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

Tables of Images *3*

Table of Tables/functions *4*

Work Distribution Table: *5*

Description of Your Game *6*

Laboratory work #1 .... *7*

List of tasks 7

Solution *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

Literature list *25*

ANNEX *26*

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

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

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 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.

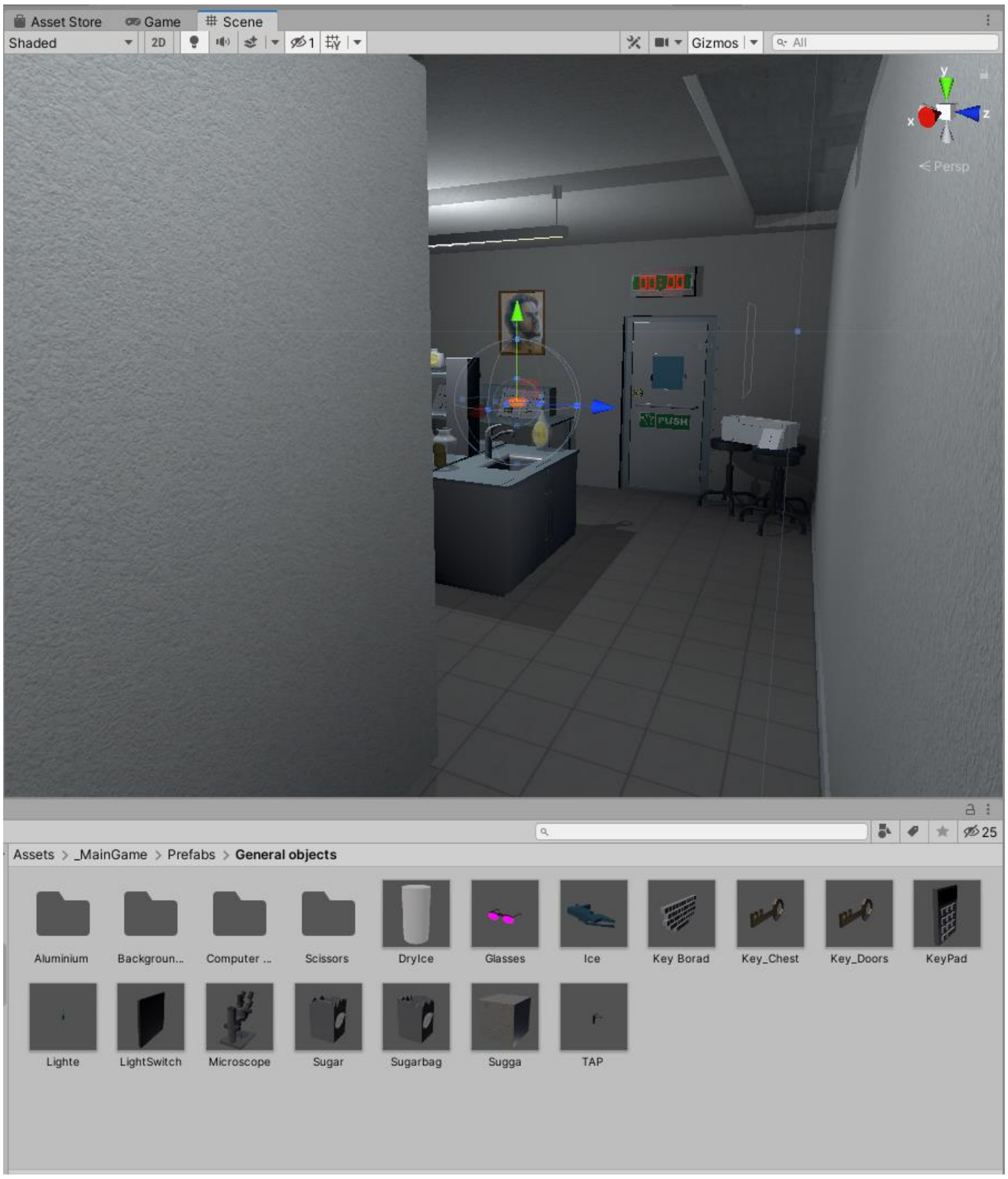


Figure 1. Second level scene with some used prefabs below

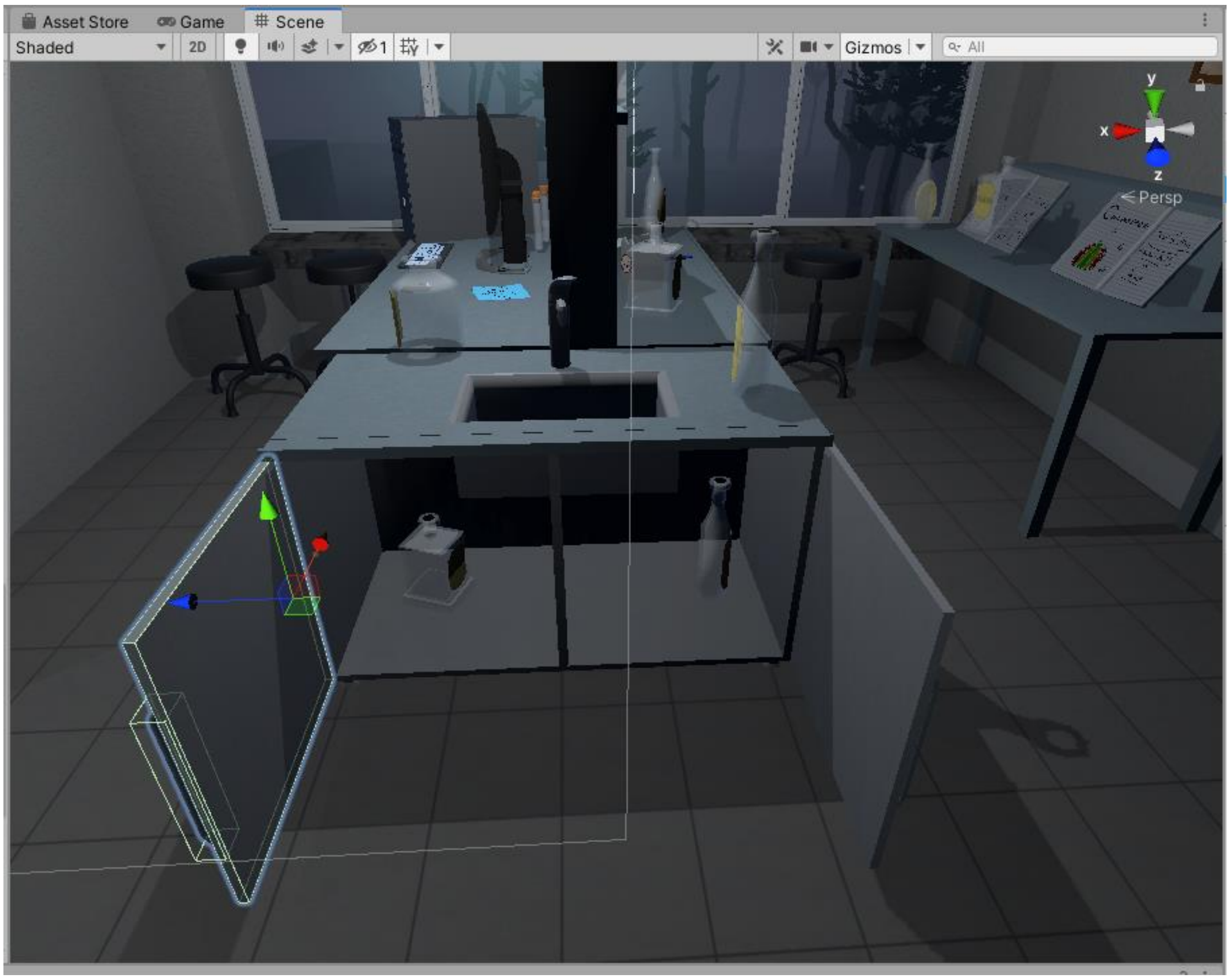


Figure 2. Objects inside of a shelf

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.

