

**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF INFORMATICS**

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

*3D Labyrinth*

*IFF 8/7 Gvidas  
Raškevičius*

Date: 2021.05.11

Kaunas, 2021

## Tables of Contents

<i>Tables of Images</i> .....	3
<i>Table of Tables/functions</i> .....	4
<i>Description of Your Game</i> .....	5
<i>Laboratory work #1</i> .....	6
List of tasks.....	6
<i>Solution</i> .....	6
Task #1. <i>Getting a Character</i> .....	6
Task #2. <i>Making character move with animations</i> .....	7
Task #3. <i>Adding more game objects and lights</i> .....	9
Task #4. <i>Making player die when saw touches him and giving points when collecting it</i> .....	11
Task #5. <i>Stopping the game</i> .....	11
Defense task. <i>Make it so that the player would be able to run (sprint) for only for a certain amount of time (configurable, e.g. 5 seconds). Then after running, there would be a cool-down (configurable, e.g. 1 second) after the player could run again.</i> .....	12
<i>Laboratory work #2</i> .....	13
List of tasks.....	13
<i>Solution</i> .....	13
Task #1. <i>Building world environment</i> .....	13
Task #2. <i>Fixing Lab #1 defense task</i> .....	18
Task #3. <i>Adding animations to game character</i> .....	19
Task #4. <i>Creating / animating game objects</i> .....	19
Task #5. <i>Creating particles for game objects</i> .....	21
Task #6. <i>Adding custom skybox</i> .....	22
Task #7. <i>Creating physics material</i> .....	22
Task #8. <i>Creating objects that uses collision or triggers</i> .....	25
Task #9. <i>Baking a lightmap</i> .....	26
Defense task. - <i>Add a "wall" which would have a shattering animation - If the player touches the wall, the animation should play and the player should be able to pass through the shattered wall - When the animation finishes, a "dust" particle effect should play out</i> .....	26
<i>Laboratory work #3</i> .....	28
List of tasks.....	28
<i>Solution</i> .....	28
Task #1. <i>Adding menu system</i> .....	28
Task #2. <i>Adding options menu</i> .....	29
Task #3. <i>Adding GUI</i> .....	30
Task #4. <i>Adding scoring system based on time / points / health collected</i> .....	32
Task #5. <i>Game over condition</i> .....	33
Task #6. <i>Adding shooting</i> .....	33

<b>Task #7. Interactive sounds .....</b>	<b>33</b>
<b>Task #8. Post processing .....</b>	<b>36</b>
<b>Defense task - Save player score to disk - When the player finishes the game, show a list of previous player scores - Names are not important, can be random number or random string.....</b>	<b>36</b>
<b>User's manual.....</b>	<b>38</b>
<b>Literature list.....</b>	<b>39</b>
<b>ANNEX .....</b>	<b>40</b>

## Tables of Images

Figure 1. Character .....	6
Figure 2. Wall .....	9
Figure 3. Sawblade .....	10
Figure 4. Point .....	10
Figure 5. Start end platforms .....	10
Figure 6. Stamina UI .....	18
Figure 7. Animations .....	19
Figure 8. Point .....	19
Figure 9. Health point.....	20
Figure 10. Sawblade .....	20
Figure 11. Ice floor .....	23
Figure 12. Honey floor .....	23
Figure 13. Regular floor .....	23
Figure 14. Main menu with a baked lightmap.....	26
Figure 15. Main menu .....	28
Figure 16 Options menu .....	30
Figure 17. GUI.....	31
Figure 18. Score.....	32
Figure 19 Game over .....	33
Figure 20 Audio mixers .....	34
Figure 21 Without post processing.....	36
Figure 22 With post procesing.....	36
Figure 23 Registry file with saved data .....	37
Figure 24. Main Menu .....	38
Figure 25. Levels menu .....	38

## Table of Tables/functions

Table 1. Move function .....	8
Table 2. OnTriggerEnter and OnCollisionEnter function .....	11
Table 3. player.Win(), player.Lose() functions .....	12
Table 4. player.Defense task.....	12
Table 5. MazeGenerator .....	14
Table 6. Randomly generated objects.....	15
Table 7. Starting point, floors generated .....	16
Table 8. Generating the maze .....	17
Table 9. Stamina for running and regenerating it .....	18
Table 10. Setting the threshold. ....	19
Table 11. Heart controller.....	20
Table 12. Sawblade controller .....	21
Table 13. Particles .....	22
Table 14. Different physics materials.....	24
Table 15. Triggers and collisions .....	25
Table 16. Defense task.....	27
Table 17 Main menu controller functions .....	29
Table 18. Settings controller.....	30
Table 19. Code to control GUI .....	31
Table 20. Code for score calculations .....	32
Table 21. Game over condition .....	33
Table 22 Shooting implementation .....	33
Table 23 Music selector script for controlling the audio in game.....	35
Table 24 Defense task.....	37

## Description of Your Game

Description of Your Game.

1. 3D or 2D? **3D**
2. What type is your game? Puzzle
3. What genre is your game? Racing
4. Platforms (mobile, PC or both?) **PC**
5. Scenario Description. *Žaidimo idėja – išeiti labirintą per kuo trumpesnį laiką. Žaidimo veikėjas ( nenuspręsta kas, žmogus, robotas ar kažkas kitas) turi rasti išėjimą per labirintą pilną kliūčių (tarpai kuriuos reik prašokti, judantys pjūklai ir t.t) per kuo trumpesnį laiką. Kuo ilgiau jis užtrunka tuo mažiau taškų gale gauna. Duodamos 3 gyvybės. Kiekvienam lygį yra checkpoint, kuriuos praėjus jeigu žaidėjas miršta , nusikelia į checkpointus. Jei išnaudojamos visos gyvybės, skelbiamas game over ir žaidėjas lygį turi pradėti nuo pradžių. Kuo didesnis lygis, tuo sunkesni ir ilgesni lygiai. Lygių generavimui bandyčiau sukurti algortimą kad juos generuotų automatiškai, tačiau nesu tikras ar tai pavyks įgyvendinti tai neatmetu varianto, kad juos galėčiau generuoti tik rankiniu būdu.*

## Laboratory work #1

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

1. Getting a Character
2. Making character movement with animations
3. Adding more game objects and lights
4. Making player die when saw touches him and giving points when player collects them
5. Stopping the game

### Solution

#### Task #1. *Getting a Character*

Description of the implementation (3-5 sentences). For character I chose free character from asset store ([Source #1](#)). It has free textures as well as animations, so I chose it to quick understand how everything works. For future I think I will be able to create my own character with Blender, or make it from objects in Unity itself.

