

**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF INFORMATICS**

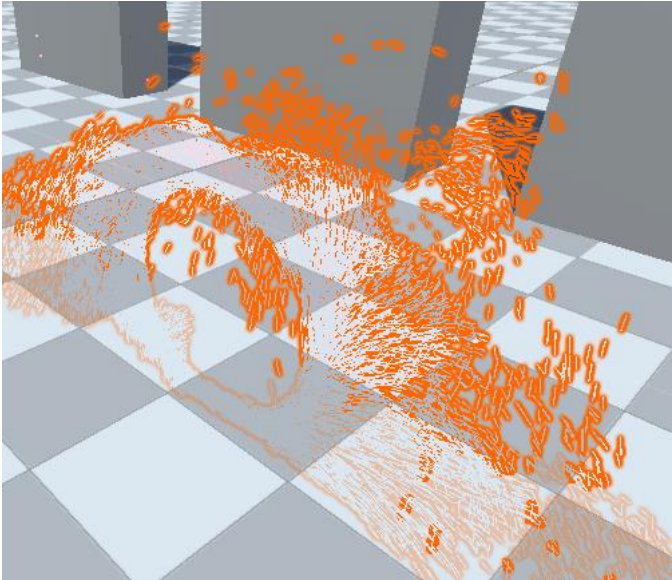
# **T120B166 Development of Computer Games and Interactive Applications**

*Magika*

*Group,  
Vilhelmas Stankevičius,  
Martynas Kuliešius*

*Date: 2024.02.13*

## Tables of Contents

|                                                                                                                                                                                                             |    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| <i>Tables of Images</i> .....                                                                                                                                                                               | 4  |
| <i>Table of Tables/functions</i> .....                                                                                                                                                                      | 5  |
| <i>Work Distribution Table:</i> .....                                                                                                                                                                       | 6  |
| <i>Description of Your Game</i> .....                                                                                                                                                                       | 7  |
| <i>Laboratory work #1</i> .....                                                                                                                                                                             | 8  |
| List of tasks.....                                                                                                                                                                                          | 8  |
| <i>Solution</i> .....                                                                                                                                                                                       | 8  |
| Task #1. <i>Create player controller for player character model</i> .....                                                                                                                                   | 8  |
| Task #3. <i>Create first spell(s)</i> .....                                                                                                                                                                 | 17 |
|  .....                                                                                                                    | 19 |
| <i>Gynimo užduotis:</i> .....                                                                                                                                                                               | 19 |
| <i>Padaryti, kad būtų galima channelint spellą, AOE damage daryt jeigu neperilgai castinamas spellas, durant damage pagal mana consumed*time, jei per ilgai castinama, playeris praranda savo HP.</i> ..... | 19 |

```

// reference
private bool HandleChargingSpell()
{
    Debug.Log("Got here");
    chargeUpState.text = currChargeTimeLab.ToString("F1");
    if (Input.GetKey(KeyCode.Mouse1))
    {
        currentMana -= Time.deltaTime * 3f;
        Debug.Log("Lul");
        currChargeTimeLab += Time.deltaTime;
        if (currChargeTimeLab > maxChargeTimeLab)
        {
            lastCharge.text = $"Overtime: {maxChargeTimeLab.ToString("F1")}";
            gameObject.GetComponent<HealthComponent>().TakeDamage(50f);
            currChargeTimeLab = 0f;
        }
        else
        {
        }
    }
    else
    {
        if (currChargeTimeLab > 0)
        {
            Collider[] hitColliders = Physics.OverlapSphere(transform.position, currChargeTimeLab * 15f);
            foreach (var col in hitColliders)
            {
                if (col.gameObject.layer == 6)
                {
                    Debug.Log("Here");
                    col.GetComponent<HealthComponent>().TakeDamage(currChargeTimeLab * 20f);
                }
            }
            lastCharge.text = $"Successful: {currChargeTimeLab.ToString("F1")}";
            currChargeTimeLab = 0f;
        }
    }
    return false;
}

```

|                                                           |    |
|-----------------------------------------------------------|----|
| .....                                                     | 20 |
| <b>Laboratory work #2</b> .....                           | 21 |
| List of tasks.....                                        | 21 |
| <b>Solution</b> .....                                     | 21 |
| Task #1. <i>Add alternative spells.</i> .....             | 21 |
| Task #2. <i>Create terrain with trees and grass</i> ..... | 22 |
| Task #3. <i>Create inventory system</i> .....             | 24 |
| <b>Laboratory work #3</b> .....                           | 25 |
| List of tasks.....                                        | 25 |
| <b>Solution</b> .....                                     | 25 |
| Task #1. <i>Title of Task</i> .....                       | 25 |
| Task #2. <i>Title of Task</i> .....                       | 28 |
| Task #3. <i>Title of Task</i> .....                       | 30 |
| <b>User's manual</b> .....                                | 31 |
| <b>Literature list</b> .....                              | 33 |
| <b>ANNEX</b> .....                                        | 34 |

## Tables of Images

|                                                                        |    |
|------------------------------------------------------------------------|----|
| Figure 1 Character model .....                                         | 8  |
| Figure 2 Character controller code .....                               | 11 |
| Figure 3 Script for making a new spell .....                           | 12 |
| Figure 4 Code for HealthComponent .....                                | 13 |
| Figure 5 Player magic system code .....                                | 14 |
| Figure 6 Code for spell logic .....                                    | 16 |
| Figure 7 First spell of the game called Dark Spirit bomb .....         | 17 |
| Figure 8 VFX Graph for Dark Spirit Bomb .....                          | 17 |
| Figure 9 VFX Graph for a work in progress spell called Fire Blade..... | 18 |
| Figure 10 Fire Blade spell VFX in scene editor .....                   | 19 |
| Figure 11 code screenshot for problem solution .....                   | 20 |
| Figure 12. Screenshot #1 .....                                         | 21 |
| Figure 13 Updated view of the terrain.....                             | 23 |
| Figure 14 Evil mountain view .....                                     | 23 |
| Figure 15. Screenshot #3 .....                                         | 24 |
| Figure 16. Screenshot #1 .....                                         | 25 |
| Figure 17. Screenshot #2 .....                                         | 29 |
| Figure 18. Screenshot #3 .....                                         | 30 |
| Figure 19. Screenshot #5 .....                                         | 31 |
| Figure 20. Screenshot #5 .....                                         | 31 |

