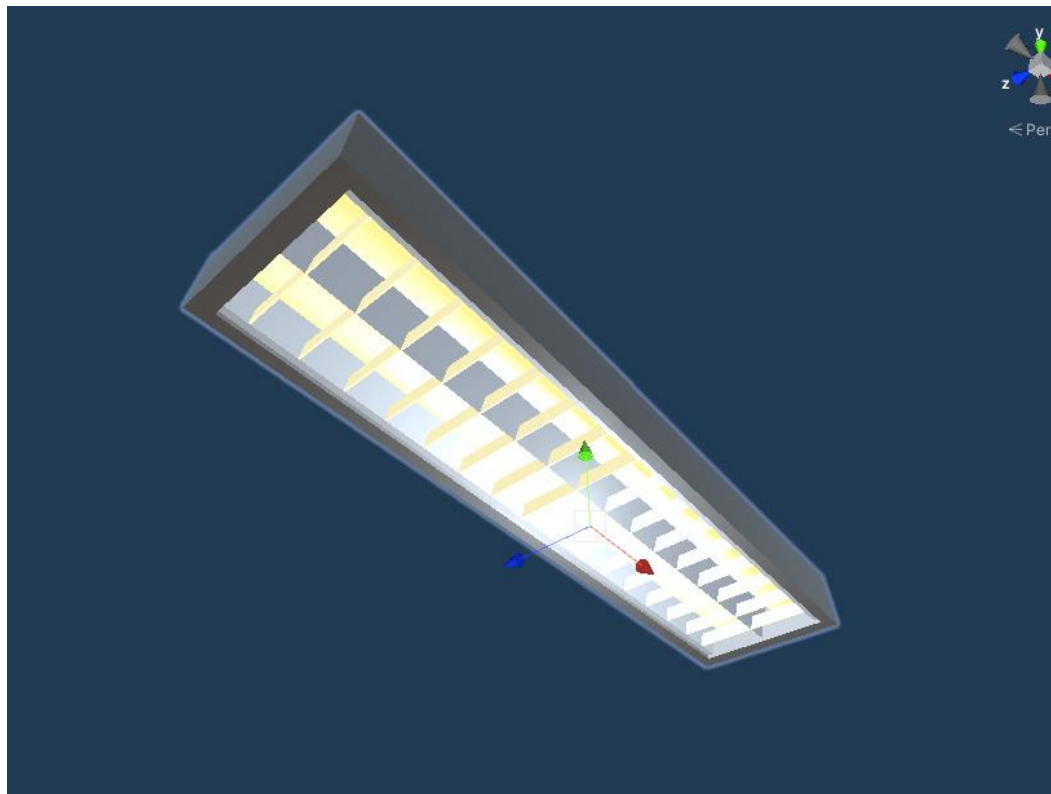


Figure 13. Lamp model #2

Task #4. Make a new GameObject, give it a Collider, use `onTriggerEnter` to track when the Player touches it, then destroy it and print out a message that says “Player touched me”

Implemented the functionality by adding `DestroyOnTouch.cs` script component into GameObject. If this GameObject collides with another GameObject, which has a layer of 10 (which is a player’s Hand), the “Player has touched me” message gets printed out and the object destroys itself.

```
public class destroyontouch : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.layer == 10)
        {
            Debug.Log("Player has touched me!");
            Destroy(this.gameObject);
        }
    }
}
```

Table 4. destroyontouch script component

Task #5. Create flask breaking onCollisionEnter

Implemented the functionality by adding BottleSmash.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;

SteamVrSkeleton skeleton;

void Start () {
    if (Liquid != null)
    {
        lva = Liquid.GetComponent<LiquidVolumeAnimator>();
    }
    skeleton = GetComponent<SteamVrSkeleton>();
    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 5. BottleSmah.cs code

Task #6. Create a GameObject and make that object starts hover by pressing spacebar and pressing space second time it come down (defense task)

Created a capsule GameObject, gave it rigidbody and collider components. To implement the jumping mechanic, a Defense.cs script was written and given to the GameObject.



Figure 14. Capsule standign still



Figure 15. Hovering capsule

The hovering works by adding force up to capsule when space is pressed first time. When capsule reached specific point rigidbody is changed to kinematic. When space is pressed second time rigidbody kinematic is set false.

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

public class Hovering : MonoBehaviour
{
    Rigidbody rb;
    float startPos = 0;
    float difference = 0;
```

```

[SerializeField] float height = 0.1f;
bool isHovering = true;
// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody>();
    startPos = rb.worldCenterOfMass.y;
}

// Update is called once per frame
void Update()
{
    if (isHovering)
    {
        if (difference == 0 || difference < height)
        {
            rb.isKinematic = false;
            rb.AddForce(Vector3.up * 5f, ForceMode.Impulse);
        }
        difference = rb.worldCenterOfMass.y - startPos;
        if (difference > height)
        {
            rb.isKinematic = true;
        }
    }