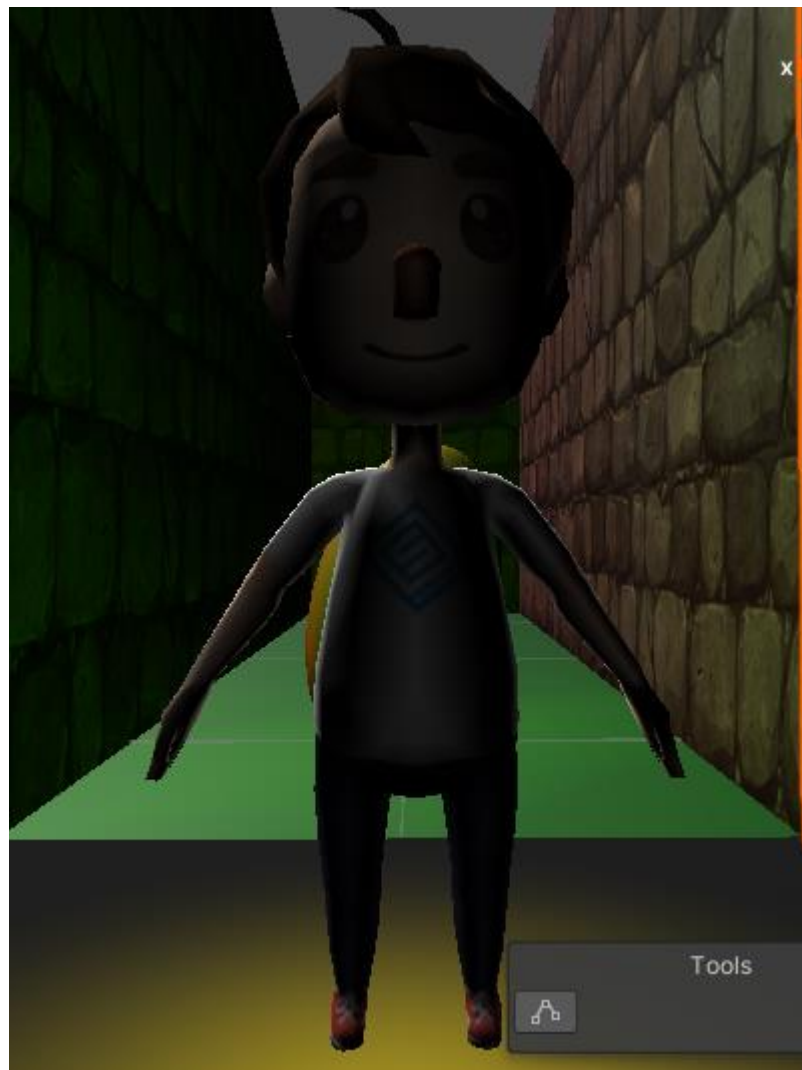


Figure 1. Character

## Task #2. *Making character move with animations*

Description of the implementation (3-5 sentences). For making my character move, I created a movementController. It has these properties:

- movementSpeed – how fast character moves at the moment.
- walkingSpeed – how fast character moves when walking.
- runningSpeed - how fast character moves when running.
- jumpHeight – how high can character jump.
- isGrounded – checks whether the character is on the ground
- groundCheckDistance – distance when character is on the ground and when it's not. For instance, if you set it 2, the character will be in y level 2 and it will understand that character is on the ground.
- groundMask – layer that is defined as a ground
- gravity – Gravity for when character jumps. It may be set, however I intend to make it static in future.

As well as properties, my controller has references to CharacterController and Animator for animations.

Main functionality is in Move() function. Additional to that there is Jump, Run, Walk and Idle functions.

```

private void Move()
{
    isGrounded = Physics.CheckSphere(transform.position, groundCheckDistance,
groundMask);

    if(isGrounded && velocity.y < 0)
    {
        velocity.y = -2f;
    }

    float moveZ = Input.GetAxis("Vertical");
    float moveX = Input.GetAxis("Horizontal");

    moveDirection = new Vector3(moveX, 0, moveZ);
    moveDirection = transform.TransformDirection(moveDirection);
    if (isGrounded)
    {
        if(moveDirection != Vector3.zero && !Input.GetKey(KeyCode.LeftShift))
        {
            Walk();
        }
        else if(moveDirection != Vector3.zero && Input.GetKey(KeyCode.LeftShift)){
            Run();
        }
        else if(moveDirection == Vector3.zero)
        {
            Idle();
        }
        moveDirection *= movementSpeed;

        if (Input.GetKeyDown(KeyCode.Space))
        {
            Jump();
        }
    }
    controller.Move(moveDirection * Time.deltaTime); //movement
    velocity.y += gravity * Time.deltaTime;
    controller.Move(velocity * Time.deltaTime); //gravity
}

```

**Table 1. Move function**



```

private void Walk()
{
    movementSpeed = walkingSpeed;
    anim.SetFloat("Speed", 0.25f, 0.1f, Time.deltaTime);
}

private void Run()
{
    movementSpeed = runningSpeed;
    anim.SetFloat("Speed", 0.5f, 0.1f, Time.deltaTime);
}

private void Idle()
{
    anim.SetFloat("Speed", 0, 0.1f, Time.deltaTime);
}

private void Jump()
{
    velocity.y = Mathf.Sqrt(jumpHeight * -2 * gravity);
    movementSpeed = walkingSpeed;
    anim.SetFloat("Speed", 0.75f);
}

```

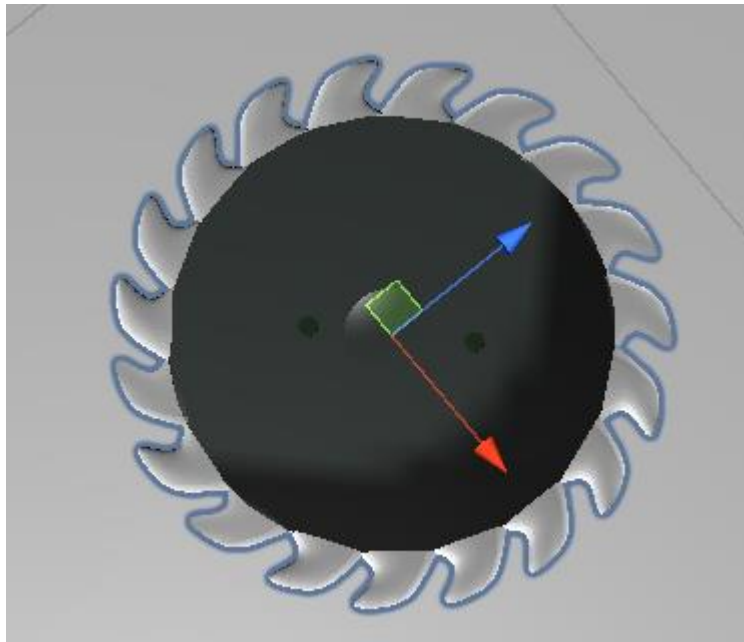
**Table 3. Additional functions**

### Task #3. *Adding more game objects and lights*

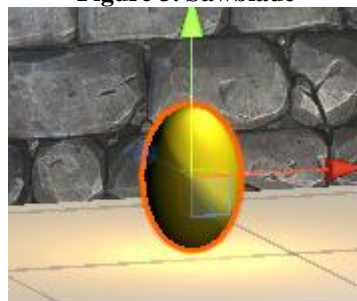
Description of the implementation (3-5 sentences). For game objects I needed walls to generate a labyrinth with, as well as ground, starting and end platforms. Also I added saw that I drew myself with Blender and animated it to spin. Also, added points that I created with unity 3d object sphere.