**Table of Tables/functions**

Table 4. Title of fragment #1 .....22

Table 6. Title of fragment #3 .....24

Table 7. Title of fragment #1 .....27

Table 8. Title of fragment #2 .....30

Table 9. Title of fragment #3 .....30

Work Distribution Table:

| <i>Name/Surname</i>           | <i>Description of game development part</i>              |
|-------------------------------|----------------------------------------------------------|
| <i>Vilhelmas Stankevičius</i> | <i>Programavimas, žemėlapių generavimas</i>              |
| <i>Martynas Kuliešius</i>     | <i>Programavimas, reikalingų modelių kūrimas/gavimas</i> |

## Description of Your Game

### Game idea:

Our game is a 3D sandbox action-adventure type game inspired by The Legend of Zelda Breath of the Wild. Instead of swords and bows the character will use magic to fight the enemies. There would be a simple inventory system to control and swap magic spells as well as any other supplemental items. Enemies will most likely use Navigation Mesh AI to control movements and player targeting while a certain condition is triggered scripts adjust will adjust the behavior of the enemies. Enemies have object that they can drop for the player to pick up and use later on. In laymen terms we want a single relatively intelligent enemy and a single boss. Terrain mesh is made by hand with Unity tools while the majority of objects are asset-based. If there are specific requirements for a mesh, it would be made by hand. The game will have a game state save system that the player can load into a save. Majority of spells will be made with VFX by hand or at least by following a learning material online. The idea is only focused on a small demo version.

# Laboratory work #1

## List of tasks (main functionality of your project)

1. Create player controller for player character model
2. Create base controller for magic system
3. Create first spell(s)

## Solution

### Task #1. *Create player controller for player character model*

Each game has a player controllable character, to make sure our game also has one, we took a character model from Unity Asset store and we used a character rig to give our playable character movement animations. Also to control each animation, we use character states to switch between different animations. Our character can move around the world, look left to right, jump around.



Figure 1 Character model

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

public class PlayerController : MonoBehaviour
{
    public enum PlayerState { Idle, Running, Jumping, Channeling, Dead };
    private PlayerState currentState = PlayerState.Idle;
    private float inputHorizontal;
```



```

private float inputVertical;
private Rigidbody rb;
public Vector3 testVector;
private Vector3 inputDirection = Vector3.zero;
[SerializeField] private GameObject animatorCharacterModel;
[SerializeField] private Animator animator;
[SerializeField] private float moveSpeed;
[SerializeField] private float jumpForce;
[SerializeField] private float mouseRotSpeed;
[SerializeField] private GameObject groundCheckObject;
[SerializeField] private bool isTouchingGround;
[SerializeField] private int currJumpCount;
[SerializeField] private int maxJumpCount;

void Start()
{
    animator = animatorCharacterModel.GetComponent<Animator>();
    rb = gameObject.GetComponent<Rigidbody>();
}

// Update is called once per frame
void Update()
{
    testVector = groundCheckObject.transform.position;

    StateManager();
    MovementInput();
    JumpHandler();
}

void StateManager()
{
    switch (currentState)
    {
        case PlayerState.Idle:
            GetMovementTrigger();
            CheckGround();
            MovementInput();
            break;

        case PlayerState.Running:
            GetMovementTrigger();
            CheckGround();
            MovementInput();
            break;

        case PlayerState.Jumping:
            GetJumpingTrigger();
            CheckGround();
            break;
    }
}

```

```

        case PlayerState.Channeling:
            break;

        case PlayerState.Dead:
            break;

    }
}

void CheckGround()
{
    Collider[] hitColliders = Physics.OverlapSphere(testVector, 0.1f, ~7);

    if (hitColliders.Length > 0)
    {
        isTouchingGround = true;
        if (currentState == PlayerState.Jumping)
            ChangeState(PlayerState.Idle);
    }
    else
    {
        isTouchingGround = false;
        ChangeState(PlayerState.Jumping);
    }
}

void GetMovementTrigger()
{
    if (inputVertical > 0.01f && inputHorizontal == 0f)
        AnimTrigger("RunningForwards");
    else if (inputVertical < -0.01f && inputHorizontal == 0f)
        AnimTrigger("RunningBackwards");
    else if (inputVertical > 0.01f && inputHorizontal > 0.01f)
        AnimTrigger("RunningForwardsRight");
    else if (inputVertical > 0.01f && inputHorizontal < -0.01f)
        AnimTrigger("RunningForwardsLeft");
    else if (inputHorizontal < -0.01f && inputVertical < 0.01f && inputVertical > -0.01f)
        AnimTrigger("RunningLeft");
    else if (inputHorizontal > 0.01f && inputVertical < 0.01f && inputVertical > -0.01f)
        AnimTrigger("RunningRight");
    else if (inputVertical < -0.01f && inputHorizontal > 0.01f)
        AnimTrigger("RunningBackwardsRight");
    else if (inputVertical < -0.01f && inputHorizontal < -0.01f)
        AnimTrigger("RunningBackwardsLeft");
    else
        AnimTrigger("Idle");
}

void GetJumpingTrigger()
{
    AnimTrigger(currentState.ToString());
}

```

```

    }

    void ChangeState(PlayerState newState)
    {
        currentState = newState;
    }