    if (Input.GetKeyDown(KeyCode.Space) && isHovering)
    {
        isHovering = false;
        rb.velocity = Vector3.zero;
        rb.isKinematic = false;
    }
    else if (Input.GetKeyDown(KeyCode.Space) && !isHovering)
    {
        isHovering = true;
    }
}
}

```

Table 6. Hovering.cs script

Laboratory work #2

Task #7. Experiment with Polybrush / SpriteShapes / ProBuilder / Tilemap Palette tools.

Created main scene terrain with polybrush and unity terrain tool.

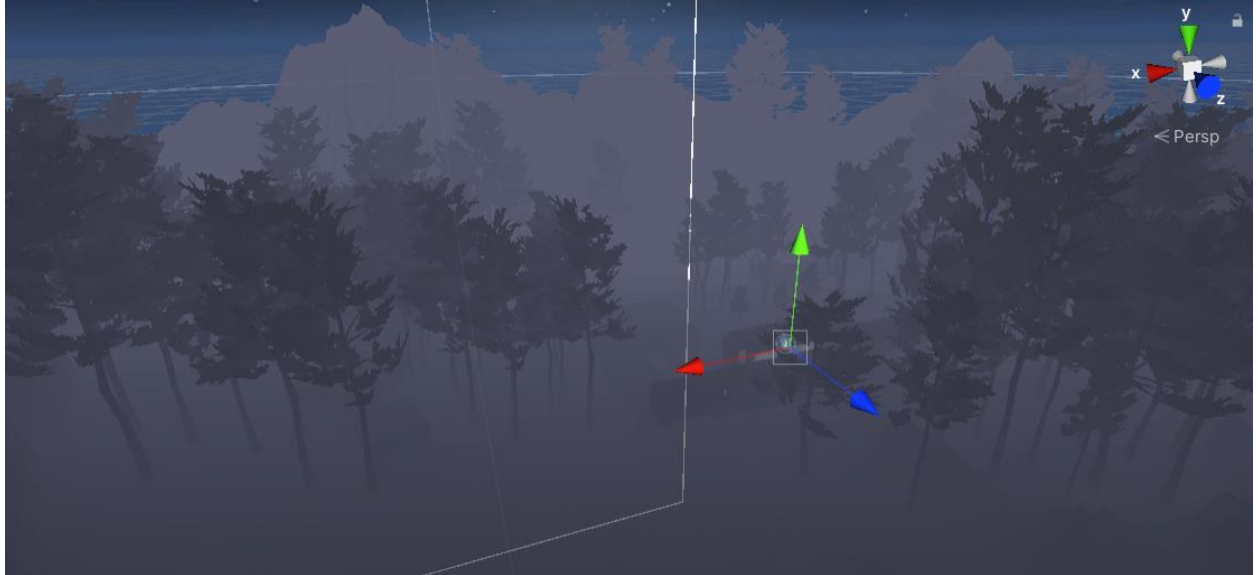


Figure 16. Created terrain

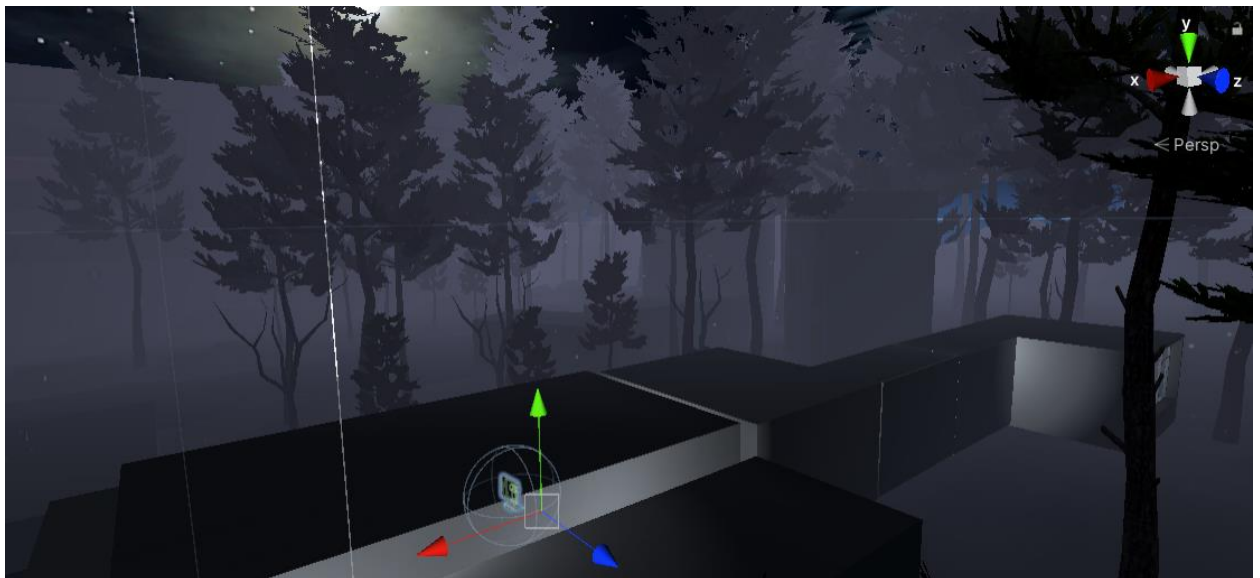


Figure 17 Created terrain

In this scene I was using polybrush for hills and trees

Task #8 Add animations to your game character (Assuming you have a working PlayerController).