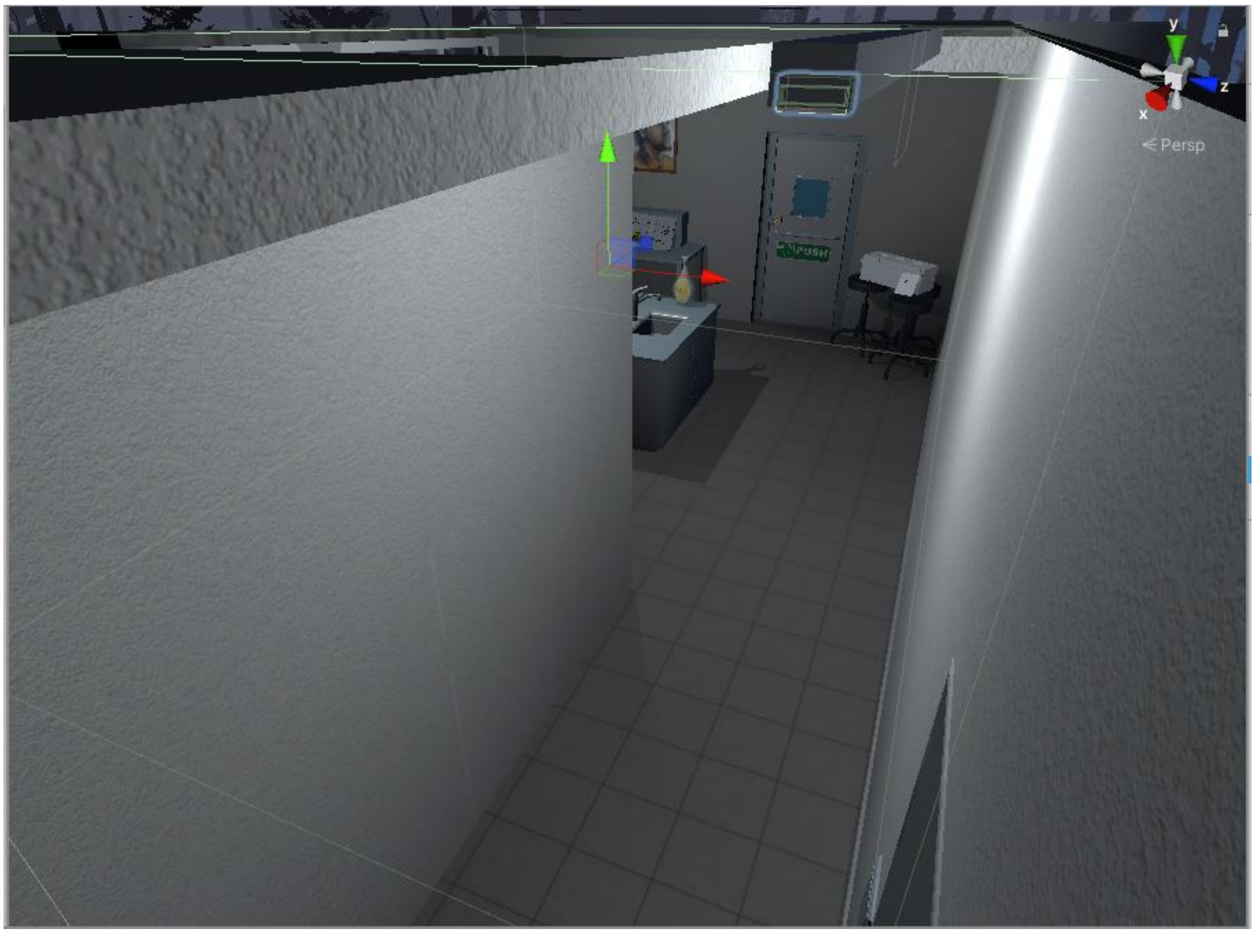


Figure 3. Timer trigger

```
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.