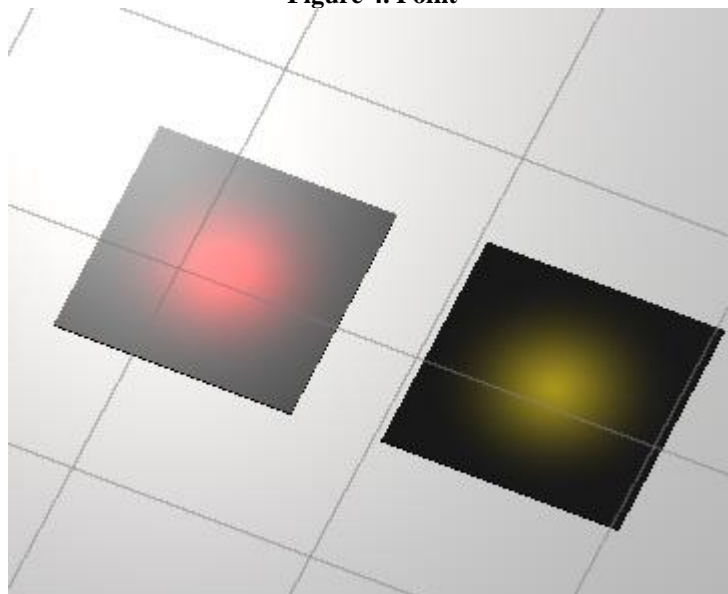
**Figure 2. Wall**



**Figure 3. Sawblade**



**Figure 4. Point**



**Figure 5. Start end platforms**

#### Task #4. *Making player die when saw touches him and giving points when collecting it*

Description of the implementation (3-5 sentences). For that, I have created a triggerController for player. There is function onTriggerEnter and onCollisionEnter. onTriggerEnter is for collecting points, onCollisionEnter is for detecting collision with the saw, and if it does that stops the game and says that it is game over.

```
private void OnTriggerEnter(Collider other)
{
    var otherGameObject = other.gameObject;
    var collected = false;

    if(other.tag == "Point")
    {
        player.AddPoints(pointValue);
        collected = true;
    }else if(other.tag == "Finish")
    {
        player.Win();
    }

    if (collected)
    {
        otherGameObject.SetActive(false);
        Destroy(otherGameObject);
    }
}

void OnCollisionEnter(Collision other)
{
    var otherGameObject = other.gameObject;
    if (otherGameObject.tag == "Saw")
    {
        player.Lose();
    }
}
```

Table 2. OnTriggerEnter and OnCollisionEnter function

#### Task #5. *Stopping the game*

Description of the implementation (3-5 sentences). It is quite simple code, if player steps on end platform, OnTriggerEnter from previous code activates and calls player.Win() function. If it gets killed by saw, it calls player.Lose() function. If you wish to reset the scene for a new round, I implemented simple code by pressing R you just refresh the scene.

```

public void Win()
{
    pointsText.gameObject.SetActive(false);
    movementController.enabled = false;
    finishText.text = $"You win!\nYou scored {points} points.";
    finishText.gameObject.SetActive(true);
}
public void Lose()
{
    pointsText.gameObject.SetActive(false);
    movementController.enabled = false;
    finishText.text = $"You lose!\nYou scored {points} points.";
    finishText.gameObject.SetActive(true);
}

```

Table 3. player.Win(), player.Lose() functions

**Defense task.** *Make it so that the player would be able to run (sprint) for only for a certain amount of time (configurable, e.g. 5 seconds). Then after running, there would be a cool-down (configurable, e.g. 1 second) after the player could run again.*

Description of the implementation (3-5 sentences). I tried to make that you set starting time when it starts to run and a simple if statement whether to check if that time is higher than runningTime set and cooldown, however after player ran for given time, it couldn't run no more, a bug with a time I believe.

```

private void Run()
{
    float start = Time.time;
    if (start > seconds + cooldown )
    {
        start = Time.time ;
        Debug.Log($"Please wait {cooldown}s before sprinting again.");
        movementSpeed = walkingSpeed;
        anim.SetFloat("Speed", 0.25f, 0.1f, Time.deltaTime);
    }
    else
    {
        movementSpeed = runningSpeed;
        anim.SetFloat("Speed", 0.5f, 0.1f, Time.deltaTime);
    }
}

```

Table 4. player.Defense task

## Laboratory work #2

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

1. Building world environment
2. Fixing Lab #1 defense task
3. Adding animations to game character
4. Creating / animating game objects
5. Creating particles for game objects
6. Adding custom skybox
7. Creating physics materials
8. Creating objects that uses different colliders or triggers
9. Baking a lightmap

### Solution

#### Task #1. *Building world environment*

Description of implementation (3-5 sentences). *This part is made out of two scripts : Maze generator and maze renderer.*

- *Maze generator works with a backtracker algorithm. The idea is simple – you give it width and height of the maze, and it randomly chooses the starting point. The algorithm is designed that it starts at random position and finds all neighbours that he can visit. Once it visits a coordinate, it is added to the stack. If there is no unvisited neighbours from the last coordinate, he pops the stack and looks for unvisited neighbours from previous coordinate and it continues until the stack is empty.*
- *Maze renderer then gets the stack with coordinates that maze is made of, and puts the prefabs that we give it for floors and walls to make the maze. It is pushed to the sides that it would make a path instead of the walls. We also generate random objects like ice floor, honey floor, saws, life points and points in the maze in this part. For that we have different prefabs ready and functions that instantiate these objects.*

```

private static Wall[,] generateMaze(Wall[,] maze, int width, int height, int seed)
{
    var rng = new System.Random(seed);
    var positionStack = new Stack<Position>();
    var position = new Position { X = rng.Next(0,width), Y = rng.Next(0, height)
};

    maze[position.X, position.Y] |= Wall.VISITED; // 1000 1111
    positionStack.Push(position);

    while(positionStack.Count > 0)
    {
        var current = positionStack.Pop();
        var neighbours = getUnvisitedNeighbours(current, maze, width, height);
        if(neighbours.Count > 0)
        {
            positionStack.Push(current);

            var randIndex = rng.Next(0, neighbours.Count);
            var randomNeighbour = neighbours[randIndex];

            var nPosition = randomNeighbour.Position;
            maze[current.X, current.Y] &= ~randomNeighbour.SharedWall;
            maze[nPosition.X, nPosition.Y] &=
~getOppositeWall(randomNeighbour.SharedWall);

            maze[nPosition.X, nPosition.Y] |= Wall.VISITED;

            positionStack.Push(nPosition);
        }
    }

    return maze
}

```

**Table 5. MazeGenerator**

```

void GenerateSaws()
{
    var r = new System.Random();
    for (int i = 0; i < NumOfSaws; i++)
    {
        int rX = r.Next((-Width - 1) / 2, Height / 2);
        int rY = r.Next((-Height - 1) / 2, Height / 2);
        var sawblade = Instantiate(sawPrefab, transform);
        var sawController = GetComponentInChildren<SawMovement>();
        if( i % 2==0 ) sawController.toggleXAxis();
        sawblade.position = new Vector3(rX, 0.5f, rY);
    }
}

void GeneratePoints()
{
    var r = new System.Random();
    for (int i = 0; i < NumOfPoints; i++)
    {
        int rX = r.Next((-Width - 1) / 2, Height / 2);
        int rY = r.Next((-Height - 1) / 2, Height / 2);
        var point = Instantiate(pointPrefab, transform);
        point.position = new Vector3(rX, 0.25f, rY);
    }
}

void GenerateHearts()
{
    var r = new System.Random(23);
    for (int i = 0; i < NumOfHearts; i++)
    {
        int rX = r.Next((-Width - 1) / 2, Height / 2);
        int rY = r.Next((-Height - 1) / 2, Height / 2);
        var point = Instantiate(heartPrefab, transform);
        point.position = new Vector3(rX, 0f, rY);
    }
}