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 18 Open hands



Figure 19 Using grip



Figure 20 Using index finger with trigger

Task #9. Create and/or animate 5 objects of your choice in your World (you can use Animator Component and/or Timeline).

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.

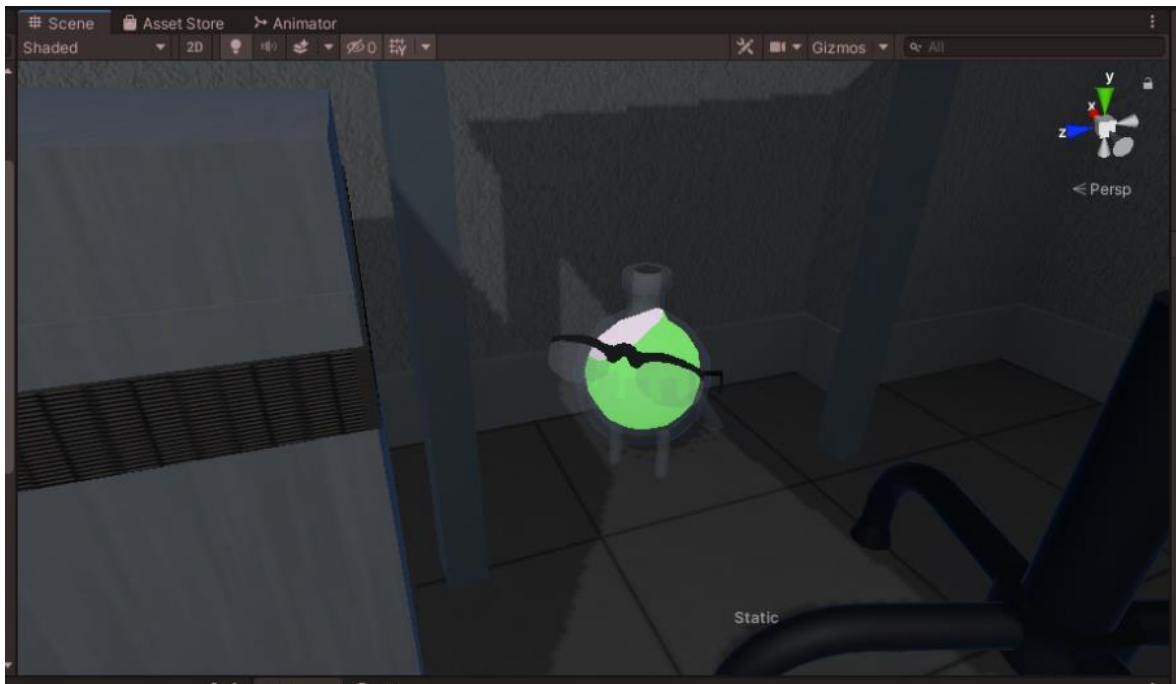


Figure 21 Running flask

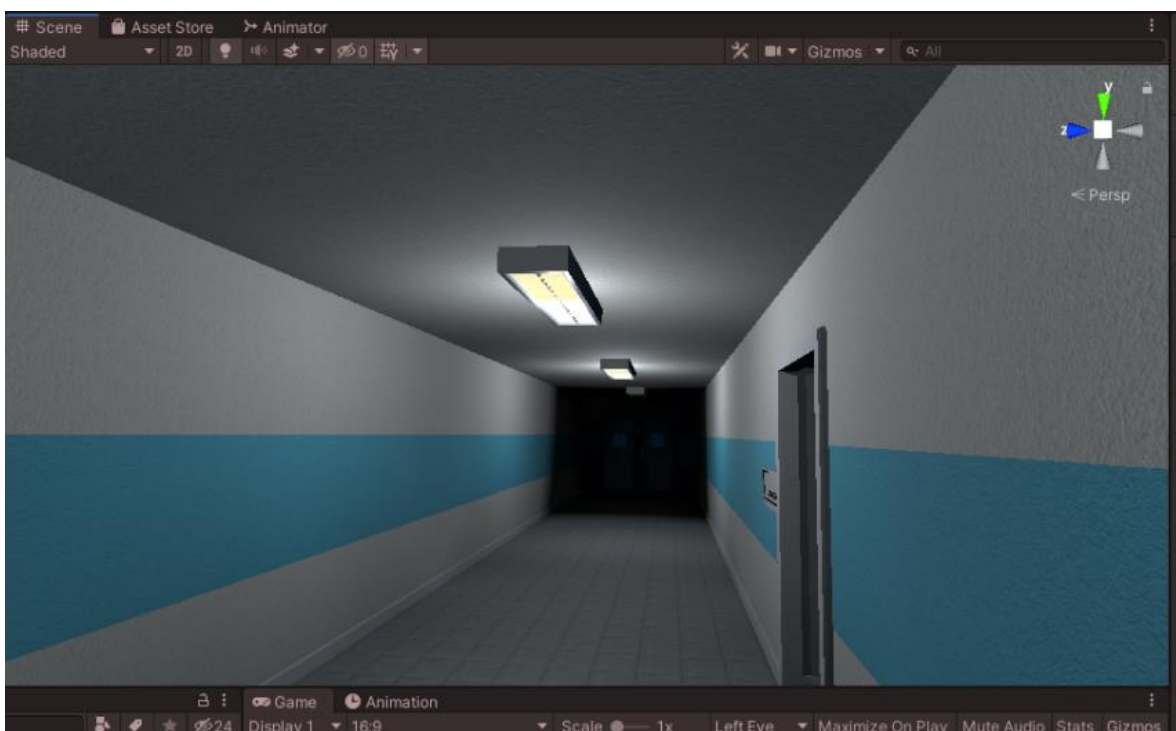


Figure 22 Flickering lights

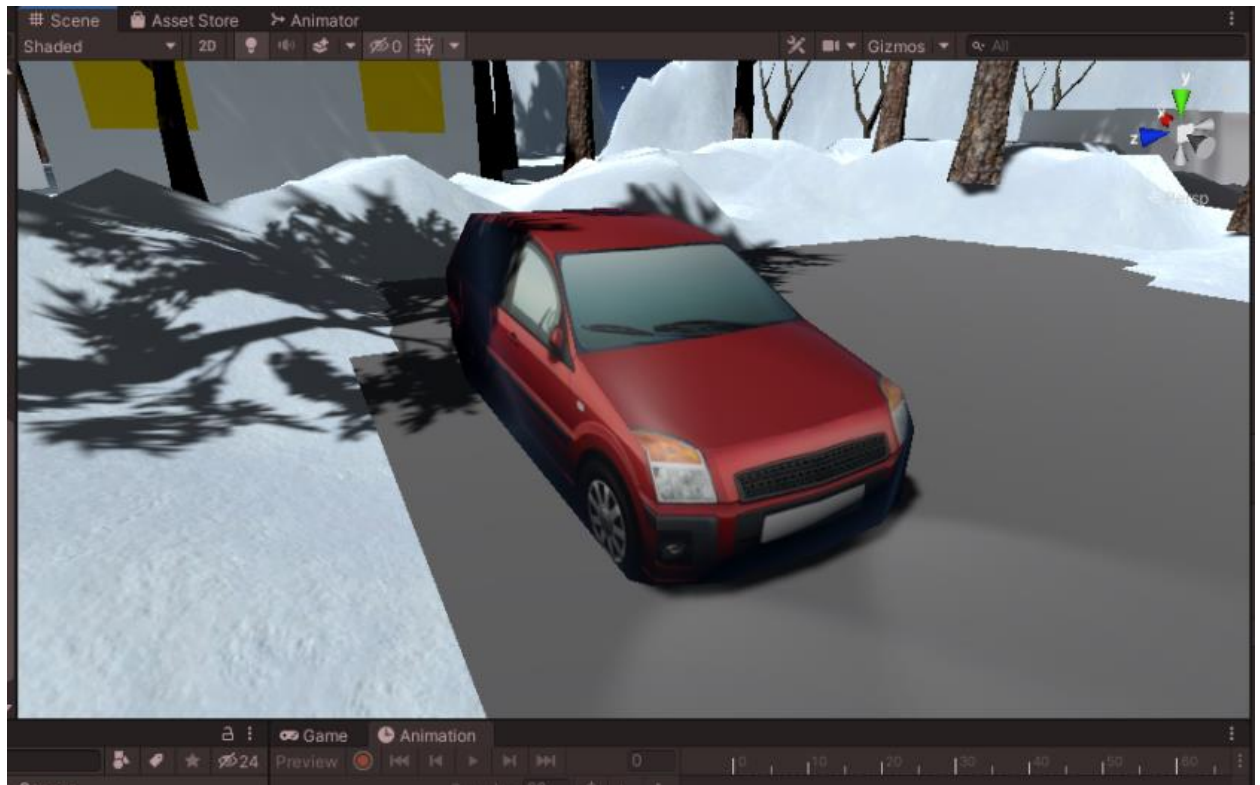


Figure 23 Driving car

Task #10. Create at least 5 particle effects for your environment (dust, explosion, smoke gas, light sparks, etc.).

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.

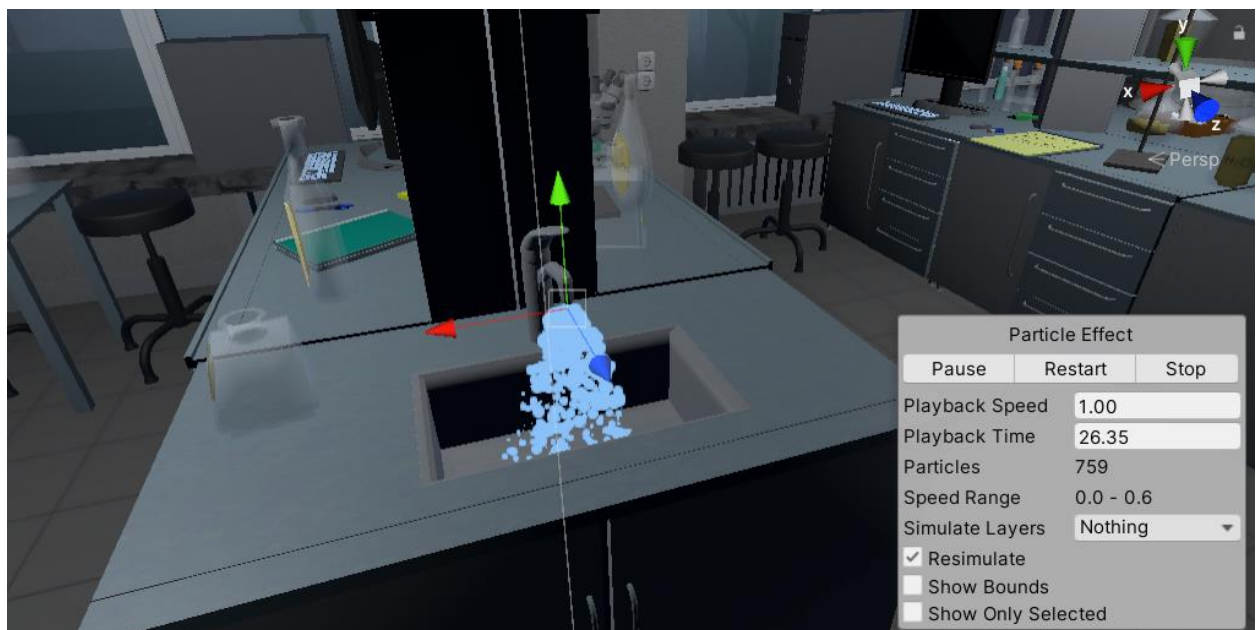