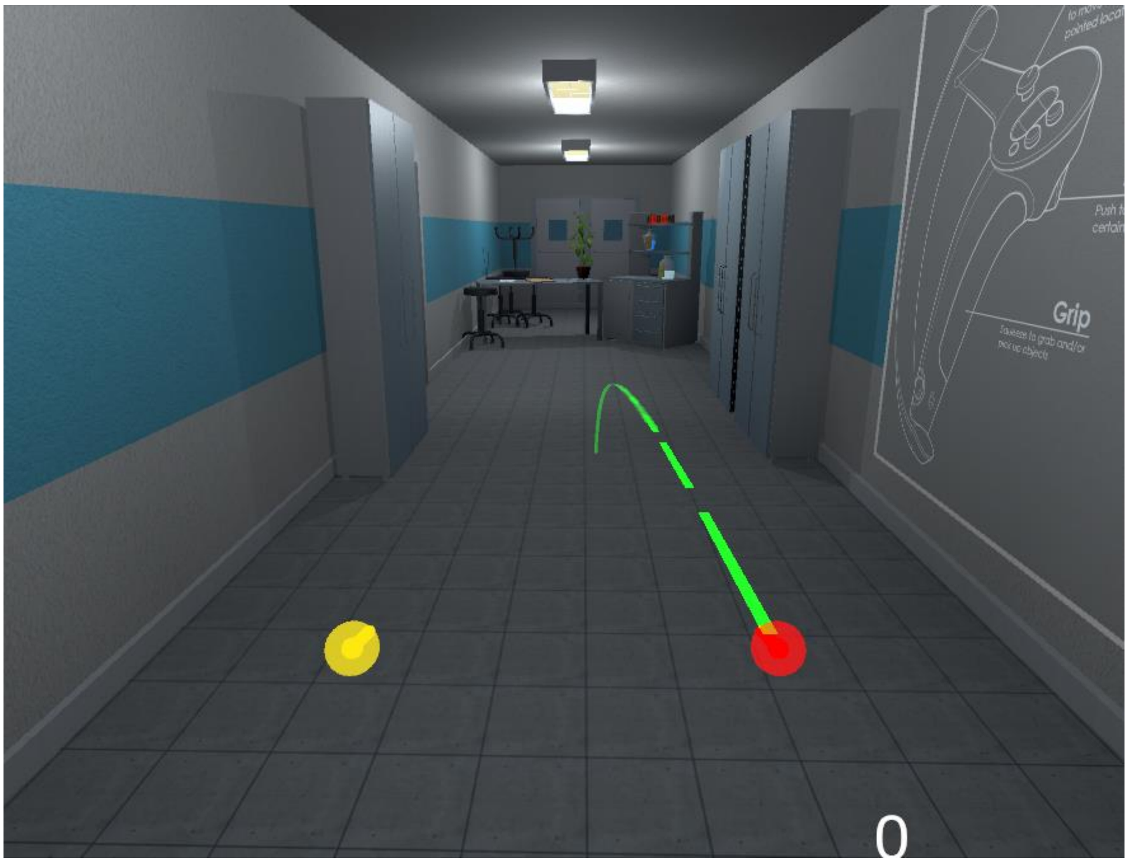


Figure 4. Correct teleportation

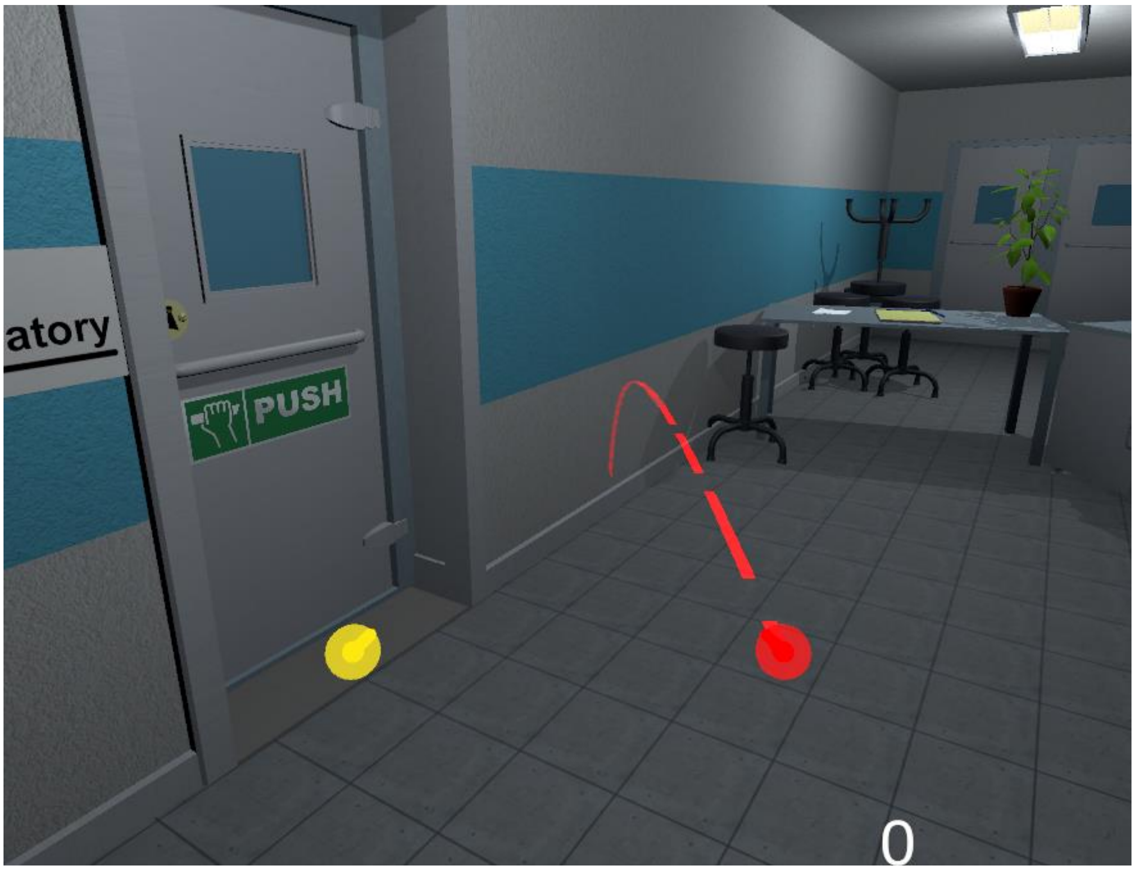