```

**Table 6. Randomly generated objects**

```

void generateStartEnd()
{
    var startPlatform = Instantiate(startObject, transform);
    startPlatform.position = new Vector3(0, 0, 0);

    var endPlatform = Instantiate(endObject, transform);
    var rnd = Random.Range(1, 4);
    switch (rnd)
    {
        case 1:
            endPlatform.position = new Vector3(-Width / 2, 0, (Height - 1) / 2);
            break;
        case 2:
            endPlatform.position = new Vector3((Width - 1) / 2, 0, -Height / 2);
            break;
        case 3:
            endPlatform.position = new Vector3((Width - 1) / 2, 0, (Height - 1) /
2);
            break;
        case 4:
            endPlatform.position = new Vector3(-Width / 2, 0, -Height / 2);
            break;
    }
}

public void generatePlayer()
{
    playerPrefab.position = new Vector3(0, 0, 0);
}

void placeIceFloor(Vector3 position)
{
    var icyFloor = Instantiate(icyFloorPrefab, transform);
    icyFloor.localScale = new Vector3(icyFloor.localScale.x,
icyFloor.localScale.y, icyFloor.localScale.z);
    icyFloor.position = position + new Vector3(0, -1, 0);
}

void placeHoney(Vector3 position)
{
    var honeyFloor = Instantiate(honeyPrefab, transform);
    honeyFloor.localScale = new Vector3(honeyFloor.localScale.x,
honeyFloor.localScale.y, honeyFloor.localScale.z);
    honeyFloor.position = position + new Vector3(0, -1, 0);
}

void placeFloor(Vector3 position)
{
    var floor = Instantiate(floorPrefab, transform);
    floor.localScale = new Vector3(floor.localScale.x, floor.localScale.y,
floor.localScale.z);
    floor.position = position + new Vector3(0, -1, 0);
}

```

**Table 7. Starting point, floors generated**



```

private void Draw(Wall[,] maze)
{
    generatePlayer();
    generateStartEnd();
    GenerateSaws();
    GeneratePoints();
    GenerateHearts();
    for (int i = 0; i < Width; i++)
    {
        for (int j = 0; j < Height; j++)
        {
            var cell = maze[i, j];
            var position = new Vector3(-Width / 2 + i, 1f, -Height / 2 + j); //OFFSET of the
middle of the cell, so there's a path
            var rnd = Random.Range(1, 11);

            if(rnd > 6 && rnd <=9)
            {
                placeIceFloor(position);
            }
            else if(rnd > 9)
            {
                placeHoney(position);
            }
            else
            {
                placeFloor(position);
            }
            if (cell.HasFlag(Wall.UP))
            {
                var topWall = Instantiate(wallPrefab, transform) as Transform;
                topWall.position = position + new Vector3(0, 0, size/2);
                topWall.localScale = new Vector3(size, topWall.localScale.y,
topWall.localScale.z);
            }
            if (cell.HasFlag(Wall.LEFT))
            {
                var leftWall = Instantiate(wallPrefab, transform) as Transform;
                leftWall.position = position + new Vector3(-size / 2, 0, 0);
                leftWall.localScale = new Vector3(size, leftWall.localScale.y,
leftWall.localScale.z);
                leftWall.eulerAngles = new Vector3(0, 90, 0);
            }
            if (i == Width - 1)
            {
                if (cell.HasFlag(Wall.RIGHT))
                {
                    var rightWall = Instantiate(wallPrefab, transform) as Transform;
                    rightWall.position = position + new Vector3(size / 2, 0, 0);
                    rightWall.localScale = new Vector3(size, rightWall.localScale.y,
rightWall.localScale.z);
                    rightWall.eulerAngles = new Vector3(0, 90, 0);
                }
            }
            if (j == 0)
            {
                if (cell.HasFlag(Wall.DOWN))
                {
                    var bottomWall = Instantiate(wallPrefab, transform) as Transform;
                    bottomWall.position = position + new Vector3(0, 0, -size / 2);
                    bottomWall.localScale = new Vector3(size, bottomWall.localScale.y,
bottomWall.localScale.z);
                }
            }
        }
    }
}

```

**Table 8. Generating the maze**

## Task #2. Fixing Lab #1 defense task

Description of implementation (3-5 sentences). *Added a user interface bar for stamina, that is getting lower once you run. After the set time of cooldown, it starts to generate back if you do not run. If you run out of stamina, you're not able to run anymore and have to wait until it charges back up.*



Figure 6. Stamina UI

```
[Range(0,10)][SerializeField] private float stamina=10;
[Range(0, 10)] [SerializeField] private float maxStamina = 10;
private float StaminaRegenTimer = 0.0f;
private const float staminaDecrease = 2.0f;
private const float staminaIncrease = 5.0f;

private void Run()
{
    if (stamina > 0)
    {
        stamina = Mathf.Clamp(stamina - (staminaDecrease * Time.deltaTime), 0.0f,
maxStamina);
        movementSpeed = runningSpeed;
        anim.SetFloat("Speed", 0.5f, 0.1f, Time.deltaTime);
        StaminaRegenTimer = 0.0f;
    }
    else
    {
        Walk();
    }
}

private void RegenerateStamina()
{
    if (StaminaRegenTimer >= cooldown)
    {
        stamina = Mathf.Clamp(stamina + (staminaIncrease * Time.deltaTime), 0.0f,
maxStamina);
    }
    else
    {
        StaminaRegenTimer += Time.deltaTime;
    }
}
```

Table 9. Stamina for running and regenerating it

### Task #3. Adding animations to game character

Description of implementation (3-5 sentences). *I used a character from asset store, so it had his animations ready, I only had to implement it, which I did. Implementation is pretty simple, I just have to set what Threshold to use when walking, running, jumping and idling.*

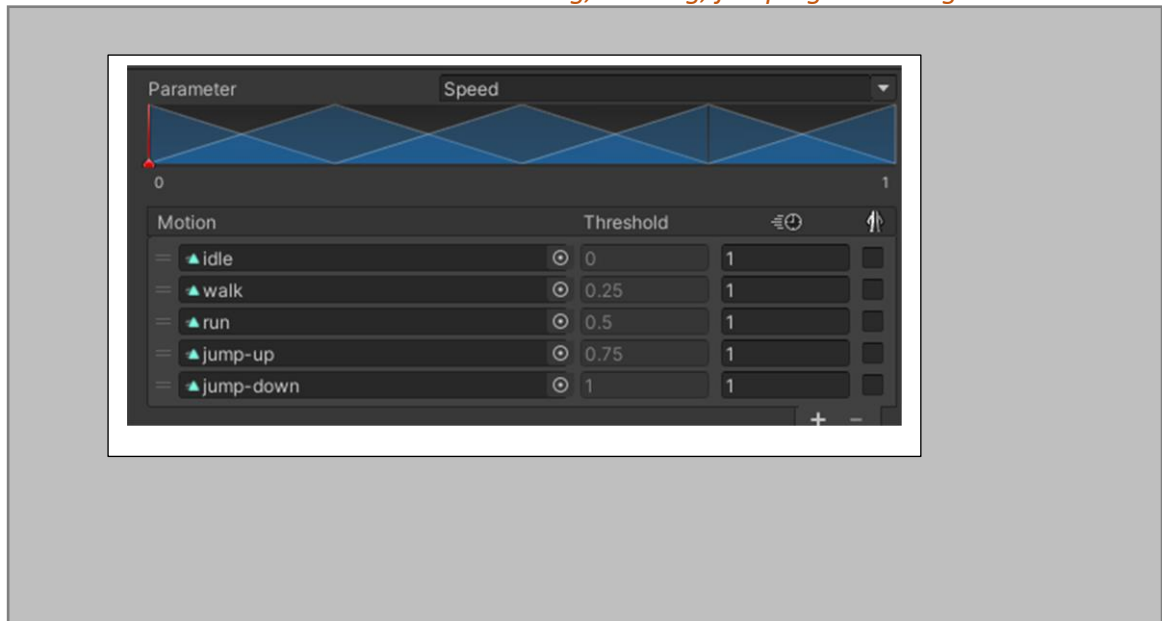


Figure 7. Animations

```
anim.SetFloat("Speed", 0,0.1f,Time.deltaTime);  
anim.SetFloat("Speed", 0.25f, 0.1f, Time.deltaTime);  
anim.SetFloat("Speed", 0.5f, 0.1f, Time.deltaTime);  
anim.SetFloat("Speed", 0.75f);
```

Table 10. Setting the threshold.