Figure 24 Water particles



Figure 25 Liquid chemical particles

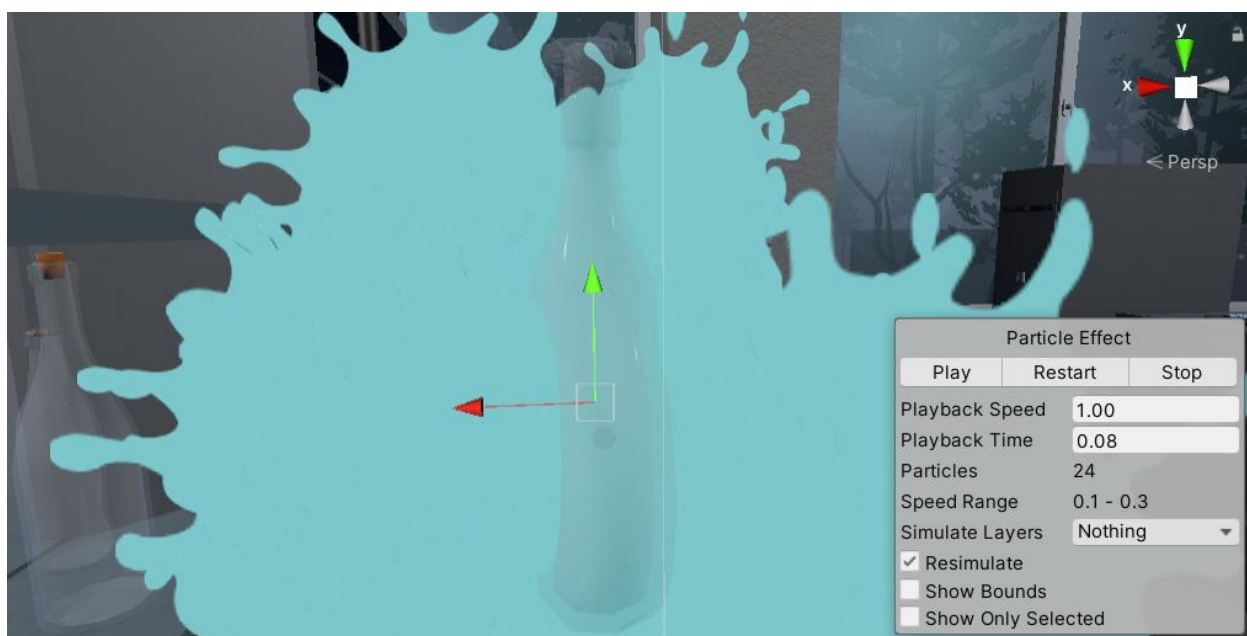


Figure 26 Flask smash particle

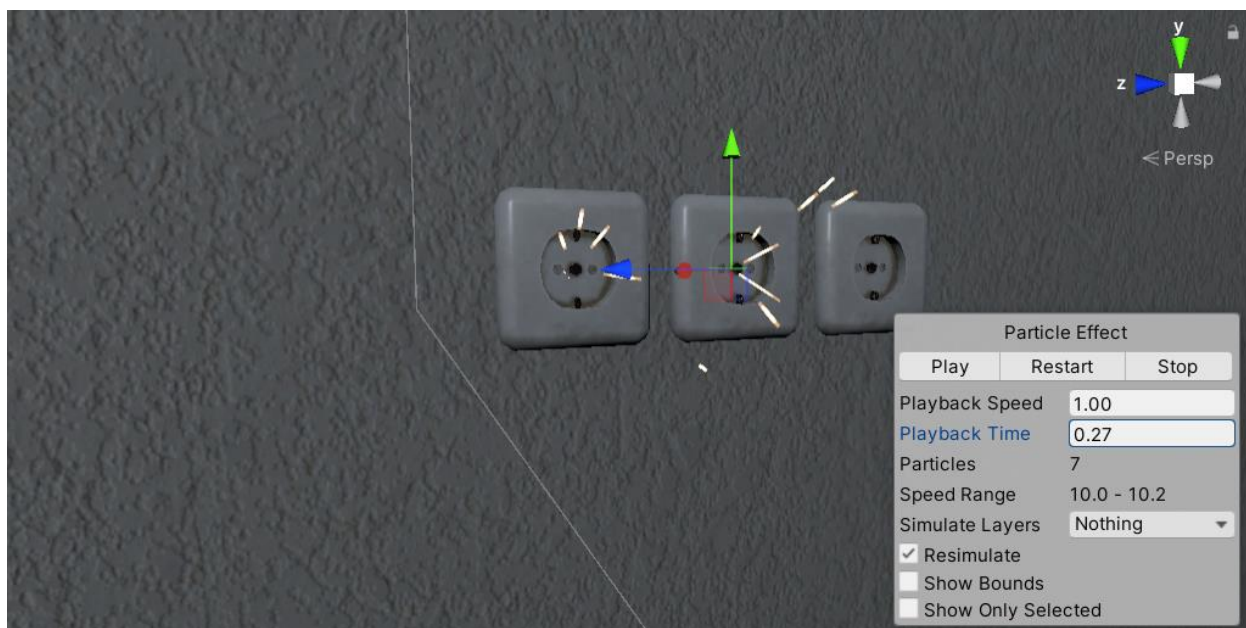


Figure 27 Outlet particles

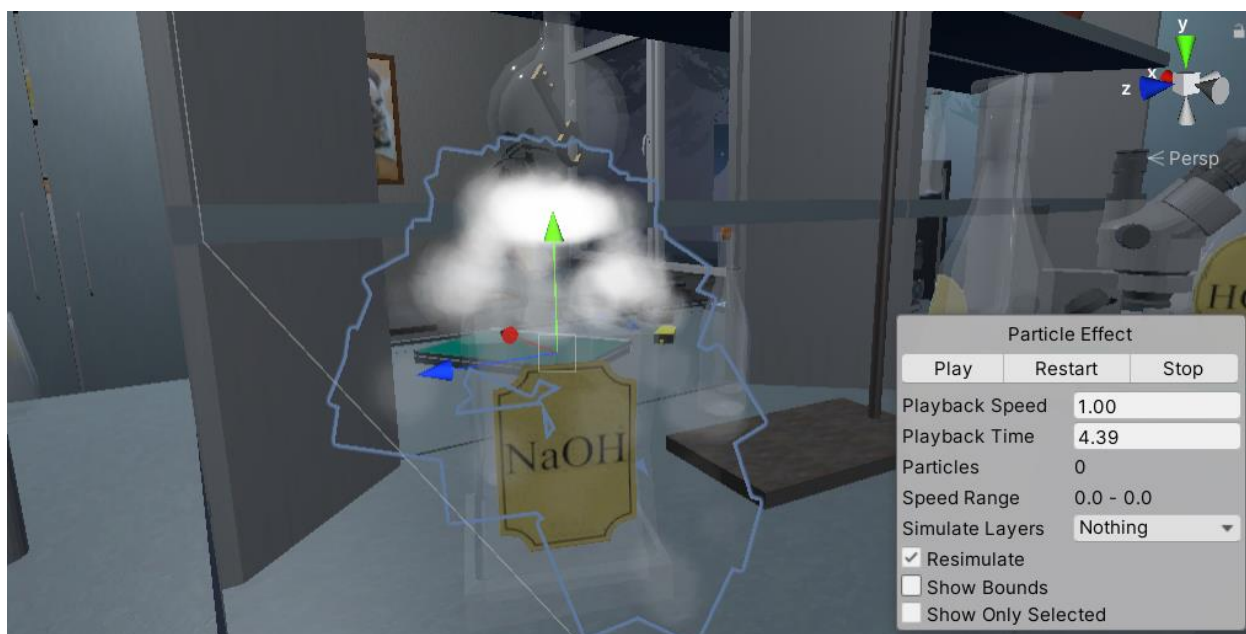


Figure 28 Dry ice reaction

Task #11. Add a custom skybox.

This project contains several skyboxes, this is one of them.

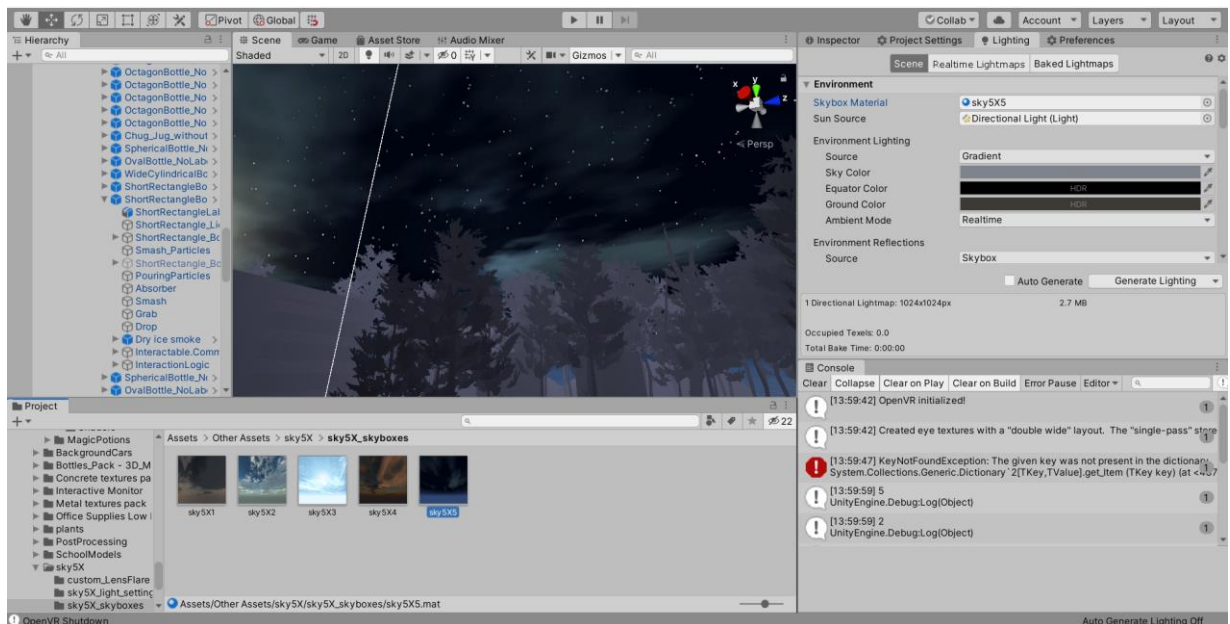


Figure 29 Editor with open scene with night skybox

Task #12. 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.).

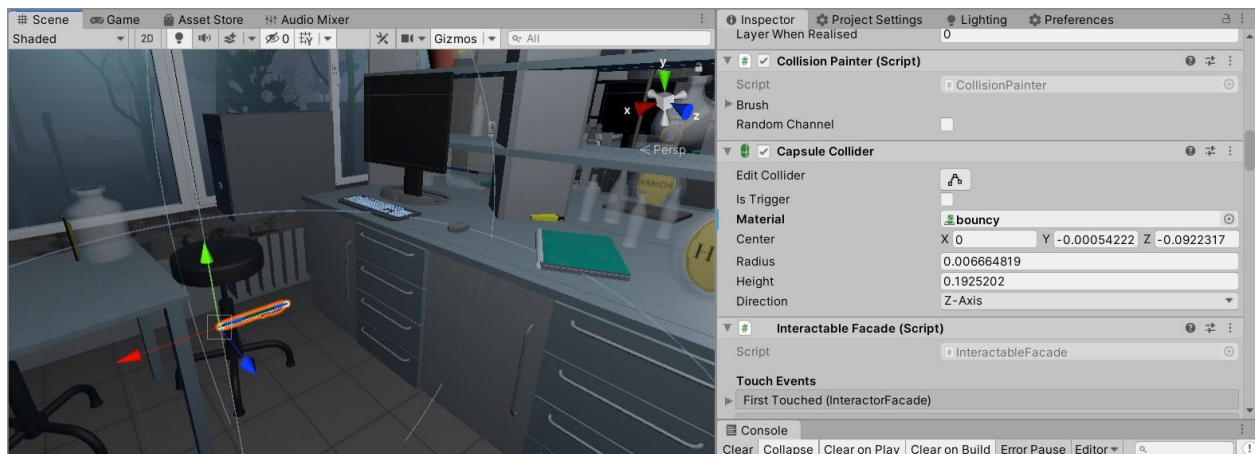


Figure 30 Bouncy pen

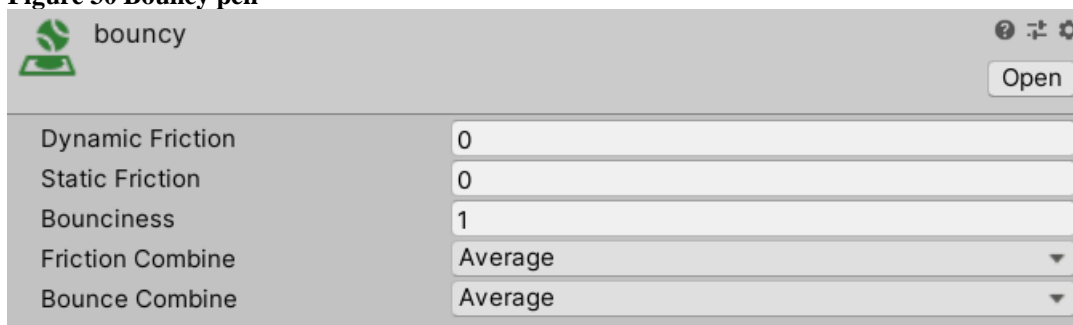


Figure 31 bouncines material



Figure 32 Flask with friction

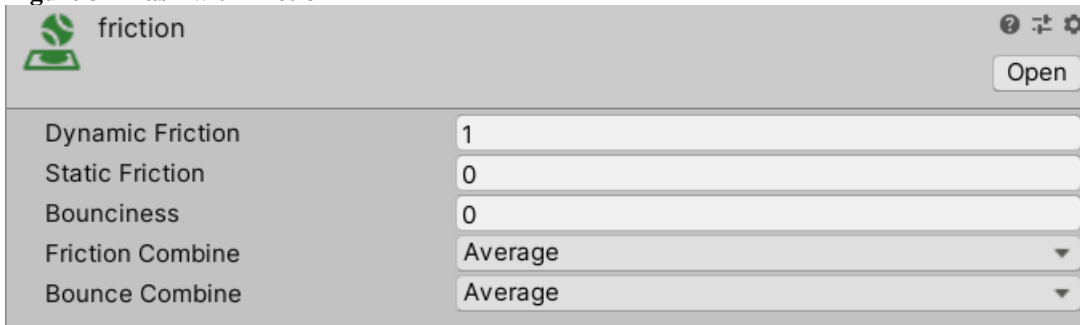