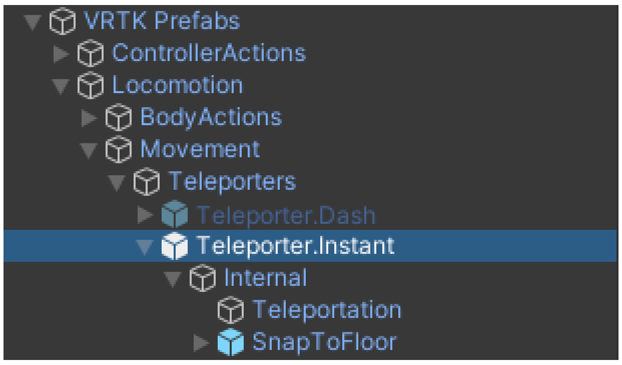
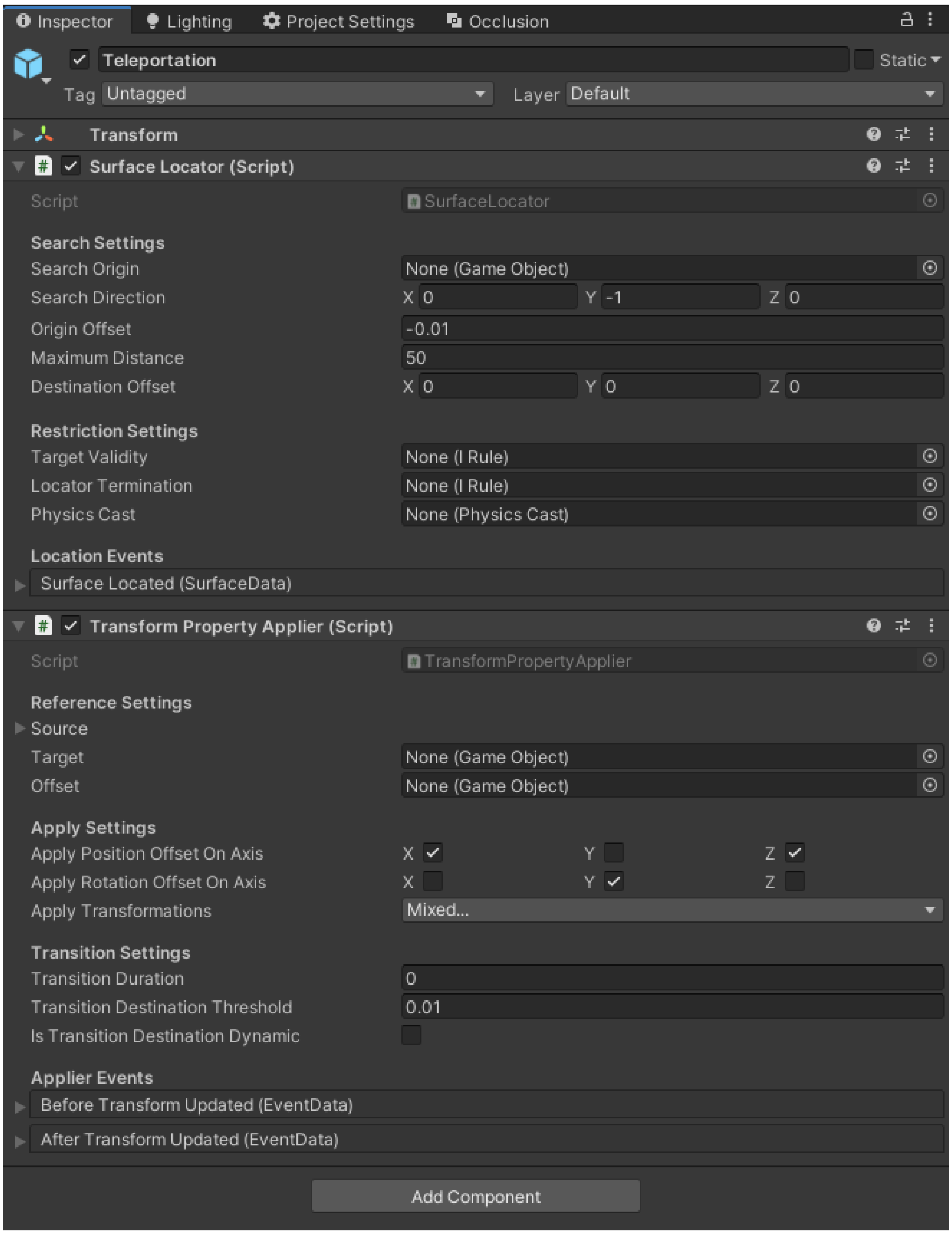


Figure 5. Incorrect teleportation