    void AnimTrigger(string trigger)
    {
        if (!animator.GetCurrentAnimatorStateInfo(0).IsName(trigger))
            animator.SetTrigger(trigger);
    }

    void MovementInput()
    {
        if (currentState == PlayerState.Idle || currentState == PlayerState.Running)
        {
            inputHorizontal = Input.GetAxis("Horizontal");
            inputVertical = Input.GetAxis("Vertical");
            inputDirection = transform.TransformDirection(new Vector3(inputHorizontal, 0f,
inputVertical));

            if (inputHorizontal != 0f && inputVertical != 0f)
                ChangeState(PlayerState.Running);
            else
                ChangeState(PlayerState.Idle);
        }
    }

    void JumpHandler()
    {
        if (!Input.GetKeyDown(KeyCode.Space))
            return;
        rb.velocity = new Vector3(rb.velocity.x, jumpForce, rb.velocity.z);
        ChangeState(PlayerState.Jumping);
    }

    void FixedUpdate()
    {
        rb.velocity = new Vector3(inputDirection.x * moveSpeed, rb.velocity.y, inputDirection.z *
moveSpeed);
        if (Input.GetAxis("Mouse X") > 0) transform.Rotate(Vector3.up * mouseRotSpeed);
        if (Input.GetAxis("Mouse X") < 0) transform.Rotate(Vector3.up * -mouseRotSpeed);
    }
}

```

**Figure 2 Character controller code**

## Task #2. Create base controller for magic system

For our sandbox action-adventure game that features magic as the primary and main type of combat mechanic, our game needs a magic system. For magic to work in any game project, developers must create some sort of magic system, which is exactly what we did. To make a somewhat working base of our magic system, we needed to make a spell script, a spellScriptableObject script, a HealthComponent script and a PlayerMagicSystem script. To create new spells, we made a new asset Menu that creates a new spell object with base stats, code for creating such a menu seen in Figure 2. Each spell has different damage amounts, different mana/health costs, lifetime of a spell which is for how long that spell will be in game after casting if it doesn't hit anything/anyone. Speed is self explanatory – how fast the spell moves in the game and spell radius shows what is the active radius of the spell. Since this is the base of our magic system for the game, we are going to add more functionality to each of the scrips to accommodate our needs and want for our game.

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

[CreateAssetMenu(fileName = "New Spell", menuName = "Spells")]

1 reference
public class SpellScriptableObject : ScriptableObject
{
    1 reference
    public float DamageAmount = 10f;
    2 references
    public float ManaCost = 5f;
    0 references
    public float HealthCost = 0f;
    1 reference
    public float LifeTime = 2f;
    2 references
    public float Speed = 15f;
    1 reference
    public float SpellRadius = 0.5f;

    //Status effects
    //Thumbnail
    //TimeBetweenCasts
    // magicElements
}
```

Figure 3 Script for making a new spell

For the game to have a working magic system, of course, it needs to have some sort of a health system, therefore we have a simple implementation of a super simple health system that once the health reaches 0, the enemy game object is destroyed. Code seen in Figure 3.

```

2 references
public class HealthComponent : MonoBehaviour
{
    1 reference
    [SerializeField] private float maxHealth = 50f;
    4 references
    private float currentHealth;

    0 references
    private void Awake()
    {
        currentHealth = maxHealth;
    }

    1 reference
    public void TakeDamage(float damageToApply)
    {
        currentHealth -= damageToApply;
        Debug.Log(currentHealth);

        if(currentHealth <= 0)
        {
            Destroy(this.gameObject);
        }
    }
}

```

Figure 4 Code for HealthComponent

```

using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Unity.VisualScripting;
using UnityEngine;

public class PlayerMagicSystem : MonoBehaviour
{
    [SerializeField] private Spell spellToCast;
    [SerializeField] private float maxMana = 100f;
    [SerializeField] private float currentMana;
    [SerializeField] private float manaRechargeRate = 2f;
    [SerializeField] private float timeWaitForRecharge = 1f;
    private float currentManaRechargeTimer;
    [SerializeField] private float timeBetweenCast = 0.25f;
    private float currentCastTimer;

    [SerializeField] private Transform castPoint;

    private bool castingMagic = false;
    private PlayerController playerController;
    // Start is called before the first frame update
    void Awake()
    {
        playerController = new PlayerController();
    }
}

```

```

// Update is called once per frame
private void OnEnable()
{
    // playerController.Enable();
}

private void Update()
{
    bool isSpellCastHeldDown = Input.GetMouseButtonDown(0);
    bool hasEnoughMana = currentMana - spellToCast.SpellToCast.ManaCost >= 0f;

    if(!castingMagic && isSpellCastHeldDown && hasEnoughMana)
    {
        castingMagic = true;
        currentMana -= spellToCast.SpellToCast.ManaCost;
        currentCastTimer=0;
        currentManaRechargeTimer=0;
        castSpell();
        print("casting");
    }

    if(castingMagic)
    {
        currentCastTimer+=Time.deltaTime;
        if(currentCastTimer > timeBetweenCast)
        {
            castingMagic = false;
        }
    }

    if(currentMana < maxMana && !castingMagic && !isSpellCastHeldDown)
    {
        currentManaRechargeTimer += Time.deltaTime;
        if(currentManaRechargeTimer> timeWaitForRecharge)
        {
            currentMana+= manaRechargeRate*Time.deltaTime;
            if(currentMana>maxMana)
            {
                currentMana=maxMana;
            }
        }
    }

}

void castSpell()
{
    Instantiate(spellToCast, castPoint.position, castPoint.rotation);
}
}

```

**Figure 5** Player magic system code

For the magic system to work in our game, we needed to create a script to be able to use any spells and control their behaviors and control their life time. In the code seen in Figure 4, our player has a selectable Spell object, maximum amount and current amount of mana variables, that can be upgradable if we introduce a scaling system. Each one of the SerializeField variables could be upgraded at a later stage of the game, for now they are what they're set to. Using void Update method, we control spell casts, magic regeneration, control cast timers.