Figure 33 Friction material

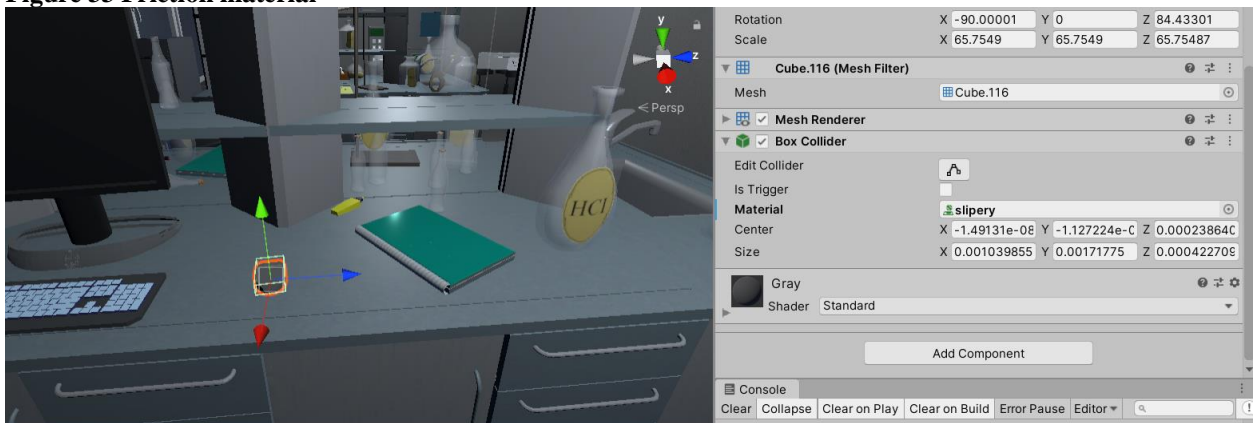


Figure 34 Slipery mouse

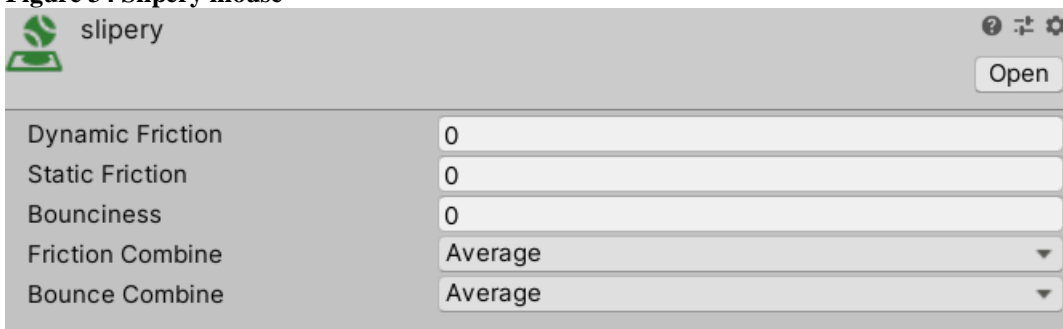


Figure 35 Slipery material

Task #13. Create 4 types of objects that use OnCollisionEnter / OnCollisionEnter2D / OnTriggerEnter / OnTriggerEnter2D.

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

- 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("_TintColor");
    }

    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 7 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 8 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 9 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 10 Doors on trigger enter

Task #14. Assign optimal colliders for the environment objects that are not moving and set their flags to static (test performance before and after).