```
▼ VRTK Prefabs
  ▶ ControllerActions
  ▼ Locomotion
    ▶ BodyActions
    ▼ Movement
      ▼ Teleporters
        ▶ Teleporter.Dash
        ▼ Teleporter.Instant
          ▼ Internal
              Teleportation
            ▶ SnapToFloor
```

Figure 6. VRTK Teleportation hierarchy



Teleportation	Static
Tag: Untagged	Layer: Default

Transform

Surface Locator (Script)

Script	SurfaceLocator
Search Settings	
Search Origin	None (Game Object)
Search Direction	X 0 Y -1 Z 0
Origin Offset	-0.01
Maximum Distance	50
Destination Offset	X 0 Y 0 Z 0
Restriction Settings	
Target Validity	None (I Rule)
Locator Termination	None (I Rule)
Physics Cast	None (Physics Cast)
Location Events	
Surface Located (SurfaceData)	

Transform Property Applier (Script)

Script	TransformPropertyApplier
Reference Settings	
Source	
Target	None (Game Object)
Offset	None (Game Object)
Apply Settings	
Apply Position Offset On Axis	X ✓ Y ☐ Z ✓
Apply Rotation Offset On Axis	X ☐ Y ✓ Z ☐
Apply Transformations	Mixed...
Transition Settings	
Transition Duration	0
Transition Destination Threshold	0.01
Is Transition Destination Dynamic	☐
Applier Events	