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

[RequireComponent(typeof(SphereCollider))]
[RequireComponent(typeof(Rigidbody))]
public class Spell : MonoBehaviour
{
    public SpellScriptableObject SpellToCast;
    private SphereCollider myCollider;
    private Rigidbody myRigidBody;

    private void Awake()
    {
        myCollider = GetComponent<SphereCollider>();
        myCollider.isTrigger = true;
        myCollider.radius = SpellToCast.SpellRadius;

        myRigidBody = GetComponent<Rigidbody>();
        myRigidBody.isKinematic = true;
        //transform.rotation = Quaternion.LookRotation(transform.forward);

        Destroy(this.gameObject, SpellToCast.LifeTime);
    }

    private void Update()
    {
        if (SpellToCast.Speed > 0)
        {
            transform.Translate(Vector3.forward * SpellToCast.Speed * Time.deltaTime);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        //apply hit particles
        //apply spell effects
        //apply sounds
        if (other.gameObject.CompareTag("Enemy"))
        {
            HealthComponent enemyHealth = other.GetComponent<HealthComponent>();
            enemyHealth.TakeDamage(SpellToCast.DamageAmount);
        }
    }
}
```

```
        Destroy(this.gameObject);  
    }  
}
```

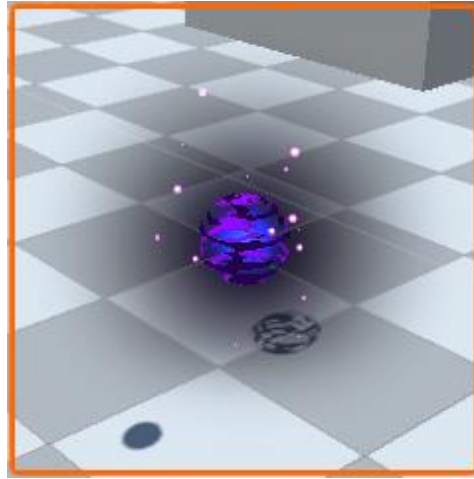
**Figure 6** Code for spell logic

We cannot forget about the spells themselves. As seen in Figure 5, on Awake of the spell, we instantiate spells rigidbody and colliders to later be used for checking if spells hit enemies, destroy them after a set lifetime. With void Update method we check spells speed, to be sure that the spell can be cast. Using method OnTriggerEnter, we use collider's trigger option to control what happens to the enemies after a spell hits enemies. As we can see, the game object with the tag "Enemy" takes a set amount of damage that is set for a certain spell and after the spell hits a collider of some sorts, the spell gets destroyed before the end of its lifetime.

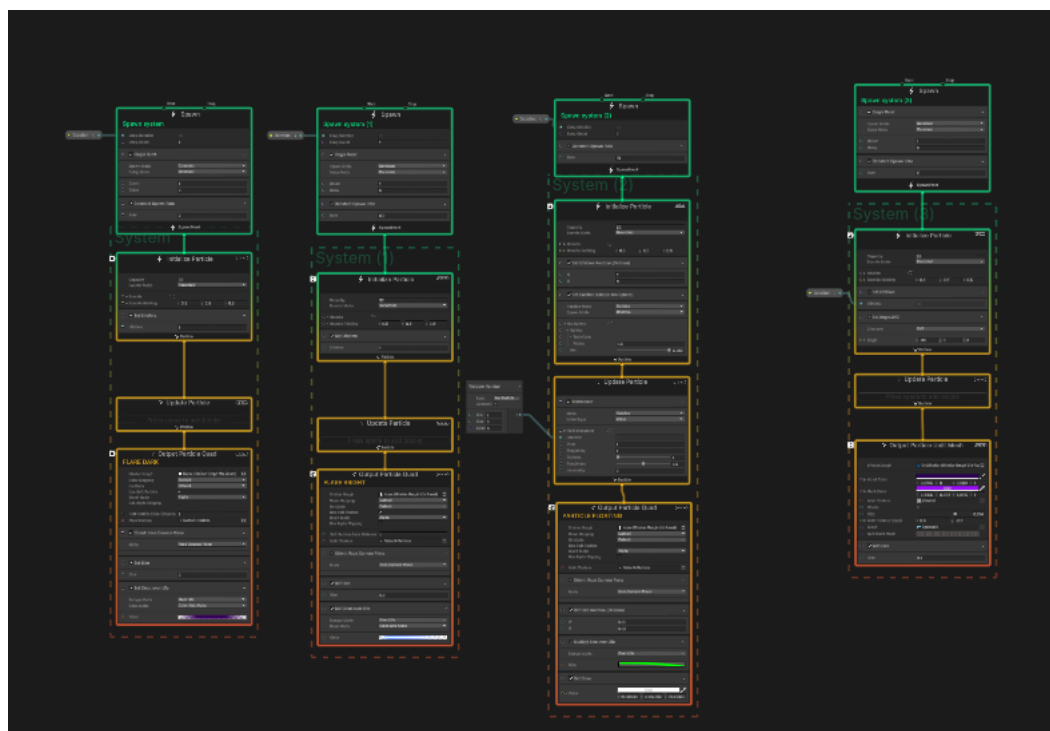


### Task #3. Create first spell(s)

The creation of a spell is somewhat difficult. We need to make a spell scriptable object for each spell, make a Unity VFX Graph visual effect for a spell, add colliders, add trails or certain features. Using Unity VFX Graph is a new experience and it is quite difficult to get the hang of things, but it is a really interesting way to make visual effects for a game.

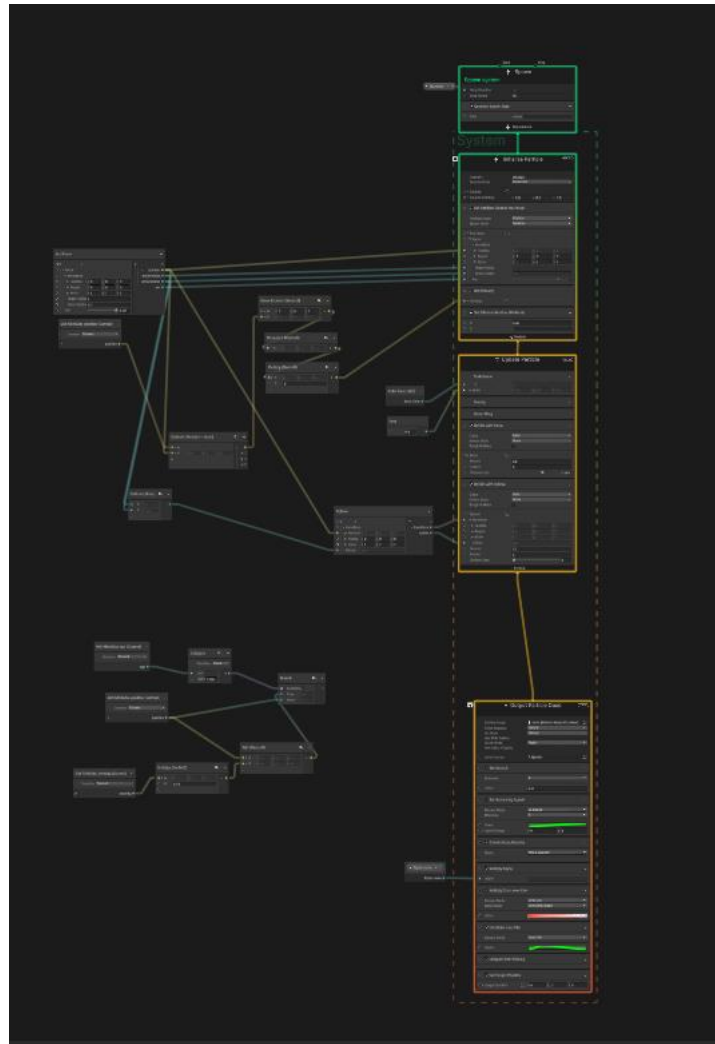


**Figure 7 First spell of the game called Dark Spirit bomb**



### Figure 8 VFX Graph for Dark Spirit Bomb

As seen in Figure 6, that's how our games first spell looks like. To make it animated and create a interesting look we used Unity VFX Graph. Figure 7 shows the graph for our first spell. It consists of 4 blocks of graphs to create a powerful looking spell.