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.

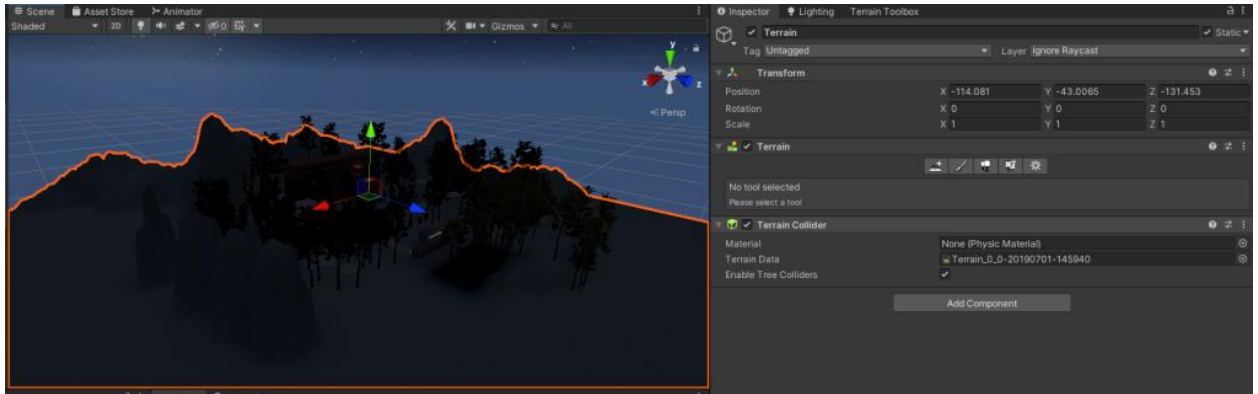


Figure 36 Objects set to static

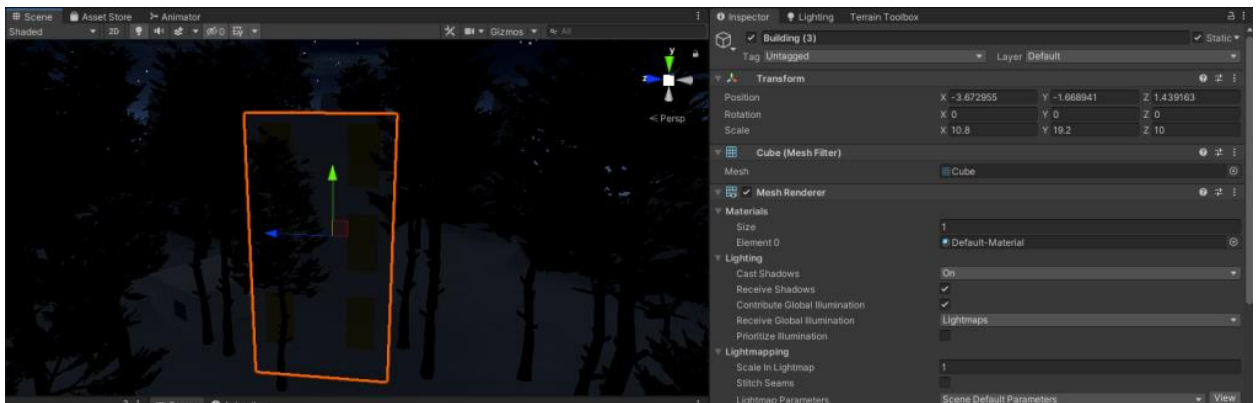


Figure 37 Objects set to static

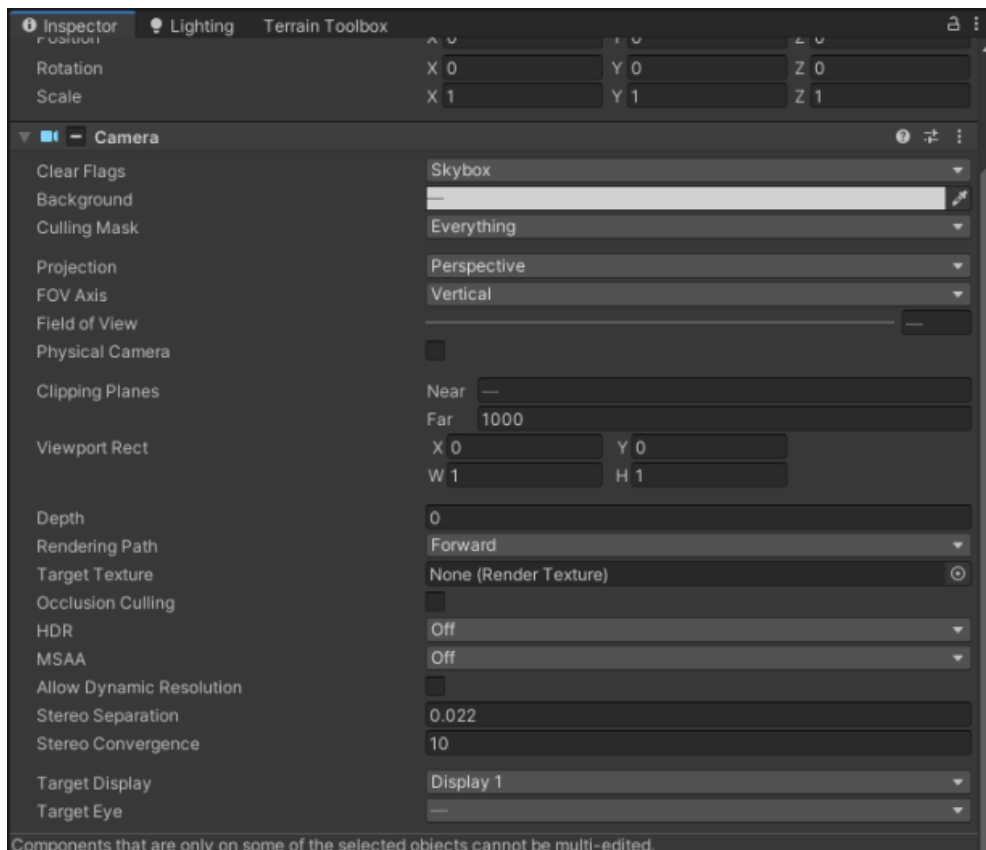


Figure 38 Camera settings

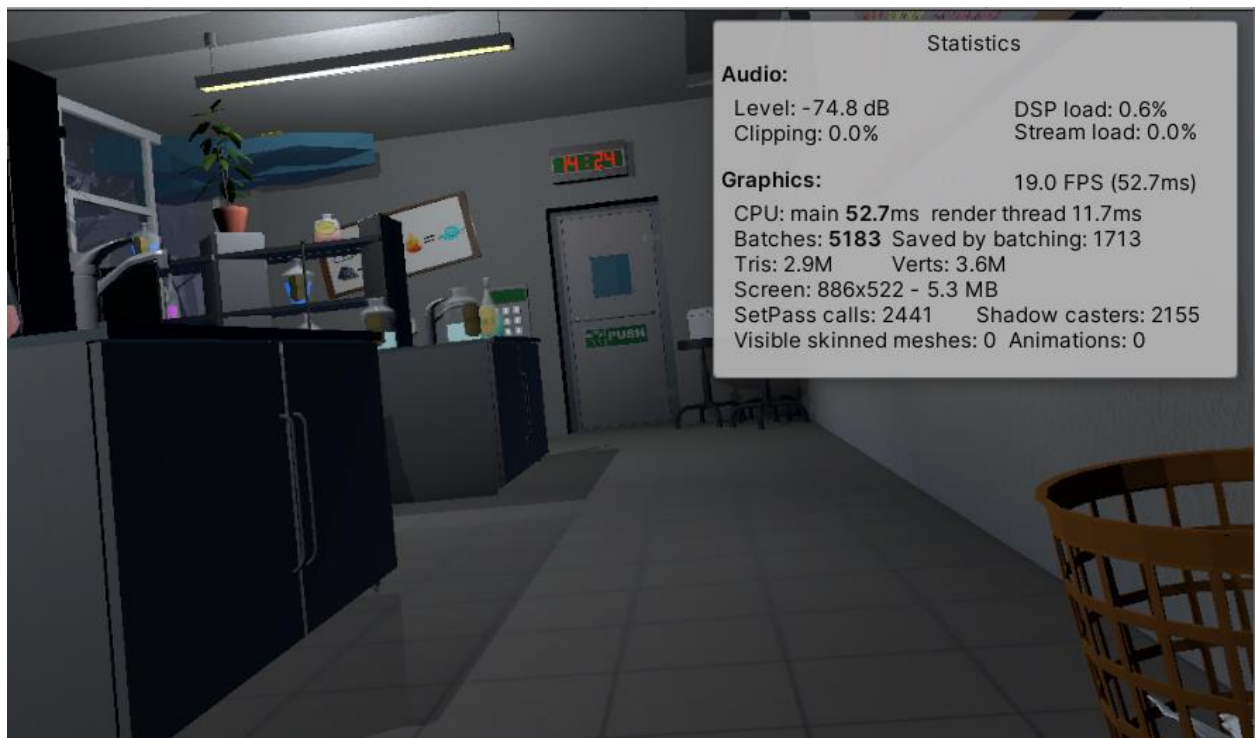


Figure 39 In game statistics

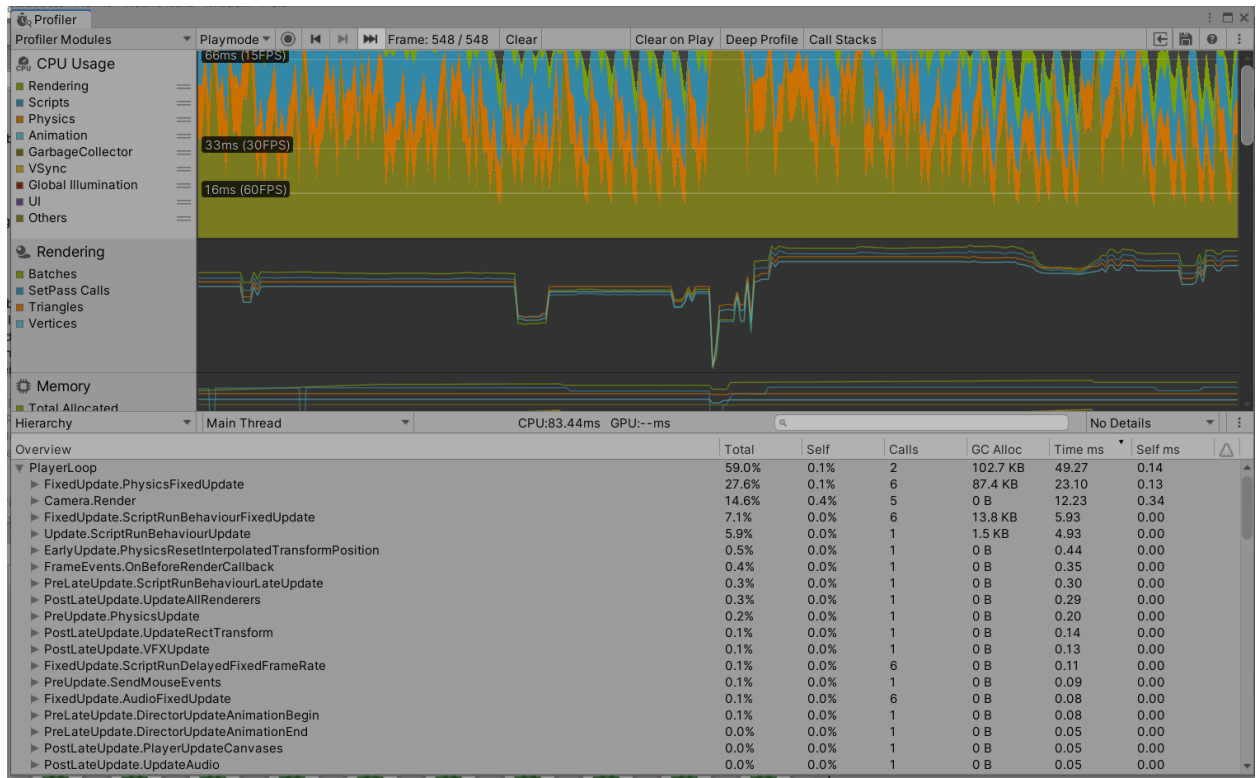


Figure 40 profiler results

Task #15. Bake a lightmap and measure performance.

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.

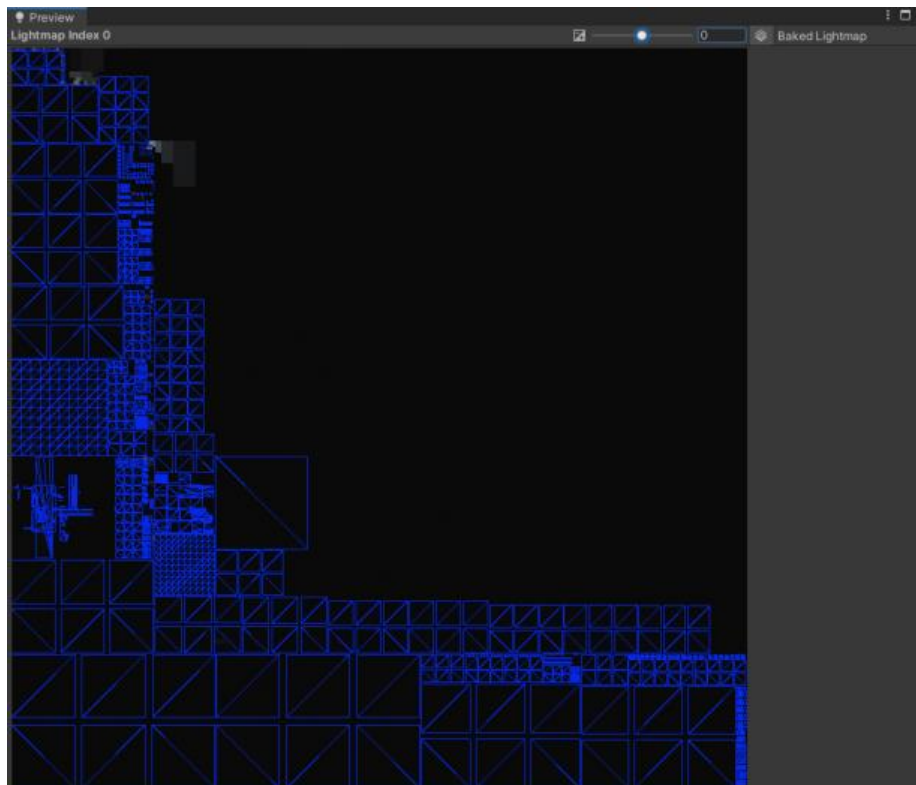


Figure 41 Baked light map

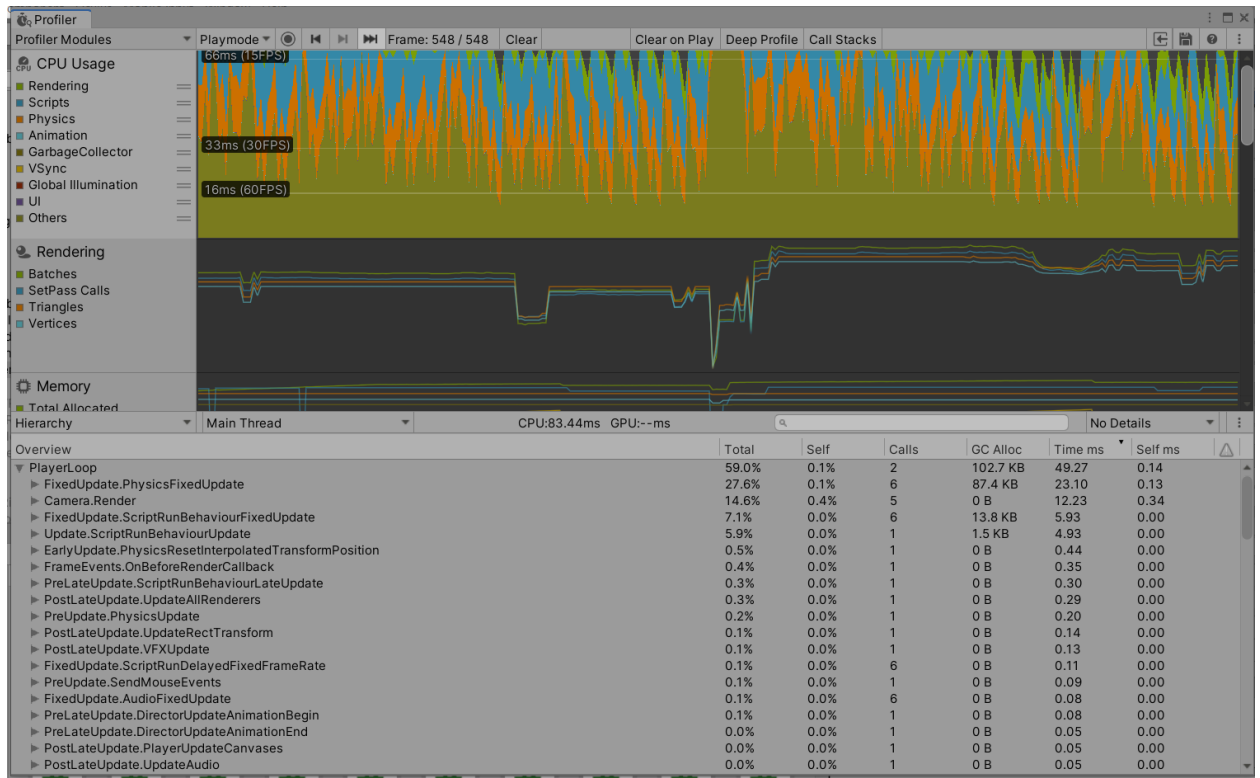


Figure 42 profiler results with baked light map