Before Transform Updated (EventData)	
After Transform Updated (EventData)	

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.

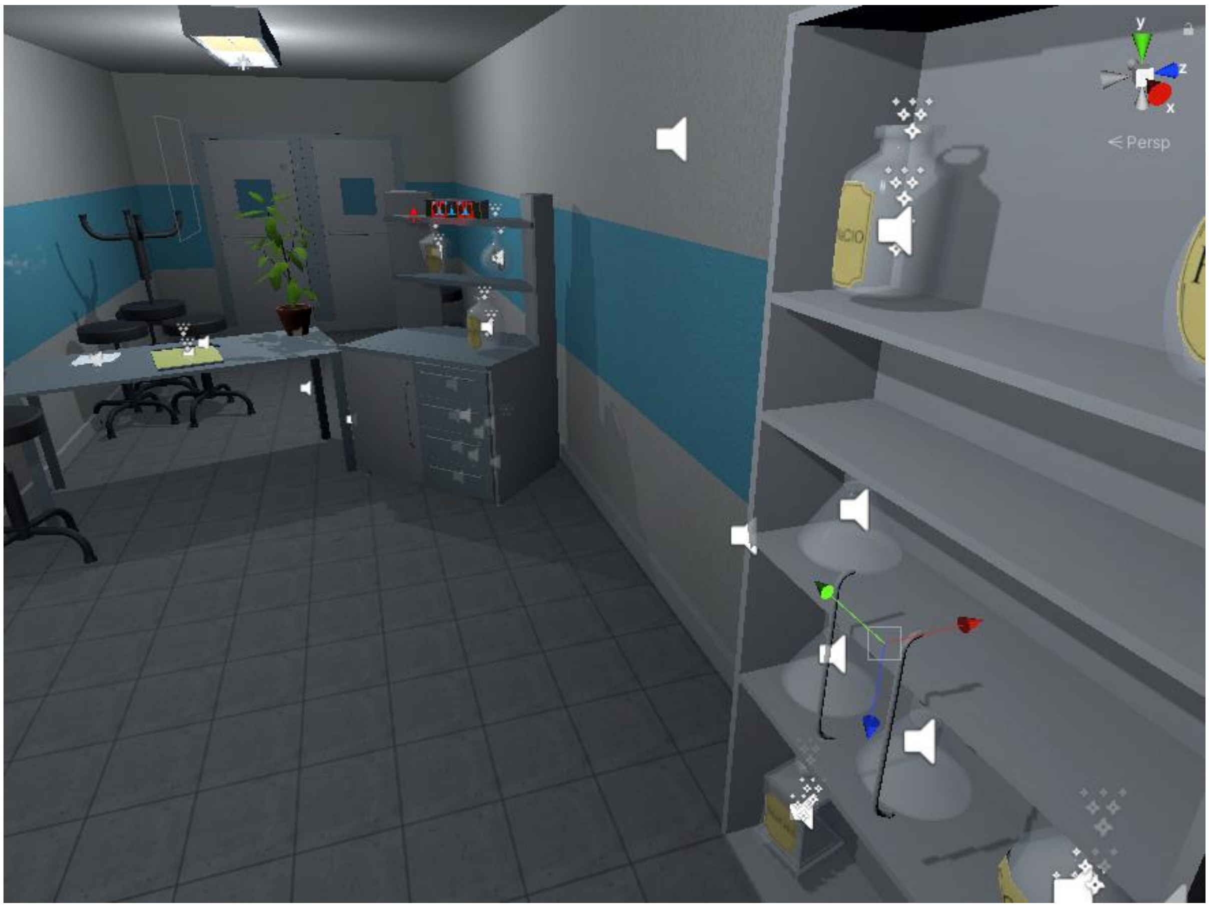


Figure 8. Scene decoration with various GameObjects

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

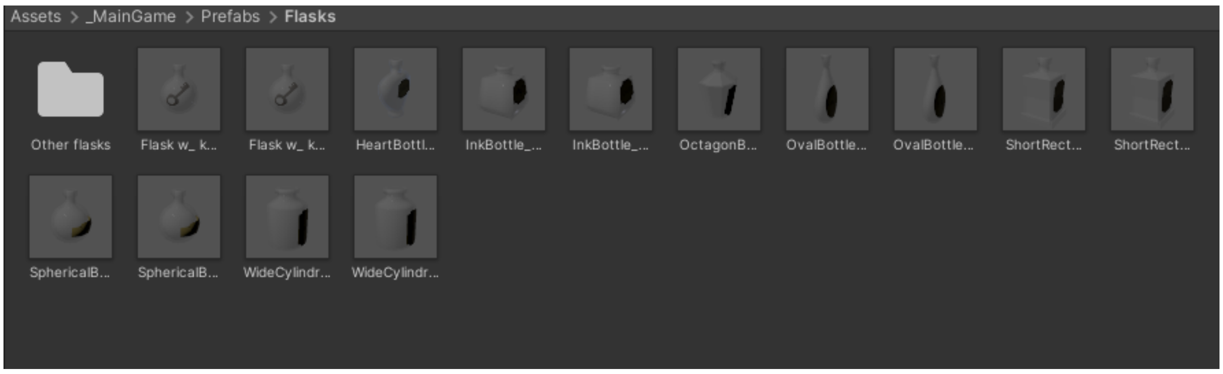


Figure 9. Objects used in the scene #1

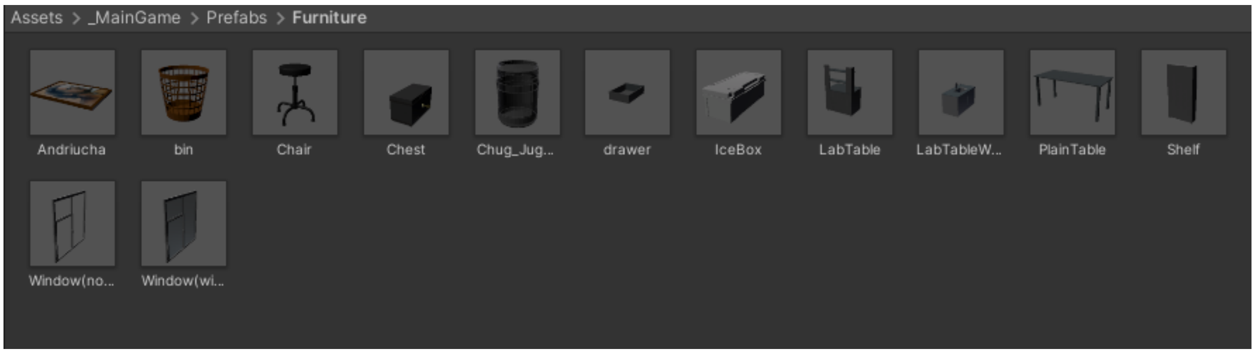


Figure 10. Objects used in the scene #2

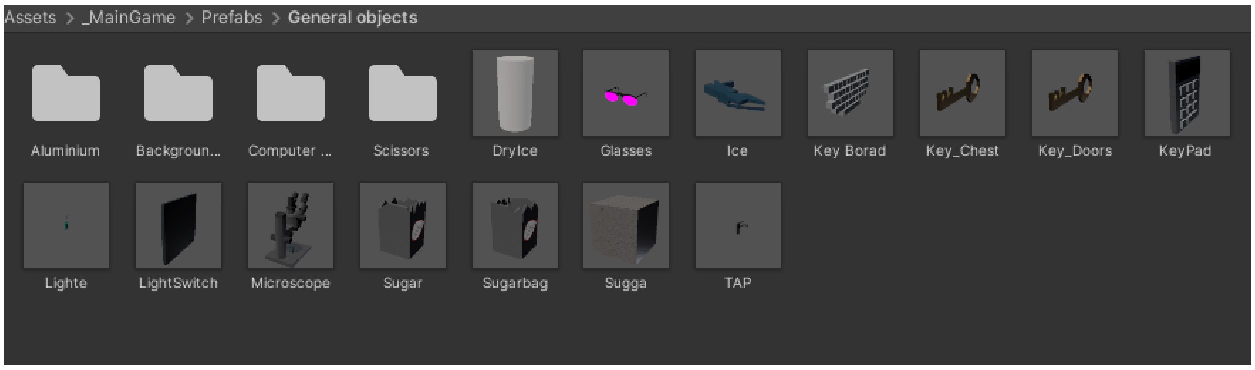


Figure 11. Objects used in the scene #3

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