**Figure 9 VFX Graph for a work in progress spell called Fire Blade**

As seen in Figure 8, this is a work-in-progress spell VFX Graph for a spell called Fire Blade. Currently, the spell is not used and probably will be reused as a teleportation spell rather than a attack spell just because of its shape, size and look. All it needs is to finish a rendering block in the graph, but due to extensive calculations needed to make such a visual effect thats reactive to the environment it takes more time to make. Figure 9 shows how this spell currently looks.

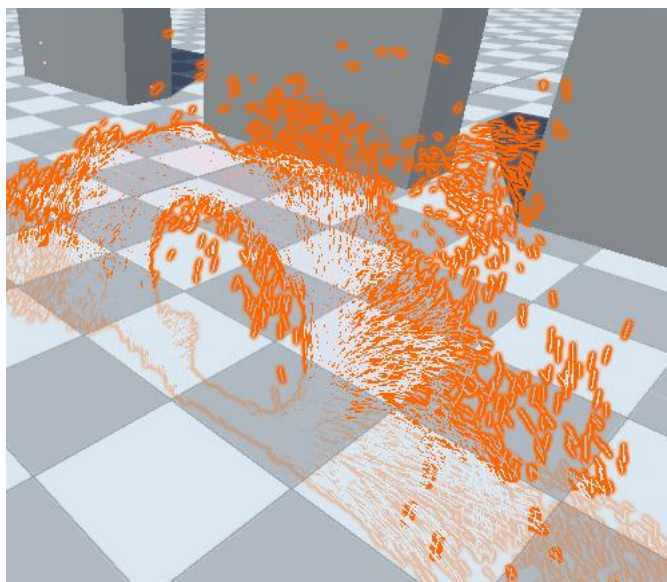


Figure 10 Fire Blade spell VFX in scene editor

Gynimo užduotis:

Padaryti, kad būtų galima channelint spellą, AOE damage daryt jeigu neperilgai castinamas spellas, darant damage pagal mana consumed\*time, jei per ilgai castinama, playeris praranda savo HP.

```

| reference
private bool HandleChargingSpell()
{
    Debug.Log("Got here");
    chargeUpState.text = currChargeTimeLab.ToString("F1");
    if (Input.GetKey(KeyCode.Mouse1))
    {
        currentMana -= Time.deltaTime * 3f;
        Debug.Log("Lul");
        currChargeTimeLab += Time.deltaTime;
        if (currChargeTimeLab > maxChargeTimeLab)
        {
            lastCharge.text = $"Overtime: {maxChargeTimeLab.ToString("F1")}";
            gameObject.GetComponent<HealthComponent>().TakeDamage(50f);
            currChargeTimeLab = 0f;
        }
        else
        {
        }
    }
    else
    {
        if (currChargeTimeLab > 0)
        {
            Collider[] hitColliders = Physics.OverlapSphere(transform.position, currChargeTimeLab * 15f);
            foreach (var col in hitColliders)
            {
                if (col.gameObject.layer == 6)
                {
                    Debug.Log("Here");
                    col.GetComponent<HealthComponent>().TakeDamage(currChargeTimeLab * 20f);
                }
            }
            lastCharge.text = $"Successful: {currChargeTimeLab.ToString("F1")}";
            currChargeTimeLab = 0f;
        }
    }
    return false;
}

```

Figure 11 code screenshot for problem solution

## Laboratory work #2

### List of tasks (main functionality of your project)

1. Add animation for charge spell
2. Create terrain with trees and grass and bushes
3. Update game UI
4. Create/update user interface for spells and such

### Solution

#### Task #1. *Add animation for charge spell*

In our first lab, we created the first spell called “Dark spirit bomb”. It’s animation was made using Unity’s own VFXGraph system. For our charge spell, it is an AOE spell that effects everything around the player, so the spell basically looks like a circle that forms around the player.



Figure 12. Screenshot #1