Task #16. Optimize all textures depending on their parameters and measure graphical memory load.

Using textures is optimized, texture resolution is decreased. For texture depth we using normal map. Gpu instancing is enabled.

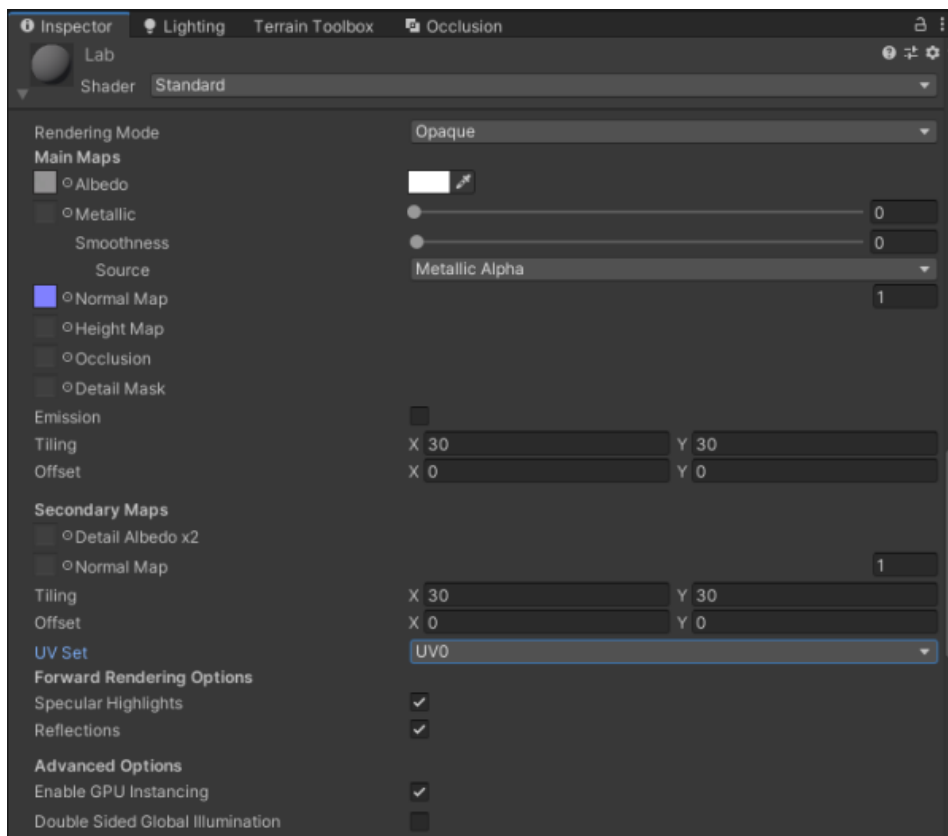


Figure 43 Material example

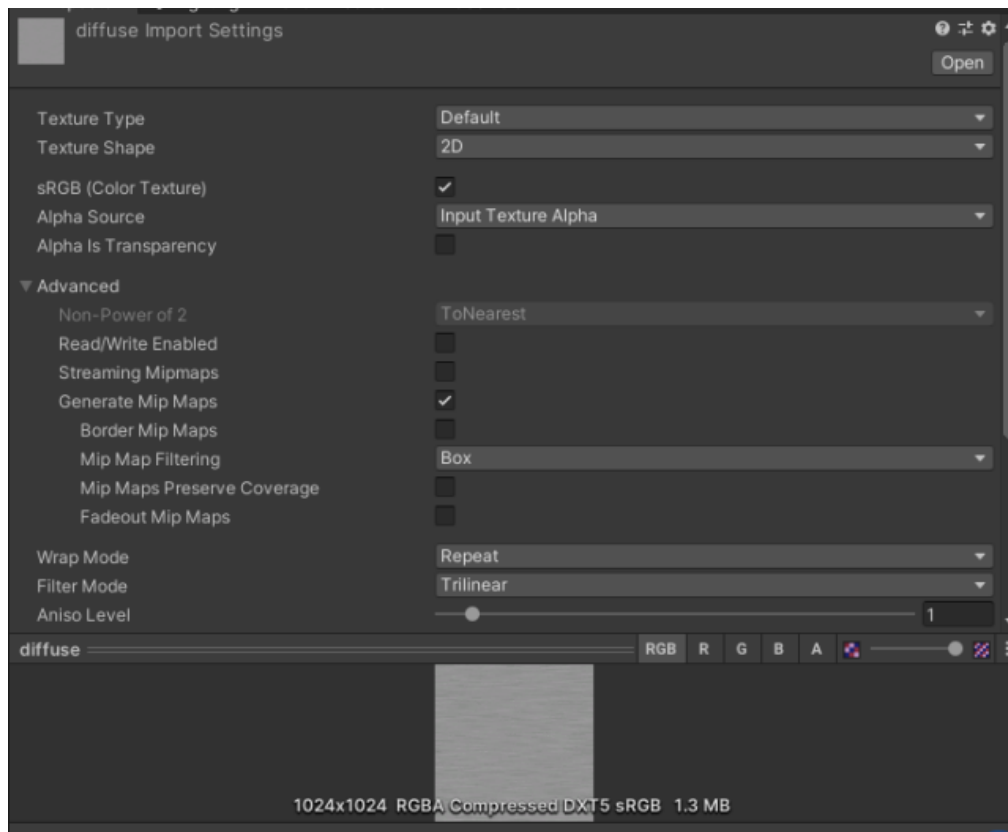


Figure 44 Material example

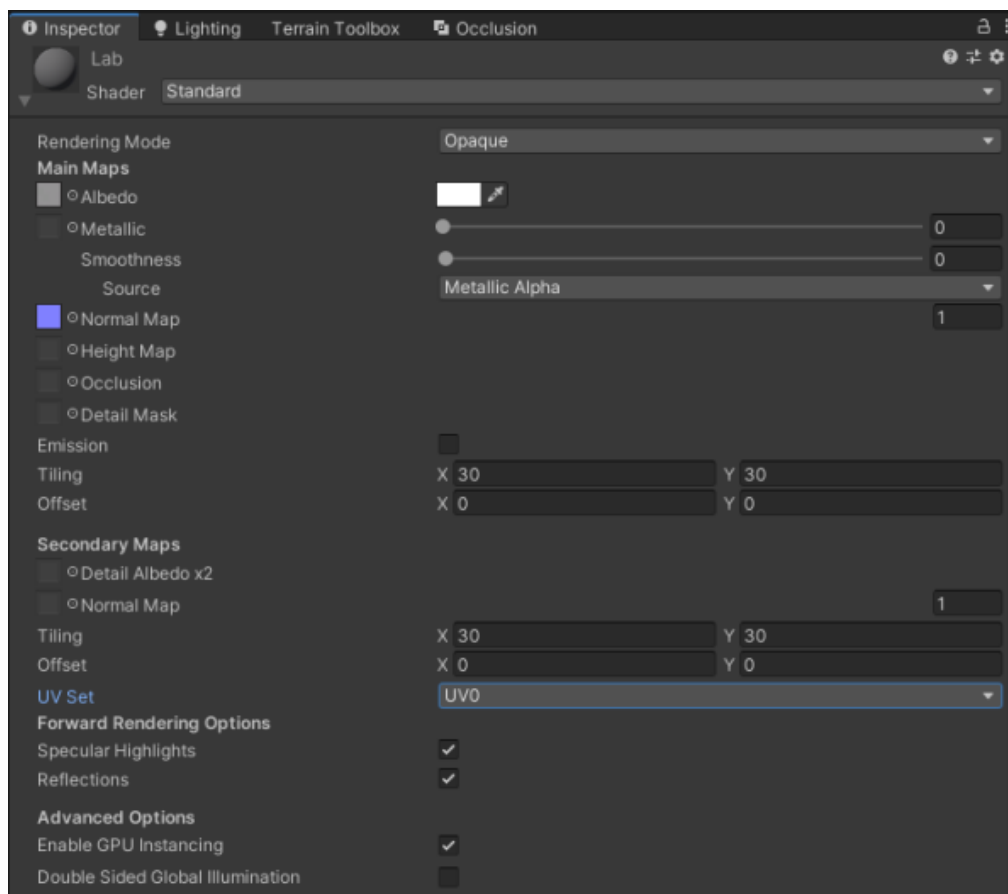


Figure 45 Material example

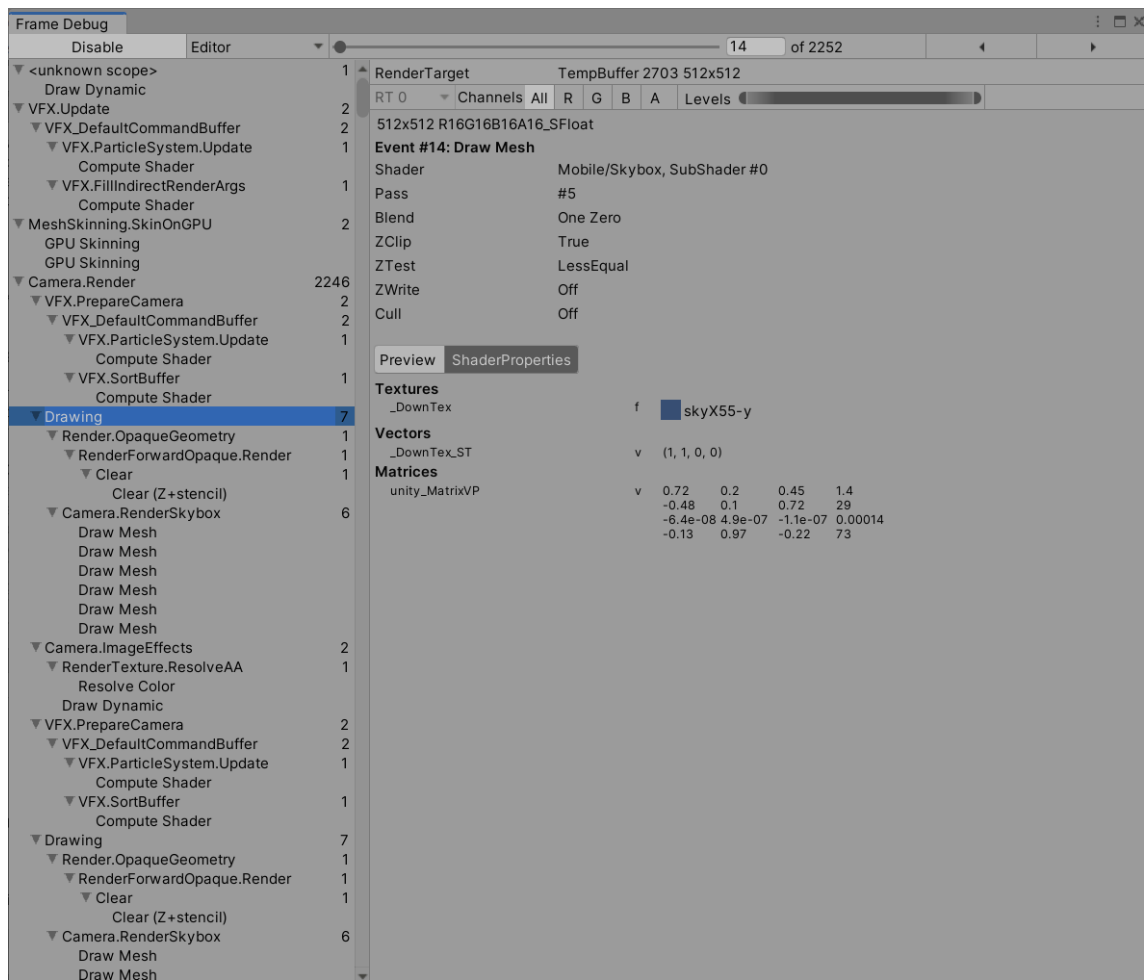


Figure 46 Frame debugger

Task #17. Try hard vs soft shadows, different quality settings and measure the performance.

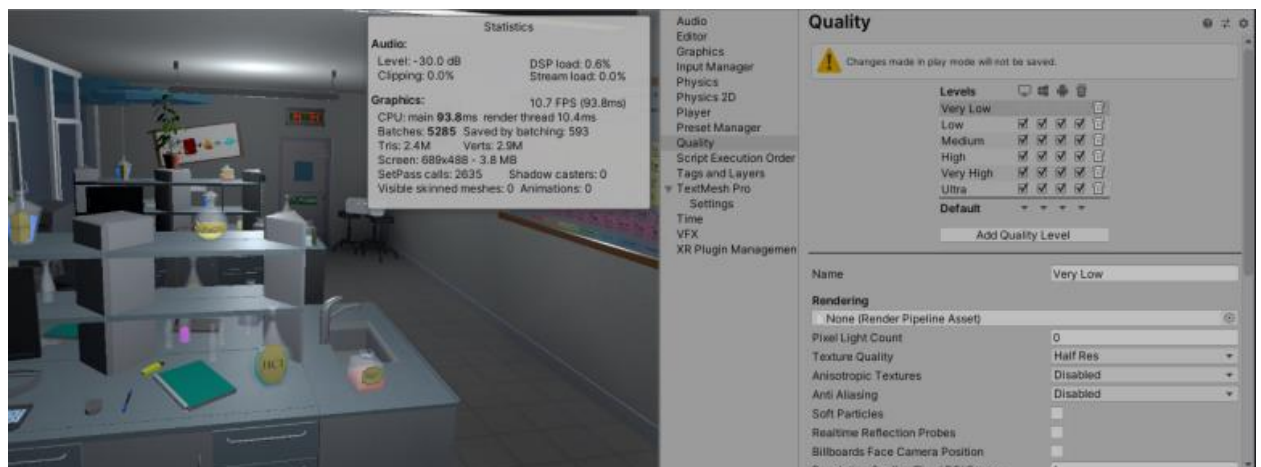


Figure 47 Low quality

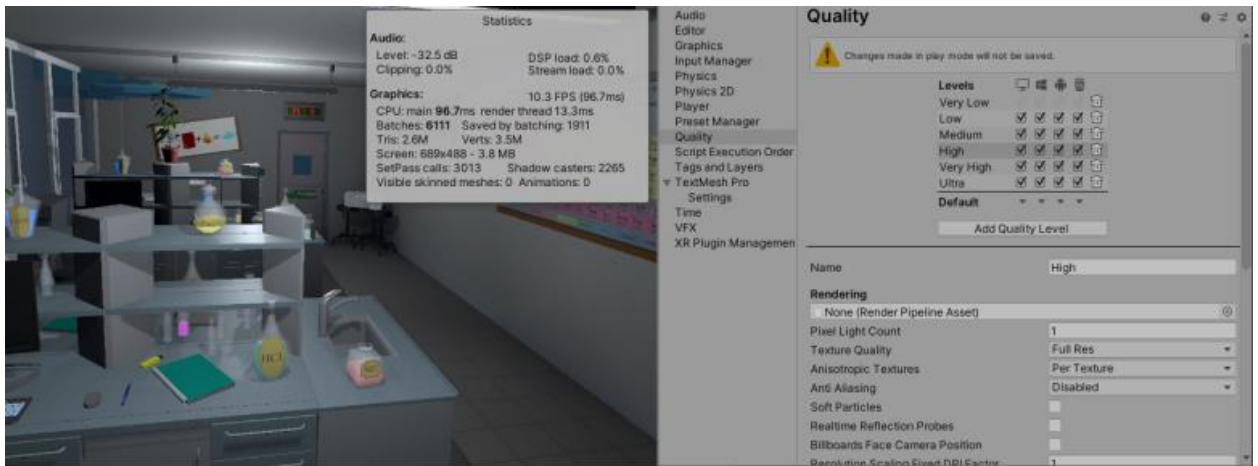


Figure 48 High Quality