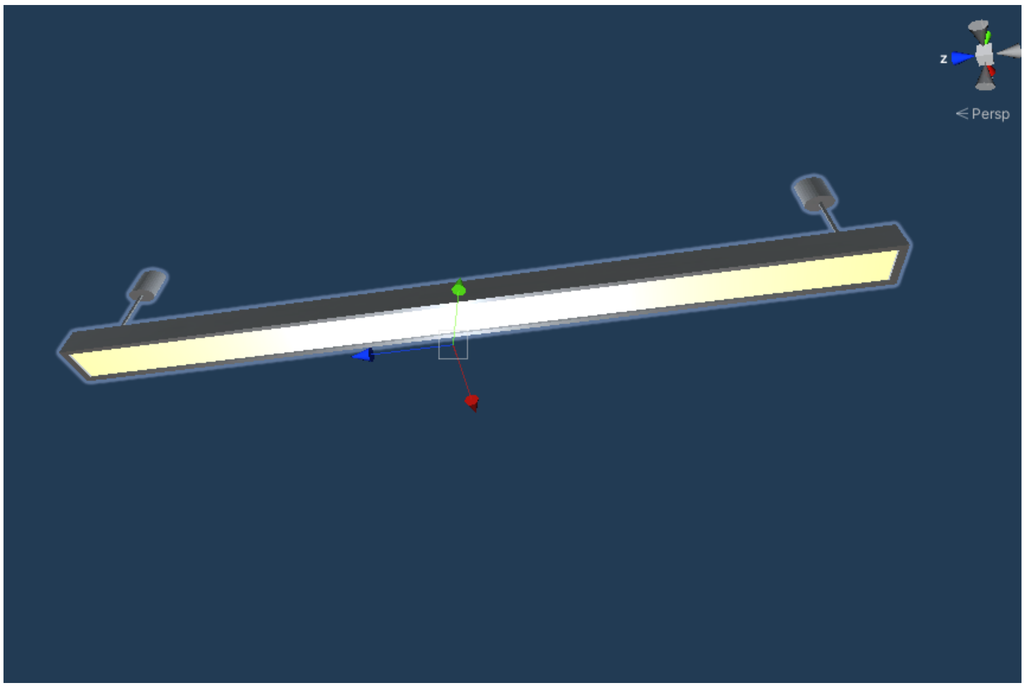


Figure 12. Lamp model #1

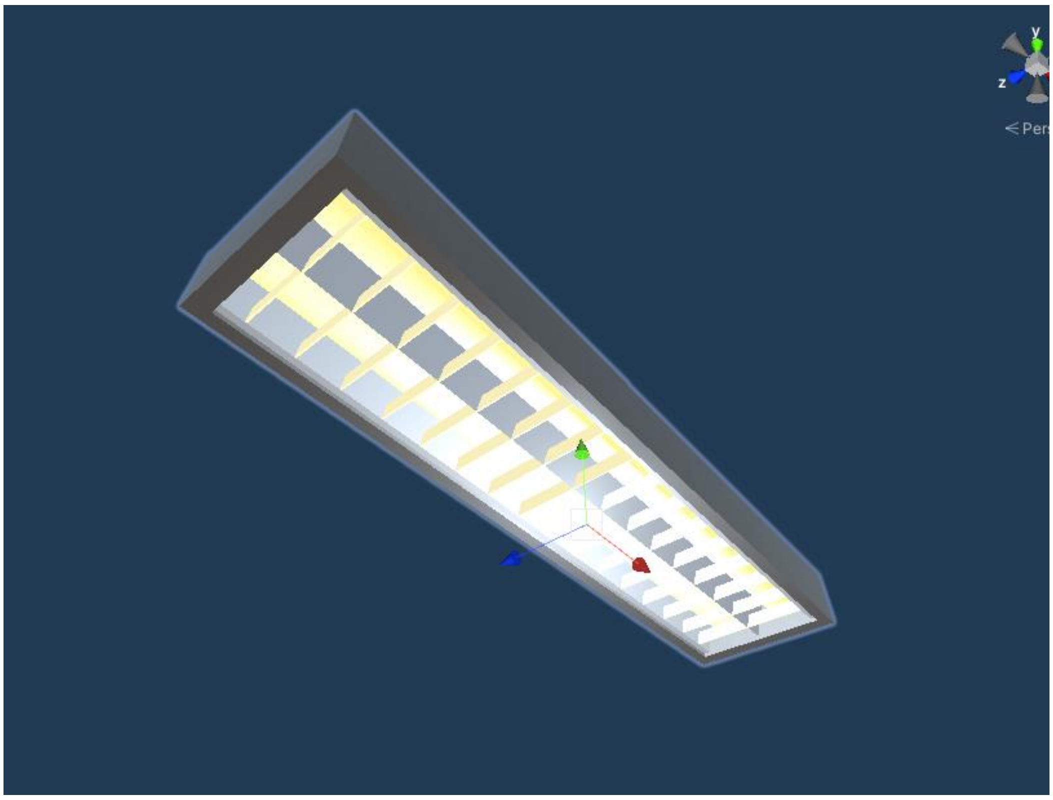


Figure 13. Lamp model #2

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 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 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.

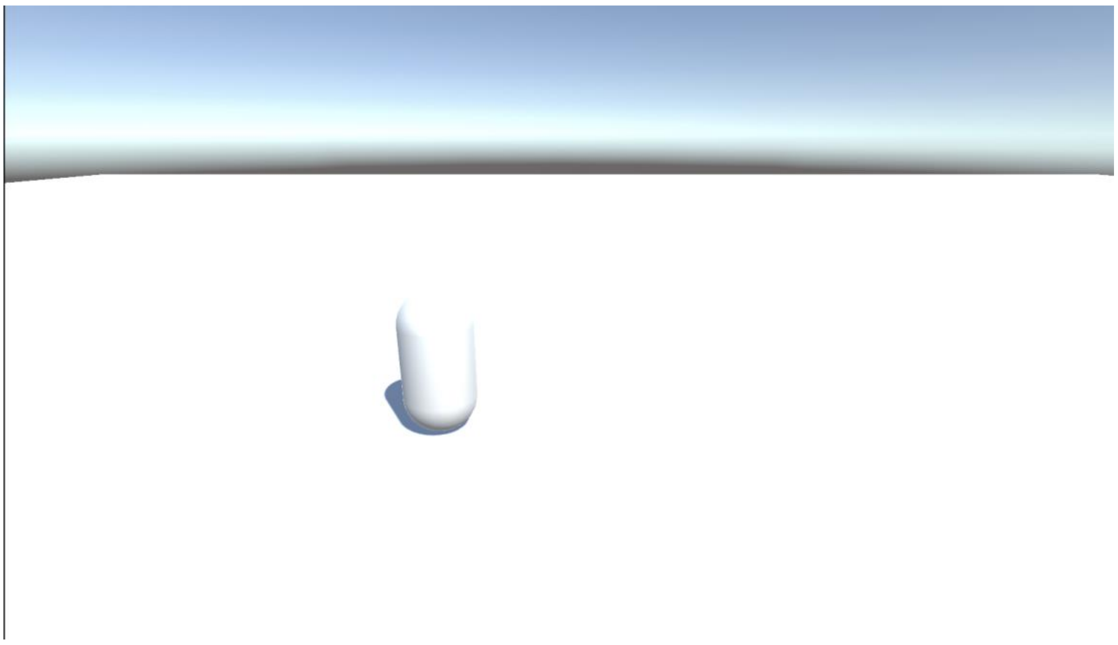


Figure 14. Capsule standign still

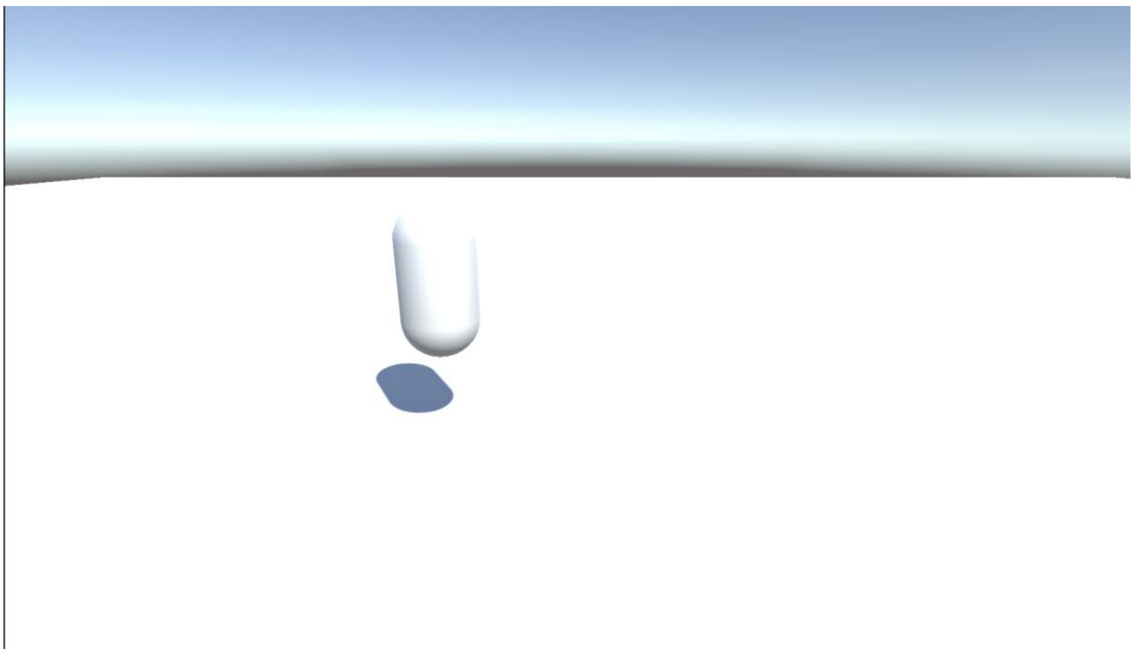


Figure 15. Hovering capsule

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

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;
    }

}
```