```

1 reference
void UpdateEffectSize()
{
    float normalizedCharge = Mathf.Clamp01(currChargeTime/maxChargeTime);
    //float size = Mathf.Lerp(0.1f, 35f, normalizedCharge);
    float size = 30f*currChargeTime;
    vfxGraph.SetFloat("EffectSize", size);
    vfxGraph.SetFloat("LifeTime", currChargeTime);
}

1 reference
void HandleSpellRotation()
{
    if(activeEffectInstance !=null)
    {
        activeEffectInstance.transform.rotation = Quaternion.identity;
    }
}

// ----- Misc -----

2 references
void HandleEffectInstantiating()
{
    if(activeEffectInstance == null)
    {
        activeEffectInstance = Instantiate(vfxGraph, castPoint.position, quaternion.identity, castPoint);
        activeEffectInstance.transform.localRotation = Quaternion.identity;
    }
    activeEffectInstance.SetFloat("EffectSize", 5f);
    UpdateEffectSize();
}

2 references
void HandleEffectRemoval()
{
    if(activeEffectInstance!=null)
    {
        Destroy(activeEffectInstance.gameObject);
        activeEffectInstance = null;
    }
}

```

Table 1. Title of fragment #1

## Task #2. *Create terrain with trees and grass*

Every game needs a good looking terrain, the same goes for our game “Magika”. To fulfill this task, we have used Unity’s own terrain editing, transforming and tree placing functionality. First of all, we needed to find some good looking assets to utilize in making our terrain look like an actual terrain, and we chose “LowPoly Environments” by “PolyTope Studios” for the trees, bushes and other little environment things. To make our skybox look better, we chose “Fantasy Skybox” by “Render Knight” to make our worlds sky look better. And we cant forget to talk about “Handpainted Grass and Ground textures” by “Chromisu” for the textures for our ground textures.





Figure 13 Updated view of the terrain



Figure 14 Evil mountain view

Task #3. Update game UI and fix up code

Every game relies on some sort UI to inform a player of certain information for example hp and mana counts. Here we have our UI that displays spells with cooldowns, hp and mana.



Figure 15. Screenshot #3

In the case of using functions, the description of each main function should be completed with the source code FRAGMENTS (the functions should be indexed in a separate table of contents);

```
// ----- UI -----  
  
1 reference  
private void HandleRMBUI()  
{  
    if (currentCastedAbility != "RMB")  
        RMBCooldownCurr -= Time.deltaTime;  
  
    RMBWheelChargeUp.GetComponent<Image>().fillAmount = currChargeTime / maxChargeTime;  
  
    if (RMBCooldownCurr > 0)  
    {  
        RMBCooldownNumber.SetActive(true);  
        RMBCooldownNumber.GetComponent<TextMeshProUGUI>().text = RMBCooldownCurr.ToString("F1");  
    }  
    else  
        RMBCooldownNumber.SetActive(false);  
  