### Task #4. Creating / animating game objects

Description of implementation (3-5 sentences). *I added Sawblade, Life point and point objects to the game. For other 2 game objects I would consider them starting and ending platforms.*

- Sawblade was created by me in blender, I animated it there as well. It has only spinning animation, and for movement, I made a simple script that controls the movement of the blade by moving it on x or y axis randomly by set distance.
- Point was created from a simple sphere in unity, animation was only adding some particles and making a shaders that would look like pulsating
- Health point was taken from a internet, link to it is added in the source list. For animation I just made a simple controller that spins the heart and added shaders for pulsating look.

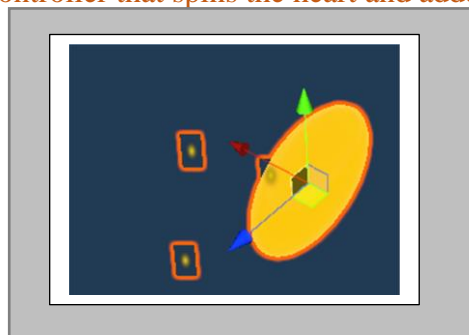
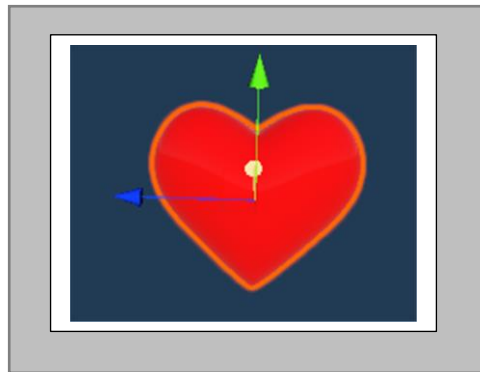
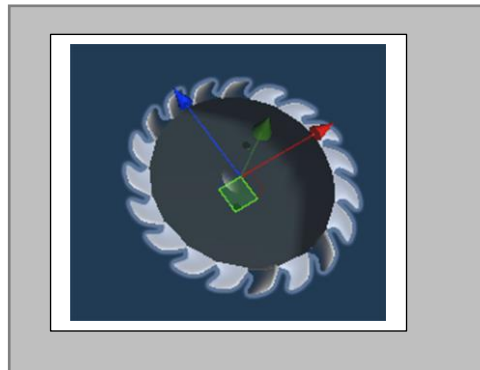


Figure 8. Point



**Figure 9.** Health point



**Figure 10.** Sawblade

```
public class HeartController : MonoBehaviour
{
    void Update()
    {
        Spin();
    }

    void Spin()
    {
        transform.Rotate(0, 50*Time.deltaTime, 0);
    }
}
```

**Table 11.** Heart controller

```

public class SawMovement : MonoBehaviour
{
    [SerializeField] private float speed = 1.0f;
    [SerializeField] private float blockMovement = 3;

    [SerializeField] private bool xAxis = false;
    private Vector3 pos;
    private Vector3 posX;
    private Vector3 posY;

    void Start()
    {
        pos = transform.position;
        posX = new Vector3(pos.x + blockMovement, pos.y, pos.z);
        posY = new Vector3(pos.x, pos.y, pos.z + blockMovement);
    }
    void Update()
    {
        if (xAxis)
        {
            xAxisMovement();
        }
        else
        {
            yAxisMovement();
        }
    }

    void xAxisMovement()
    {
        transform.position = Vector3.Lerp(pos, posX, Mathf.PingPong(Time.time * speed, 1.0f));
    }
    void yAxisMovement()
    {
        transform.position = Vector3.Lerp(pos, posY, Mathf.PingPong(Time.time * speed, 1.0f));
    }

    public void toggleXAxis()
    {
        xAxis = !xAxis;
    }
}

```

Table 12. Sawblade controller

## Task #5. Creating particles for game objects

Description of implementation (3-5 sentences). *For particles, I used 4 of them at the moment, did not have where to use a 5<sup>th</sup> one, unfortunately. I implemented it on heart pickup, for emitting particles to show player where is the points, when he get hits by the sawblade and when it steps on the icy floor.*

```

private void createHeartParticles(Vector3 position)
{
    Instantiate(heartParticlePrefab, position, Quaternion.identity);
}
private void createBloodParticles(Vector3 position)
{
    Instantiate(bloodPrefab, position, Quaternion.identity);
}

private void OnTriggerEnter(Collider other)
{
    var otherGameObject = other.gameObject;
    var collected = false;
    else if(other.tag == "Heart")
    {
        player.AddHealthPoint(heartValue);
        SoundSource.PlayOneShot(heartSound);
        createHeartParticles(otherGameObject.transform.position);
        collected = true;
    }

    if (otherGameObject.tag == "Saw")
    {
        player.removeHealthPoint();
        createBloodParticles(other.transform.position);
        SoundSource.PlayOneShot(sawSound);
    }
}

/////
if (icy)
{
    moveDirection = lastMoveDirection * slideSpeed;
    if (!isParticleRunning)
    {
        Instantiate(snowParticles,transform.localPosition,
Quaternion.identity);
        isParticleRunning = true;
    }
    canJump = true;
}

```

**Table 13. Particles**

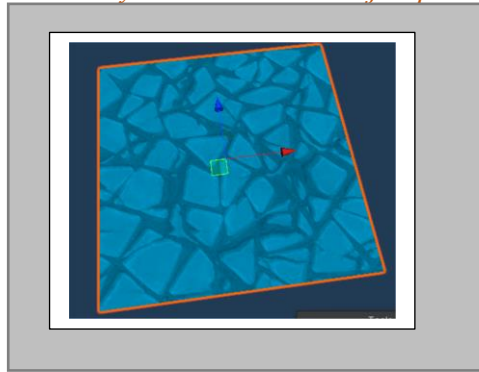
## Task #6. Adding custom skybox

Description of implementation (3-5 sentences). *At this step, I just took the free skybox assets from asset store, which had cubemaps, so it gives me a good quality skyboxes and I can change them if I wish easily as well. The source for it will be in the source lists.*

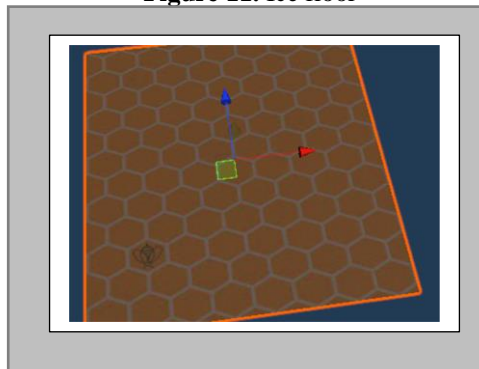
## Task #7. Creating physics material

Description of implementation (3-5 sentences). *I have decided to go with 3 physics materials – solid ground, icy ground and honey combs. If you step on icy, it gives you a feeling that you're*