Figure 49 Ultra Quality

Task #18. Bake on cube each side different light.

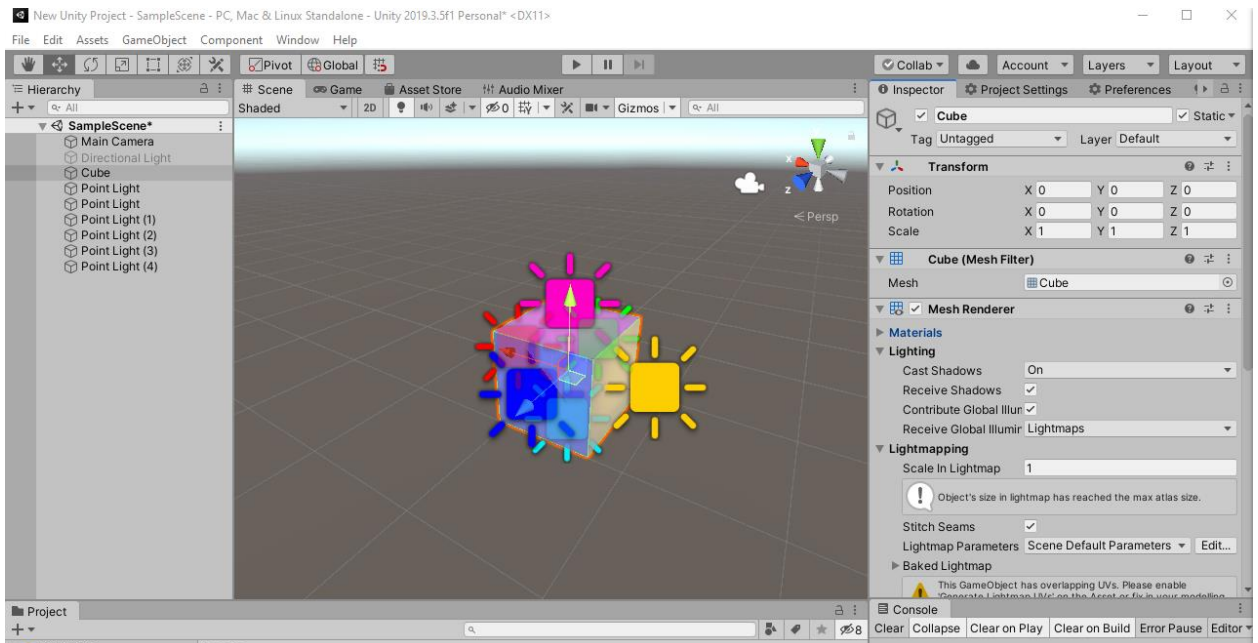


Figure 50 Cube for baking

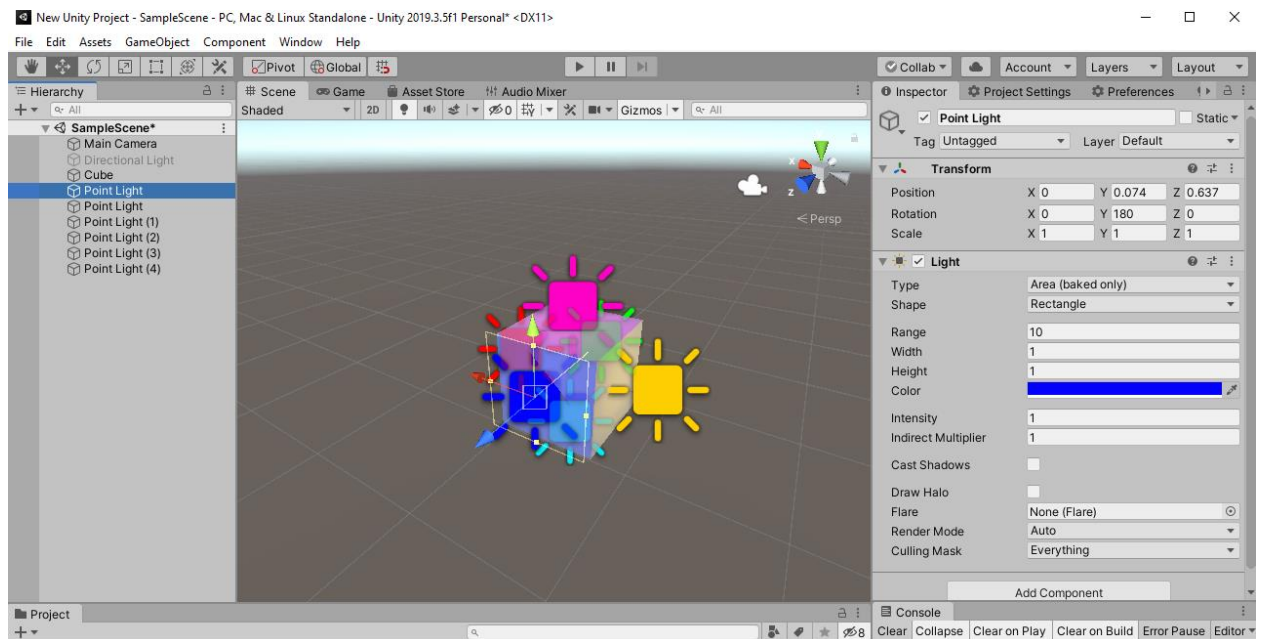


Figure 51 baking Light

Other lights are similar, color of lights just changes.

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;
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("_TintColor");
    }

    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();
}
}

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;
    }
}

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)
        //{
        //}
    }
}

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] GroundControl 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);
    }
}

```