    RMBPerlinNoise.GetComponent<Image>().fillAmount = RMBCooldownCurr / RMBCooldownMax;  
}
```

Table 2. Title of fragment #3



## Laboratory work #3

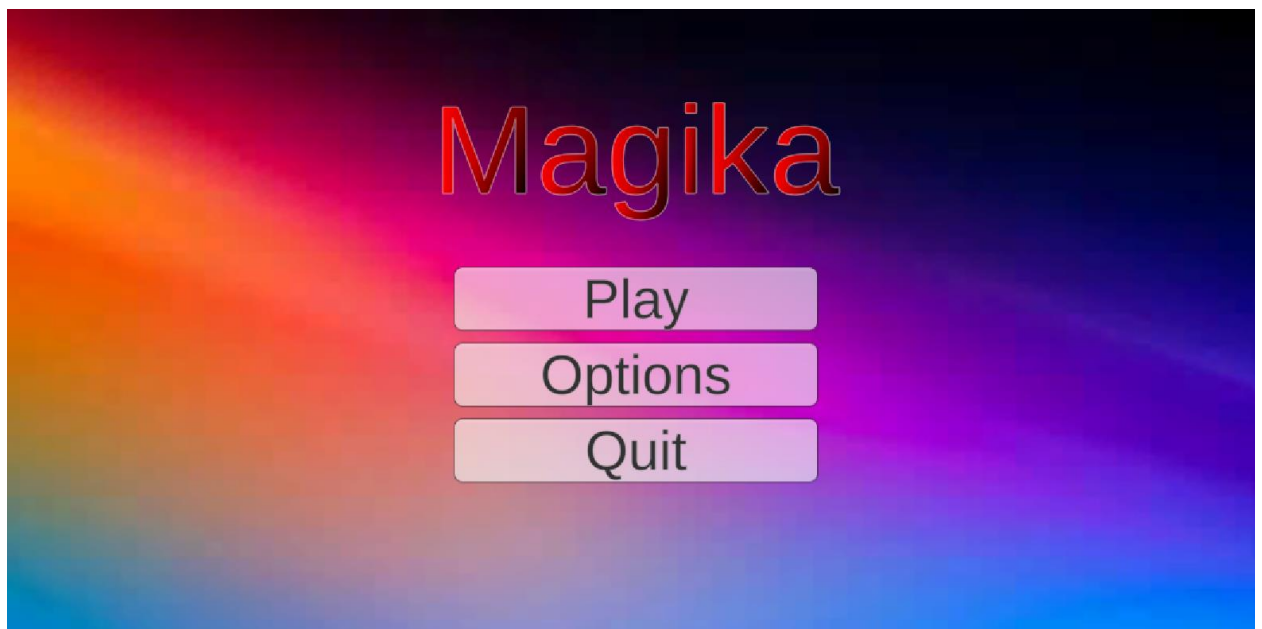
### List of tasks (main functionality of your project)

1. Add Menu UI, Pause Menu, GameOver screen
2. Add HP bar on enemies
3. Upgrade enemies AI

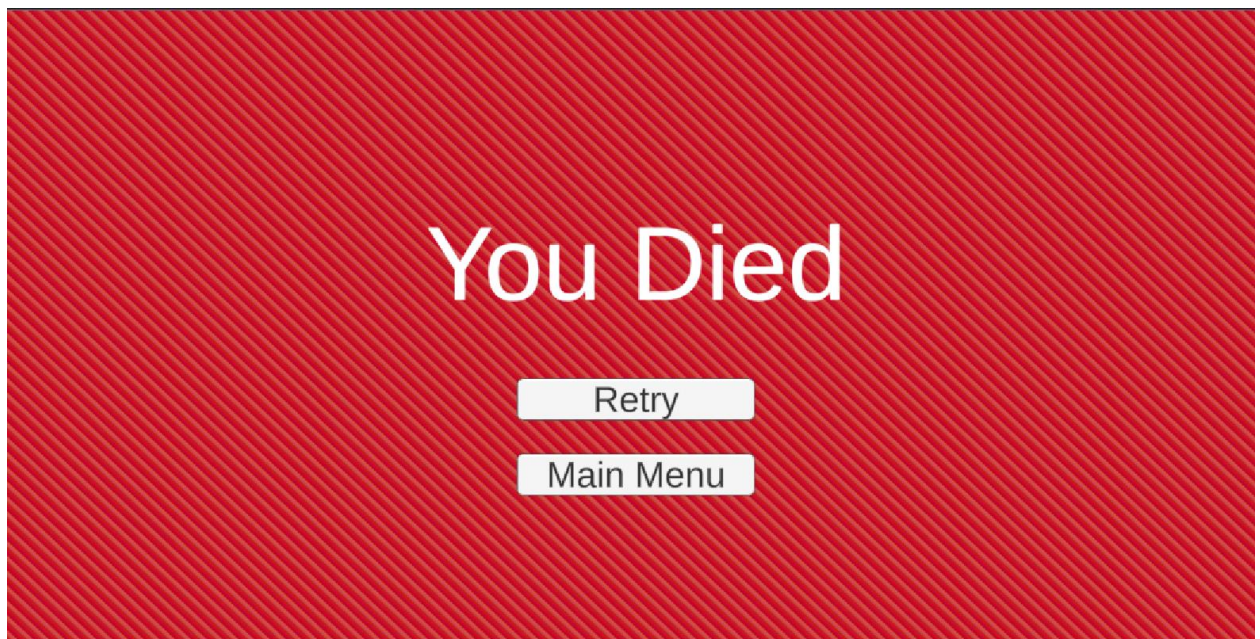
### Solution

#### Task #1. *Add Main menu, Pause menu, Game Over screen*

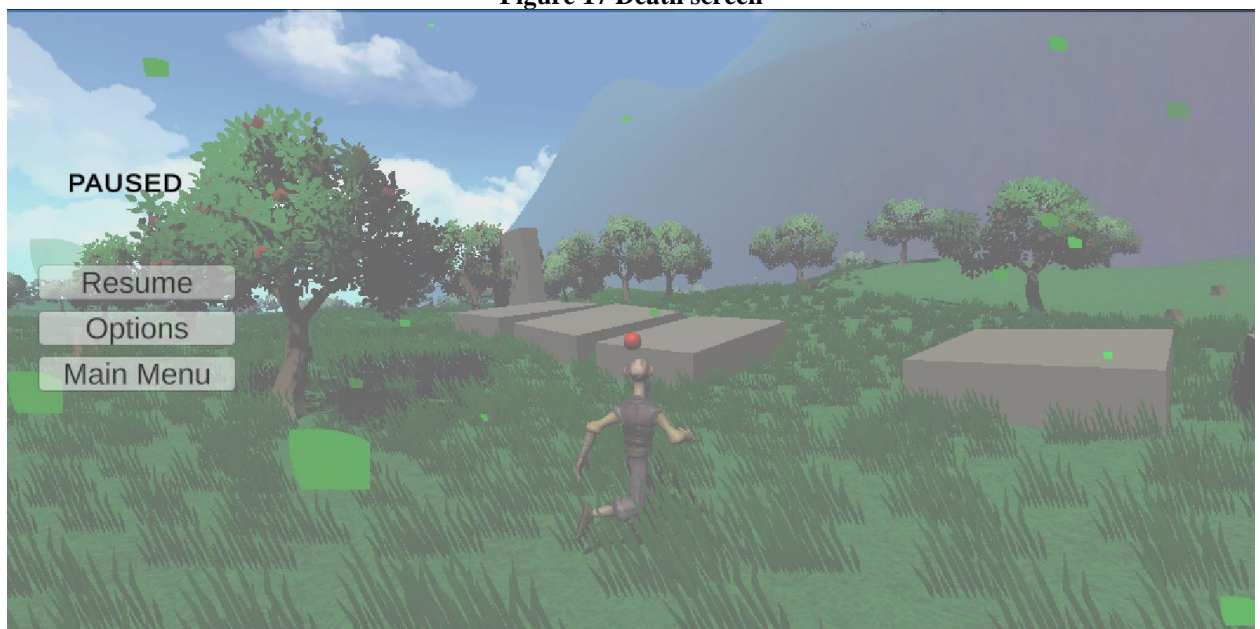
Every game needs some sort of ui so players can do something else instead of only playing the game. Therefore we needed to make some menus for players to be able to start a new game, go to settings and get to restart game if they manage to die.



**Figure 16.** Main menu screenshot



**Figure 17 Death screen**



**Figure 18 Pause menu**

Code fragments that help/make the UI and pause screens work:

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

public class GameOverScript : MonoBehaviour
{
    public void Retry()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
    }
    public void MainMenu()
```

```

    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 2);
    }
}