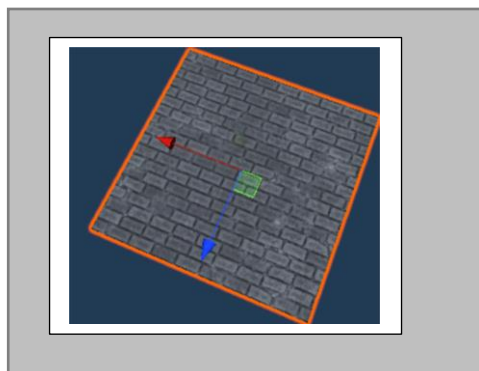
*sliding by taking your last position and multiplying it by slide speed that you set. On honey comb, you're moving a lot slower and you are not able to jump at all.*



**Figure 11.** Ice floor



**Figure 12.** Honey floor



**Figure 13.** Regular floor

```

private bool icy = false;
private bool honey = false;
void OnControllerColliderHit(ControllerColliderHit hit)
{
    icy = hit.collider.CompareTag("Ice");
    honey = hit.collider.CompareTag("Honey");
}
private void Move()
{
    isGrounded = Physics.CheckSphere(transform.position, groundCheckDistance, groundMask);
    if(isGrounded && velocity.y < 0)
    {
        velocity.y = -2f;
    }
    float moveZ = Input.GetAxis("Vertical");
    float moveX = Input.GetAxis("Horizontal");
    float inputMagnitude = Mathf.Min(new Vector3(moveX, 0, moveZ).sqrMagnitude, 1f);
    moveDirection = new Vector3(moveX, 0, moveZ);
    moveDirection = transform.TransformDirection(moveDirection);
    if(inputMagnitude > 0.225f)
    {
        lastMoveDirection = moveDirection;
    }
    if (isGrounded)
    {
        if (icy)
        {
            moveDirection = lastMoveDirection * slideSpeed;
            if (!isParticleRunning)
            {
                Instantiate(snowParticles,transform.localPosition, Quaternion.identity);
                isParticleRunning = true;
            }
            canJump = true;
        }
        else if(honey)
        {
            DestroyAll("Snow");
            isParticleRunning = false;
            moveDirection *= honeySpeed;
            canJump = false;
        }
        else
        {
            DestroyAll("Snow");
            isParticleRunning = false;
            canJump = true;
            if(moveDirection != Vector3.zero && !Input.GetKey(KeyCode.LeftShift))
            {
                Walk();
            }
            else if(moveDirection != Vector3.zero && Input.GetKey(KeyCode.LeftShift)){
                Run();
            }
            else if(moveDirection == Vector3.zero)
            {
                Idle();
            }
            moveDirection *= movementSpeed;
        }

        if (Input.GetKeyDown(KeyCode.Space))
        {
            Jump(canJump);
        }
    }
    controller.Move(moveDirection * Time.deltaTime); //movement
    velocity.y += gravity * Time.deltaTime;
    controller.Move(velocity * Time.deltaTime); //gravity
}

```

**Table 14. Different physics materials**



## Task #8. Creating objects that uses collision or triggers

Description of implementation (3-5 sentences). *My previously showed game objects uses colliders and triggers. Heart point on trigger enter gets destroyed, lets out a sound and burst a particle effects. Point also lets out a sound and gets destroyed. My saw blade on collision enter takes out a life of your life points and plays a sound with activating blood particle effects as well. Also, there is an trigger on the finish tile, when you step on it, a win menu displays allowing you to go to next level.*

```
private void OnTriggerEnter(Collider other)
{
    var otherGameObject = other.gameObject;
    var collected = false;

    if(other.tag == "Point")
    {
        player.AddPoints(pointValue);
        SoundSource.PlayOneShot(pointSound);
        collected = true;
    }else if(other.tag == "Finish")
    {
        player.Win();
        SoundSource.PlayOneShot(finishSound);
    }
    else if(other.tag == "Heart")
    {
        player.AddHealthPoint(heartValue);
        SoundSource.PlayOneShot(heartSound);
        createHeartParticles(otherGameObject.transform.position);
        collected = true;
    }

    if (collected)
    {
        otherGameObject.SetActive(false);
        Destroy(otherGameObject);
    }
}

void OnCollisionEnter(Collision other)
{
    var otherGameObject = other.gameObject;
    if (otherGameObject.tag == "Saw")
    {
        player.removeHealthPoint();
        createBloodParticles(other.transform.position);
        SoundSource.PlayOneShot(sawSound);
    }
}
```

Table 15. Triggers and collisions

## Task #9. Baking a lightmap

Description of implementation (3-5 sentences). *Since my levels is generated automatically, I am not able to bake a lightmap, so I made it on the main menu, that took sample scene with a baked lightning.*



**Figure 14.** Main menu with a baked lightmap

**Defense task.** - Add a "wall" which would have a shattering animation - If the player touches the wall, the animation should play and the player should be able to pass through the shattered wall - When the animation finishes, a "dust" particle effect should play out

Description of the implementation (3-5 sentences). I made one singular wall, then made another prefab with a lot of small cubes that makes the same wall, once the collider triggers, small cubes gets spawned and the old wall gets deleted, it makes it look like it shattered, and on top of that I let out particle system.

```

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

public class WallController : MonoBehaviour
{
    [SerializeField] private GameObject shatterPrefab;
    [SerializeField] private GameObject wallParticles;
    void Start()
    {

    }

    void Update()
    {

    }

    void OnCollisionEnter(Collision other)
    {
        Debug.Log(gameObject.transform.position);
        Instantiate(shatterPrefab,gameObject.transform.position,gameObject.transform.rotation)
        ;
        Instantiate(wallParticles, transform.localPosition, Quaternion.identity);
        Destroy(gameObject);
    }
}

```

**Table 16. Defense task**

## Laboratory work #3

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

1. Adding menu system
2. Adding options menu
3. Adding GUI
4. Adding scoring system based on time / points / health collected
5. Game over condition
6. Adding shooting
7. Interactive sounds
8. Post Procesing

### Solution

#### Task #1. *Adding menu system*

Description of implementation (3-5 sentences). *Added a simple menu with options, choose level, start or quit options.*



Figure 15. Main menu

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);

```

private static void Show(Component component)
{
    component.gameObject.SetActive(true);
}
private static void Hide(Component component)
{
    component.gameObject.SetActive(false);
}

private void Start()
{
    ShowMainMenu();
}

public void StartGame()
{
    SceneManager.LoadScene("Level", LoadSceneMode.Single);
}
public void ExitGame()
{
    Scenes.ExitGame();
}

public void ShowMainMenu()
{
    Show(mainMenu);
    Hide(optionsMenu);
    Hide(levelsMenu);
}
public void ShowOptionsMenu()
{
    Show(optionsMenu);
    Hide(mainMenu);
}

public void ShowLevelsMenu()
{
    Show(levelsMenu);
    Hide(mainMenu);
}
public void SelectLevel(string levelName)
{
    SceneManager.LoadScene(levelName, LoadSceneMode.Single);
}

```

**Table 17 Main menu controller functions**

## Task #2. Adding options menu

Description of implementation (3-5 sentences). *Here I made a simple options menu for controlling effects and music sounds and if you wish to play a fullscreen or not game.*

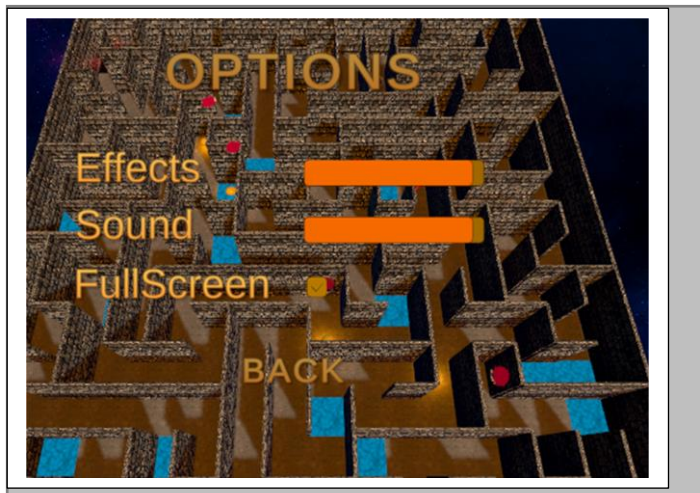


Figure 16 Options menu

```
public AudioManager musicAudioMixer;
public AudioManager effectsAudioMixer;

public Slider musicSlider;
public Slider effectSlider;

public void SetVolumeMusic(float volume)
{
    musicSlider.value = volume;
    musicAudioMixer.SetFloat("MusicVolume", volume);
}
public void SetVolumeEffects(float volume)
{
    effectSlider.value = volume;
    effectsAudioMixer.SetFloat("EffectVolume", volume);
}

public void SetFullscreen(bool isFullscreen)
{
    Screen.fullScreen = isFullscreen;
}
```

Table 18. Settings controller

### Task #3. Adding GUI

Description of implementation (3-5 sentences). *I have added a simple text GUI which best fits my needs. It has a timer, points, health point, stamina and ammo.*

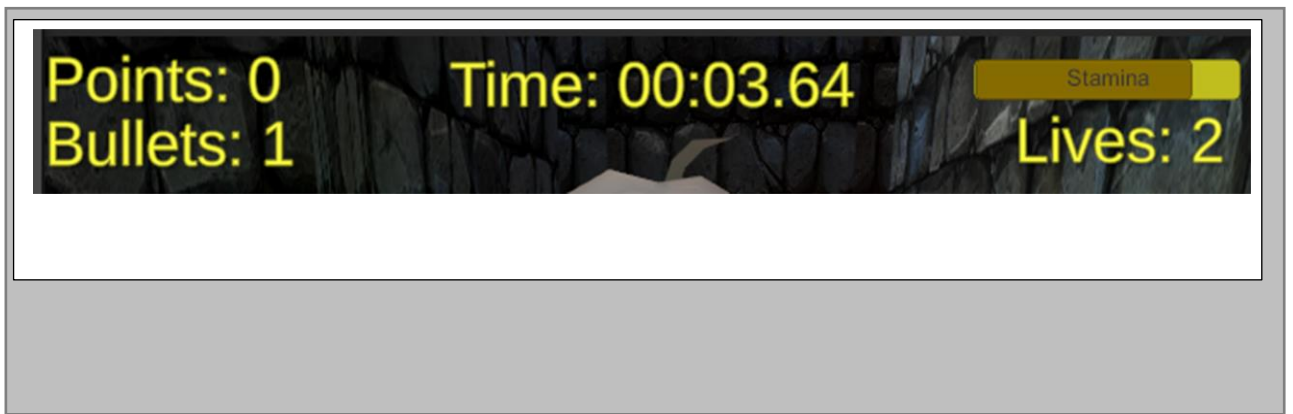


Figure 17. GUI

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);

```
public void AddPoints(int value)
{
    points += value;
    UpdatePointsText();
}
public void AddHealthPoint(int value)
{
    if(healthPoints < maxHealthPoints)
    {
        healthPoints += value;
        UpdateHealthPointsText();
    }
}
public void removeHealthPoint()
{
    if(healthPoints > 0)
    {
        healthPoints -= 1;
        UpdateHealthPointsText();
    }
    else
    {
        Lose();
    }
}
private void UpdatePointsText()
{
    pointsText.text = $"Points: {points}";
}
private void UpdateBulletsText()
{
    bulletsText.text = $"Bullets: {bullets}";
}
private void UpdateHealthPointsText()
{
    healthPointsText.text = $"Lives: {healthPoints}";
}

void Start()
{
    staminaBar.maxValue = movController.getMaxStamina();
    staminaBar.value = movController.getStamina();
}
private void Update()
{
    staminaBar.value = movController.getStamina();
    if (staminaBar.value == 0)
    {
        staminaText.text = $"{System.Math.Round(movController.getCooldown() *-1)}s. cooldown";
    }
    else
    {
        staminaText.text = "Stamina";
    }
}
```

Table 19. Code to control GUI

#### Task #4. Adding scoring system based on time / points / health collected

Description of implementation (3-5 sentences). *There was used a simple line of code to get how much time were used to completing the level, calculating the points and lives points and then taking away time spent points.*



Figure 18. Score

```
private int CalculateScore(int points,int healthPoints)
{
    int minus = 0;
    double milliseconds = timerController.getTime().TotalMilliseconds;
    if (milliseconds > 10000)
    {
        minus = -20;
    }else if(milliseconds > 15000)
    {
        minus = -50;
    }
    else if (milliseconds > 20000)
    {
        minus = -70;
    }
    else if (milliseconds > 25000)
    {
        minus = -100;
    }
    else if (milliseconds > 30000)
    {
        minus = -200;
    }
    else
    {
        minus = 0;
    }
    return points * 100 + healthPoints * 50 + minus;
}
```

Table 20. Code for score calculations



## Task #5. Game over condition

Description of implementation (3-5 sentences). *The game is over if character loses all his health points.*

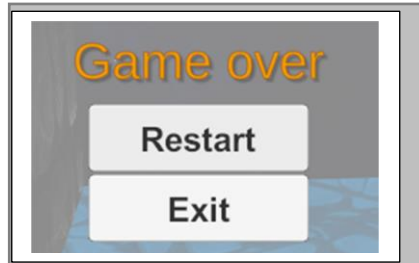


Figure 19 Game over

```
if(healthPoints > 0)
{
    healthPoints -= 1;
    UpdateHealthPointsText();
}
else
{
    Lose();
}
```

Table 21. Game over condition

## Task #6. Adding shooting

Description of implementation (3-5 sentences). *I used a simple 3 line code to spawn a new object from a given position and launching it towards front and after 3 seconds the object is destroyed. If the "bullets" tag hits the wall it gets destroyed. Haven't added animations yet to the collapsing and also need to make a protection to outer walls, however shooting is fully functional.*

```
private void Shoot()
{
    if(Input.GetButtonDown("Fire1") && bullets > 0)
    {
        Rigidbody bulletInstance = Instantiate(bulletPrefab,
firePosition.position, firePosition.rotation) as Rigidbody;
        bulletInstance.velocity = transform.TransformDirection(0, 0, bulletSpeed);
        bullets--;
        UpdateBulletsText();
        Destroy(bulletInstance.gameObject, 3);
    }
}
```

Table 22 Shooting implementation

## Task #7. Interactive sounds

Description of implementation (3-5 sentences). *I have added steps, sounds when getting hit, when on ice, main music of different 3 songs and 1 different for main menu. Used 2 different audio mixers for effect and main music to control the volume of them independently.*