```

Table 3. Title of fragment #1

```

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

public class PauseMenu : MonoBehaviour
{
    public static bool Paused = false;
    public GameObject PauseMenuCanvas;
    public GameObject GameUI;

    // Start is called before the first frame update
    void Start()
    {
        Time.timeScale = 1f;
    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape))
        {
            if(Paused)
            {
                Play();
            }
            else
            {
                Stop();
            }
        }
    }

    void Stop()
    {
        PauseMenuCanvas.SetActive(true);
        GameUI.SetActive(false);
        Time.timeScale = 0f;
        Paused = true;
    }

    public void Play()
    {

```

```

        PauseMenuCanvas.SetActive(false);
        GameUI.SetActive(true);
        Time.timeScale = 1f;
        Paused = false;
    }

    public void MainMenuButton()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
    }
}

```

```

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

public class NewBehaviourScript : MonoBehaviour
{
    /// <summary>
    /// Start play to load into the game
    /// </summary>
    public void Play()
    {
        //Scenos veikia pagal build index 0-main menu, 1 - main game scene,
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    /// <summary>
    /// Quit the game
    /// </summary>
    public void Quit()
    {
        Application.Quit();
        Debug.Log("Player has Quit the game");
    }
}

```

## Task #2. *Make enemy HP bar*

When fighting enemies it is a must to see their hp and to see how much damage the player is doing to them, therefore we need to make a health bar that would show the player how hard he's been hitting the enemy.





**Figure 19.** Health bar on top of enemy

To control the health bar I updated the healthComponent script to perform slider adjustments:

```
public Slider healthSlider;
public Slider easeHealthSlider;
private float lerpSpeed = 0.005f;

private void Awake()
{
    currentHealth = maxHealth;
    healthSlider.value = healthSlider.maxValue = maxHealth;
    easeHealthSlider.maxValue = healthSlider.maxValue;
}

void Update()
{
    UpdateUI();
    UpdateHealthBar();
}

private void UpdateUI()
{
    if (LayerMask.NameToLayer("Player") == gameObject.layer)
    {
        healthBar.GetComponent<Image>().fillAmount = currentHealth /
maxHealth;
        healthBarNumber.GetComponent<TextMeshProUGUI>().text =
currentHealth.ToString("F0");
    }
}

private void UpdateHealthBar()
{
    if(healthSlider.value != currentHealth)
```

```
{
    healthSlider.value = currentHealth;
}
if(healthSlider.value != easeHealthSlider.value)
{
    easeHealthSlider.value = Mathf.Lerp(easeHealthSlider.value,
currentHealth, lerpSpeed);
}
}
```

Table 4. Title of fragment #2

### Task #3. Upgrade enemy AI

In games it's no fun when they enemies are standing still without doing anything, therefore we need to make the enemy have some roaming capabilities, switch between animations and attack or at least follow the player. Enemy now runs after the player if it sees it in sight, attack the player dealing damage and if player runs far away from the enemy it stops chasing the player.



Figure 20. Enemy walking away from the player

In the case of using functions, the description of each main function should be completed with the source code FRAGMENTS (the functions should be indexed in a separate table of contents);

|                         |
|-------------------------|
| Fragment of Source Code |
|-------------------------|

Table 5. Title of fragment #3



## User's manual (for the Individual work defence)

### How to play?

When the player launches the game, they're greeted by the Main Menu in which they can choose to play, go to options and quit. In order to play the game, the Player needs to press Play as seen below.



**Figure 21.** Main Screen of the game

When the player loads into the game, they're greeted by the games world and first enemy: For now the game consists of running around and defeating every enemy they encounter



**Figure 22.** Screenshot of player in the game.

**Descriptions of the rules of the game.** There are no rules in particular to the game, all a player needs to do is go walk around and have fun. If the player overuses the charge spell or die in combat with enemy mobs, they will have to restart their game from the beginning. For now there

is no end screen if all the enemies are defeated, but if we decide to do more for our game, we will add it at a later date.

**Descriptions of the controls / keys.** To play the game the player must use mouse and keyboard to control the player. We use the standard control scheme:

- WASD to walk around,
- Space to jump,
- LMB to use spell,
- RMB to charge spell,

Paskutinio gynimo metu gavome atlikti tokią užduotį – kai enemy gyvybės nukrenta žemiau 50% jie turi bėgti iki gydymo obelisko ir kai pasigydys toliau pulti žaidėją.



Figure 23 Gydymo obeliskas prie kurio bėga.



```

private void StateHeal()
{
    CheckHealth();
    float distance = Vector3.Distance(healingObject.transform.position,
gameObject.transform.position);
    if (distance <= 5f)
        agent.destination = healingObject.transform.position;
    else
    {
        animator.SetTrigger("Walking");
        // if(GetComponent<>)
    }
    if (GetComponent<HealthComponent>().GetHealth() >=
GetComponent<HealthComponent>().GetMaxHealth())
    {
        ChangeToStatePatrol();
    }
}

```

Scriptas skirtas valdyti enemy behaviour. Su juo valdomas gyvybės tikrinimas ir pradedamas ėjimas link gydymo obelisk loiacijos. Kai pasigyro enemy, jis grįžta prie patruliavimo.

## Literature list

1. Source #1. *Url*
2. Source #2. *Url*
3. ...
4. Source #N. *Url*

## ANNEX

All source code is contained in this part.