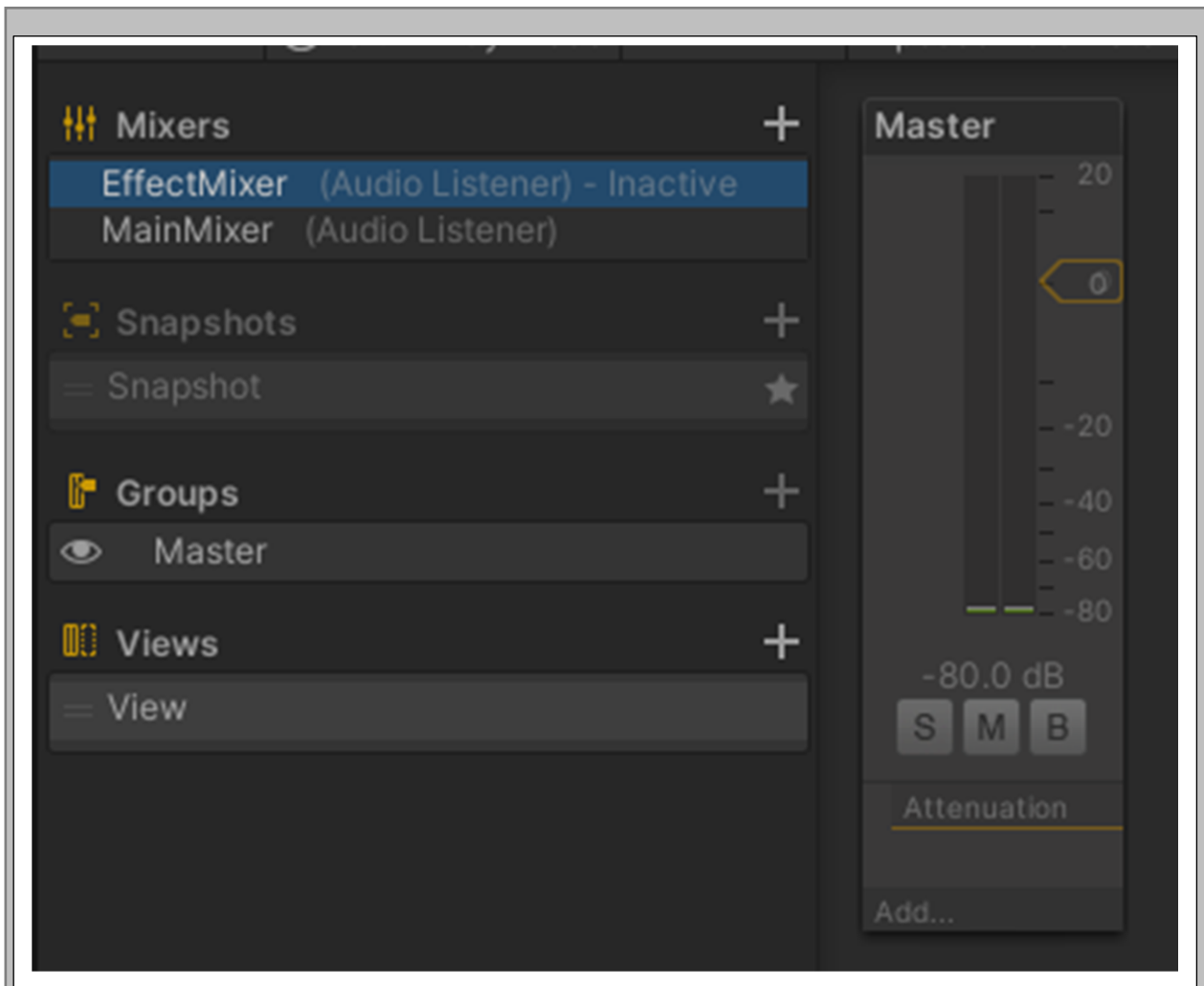


Figure 20 Audio mixers

```

public AudioSource Track1;
public AudioSource Track2;
public AudioSource Track3;

public int TrackSelector;
public int TrackHistory;

void Start()
{
    TrackSelector = Random.Range(0, 3);
    if(TrackSelector == 0)
    {
        Track1.Play();
        TrackHistory = 0;
    }
    else if(TrackSelector == 1)
    {
        Track2.Play();
        TrackHistory = 1;
    }
    else if(TrackSelector == 2)
    {
        Track3.Play();
        TrackHistory = 2;
    }
}

void Update()
{
    if (Track1.isPlaying == false && Track2.isPlaying == false && Track3.isPlaying
== false)
    {
        TrackSelector = Random.Range(0, 3);
        if (TrackSelector == 0 && TrackHistory != 0)
        {
            Track1.Play();
        }
        else if (TrackSelector == 1 && TrackHistory != 1)
        {
            Track2.Play();
        }
        else if(TrackSelector == 2 && TrackHistory != 2)
        {
            Track3.Play();
        }
    }
}

```

**Table 23 Music selector script for controlling the audio in game**

## Task #8. *Post processing*

Description of implementation (3-5 sentences). *Post processing was made using the tutorial from lecturers, it makes the world a little bit nicer, with a darker tone.*

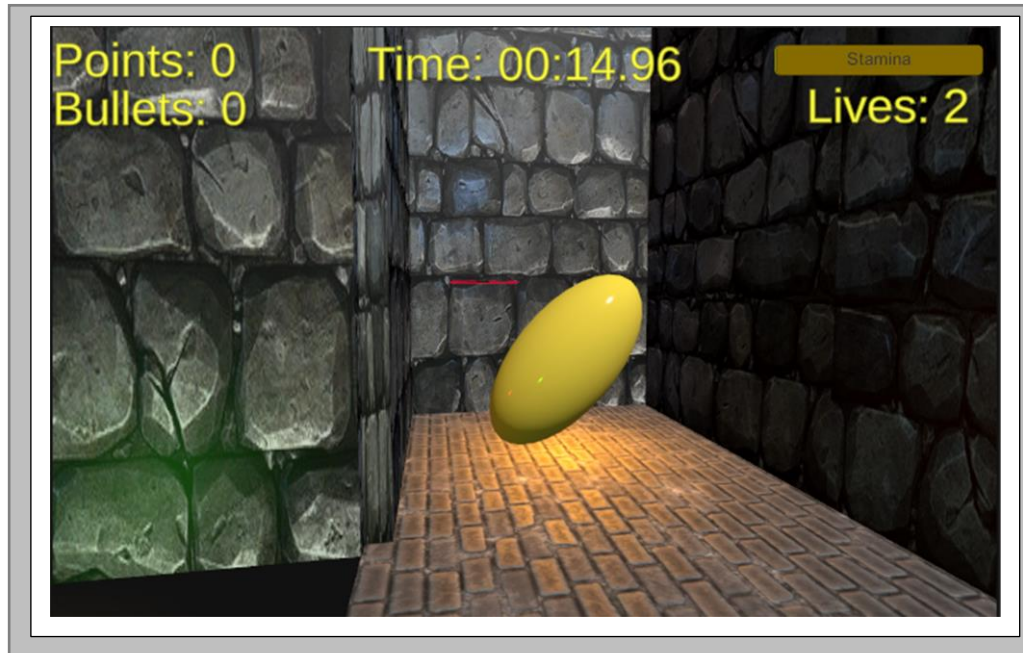


Figure 21 Without post processing

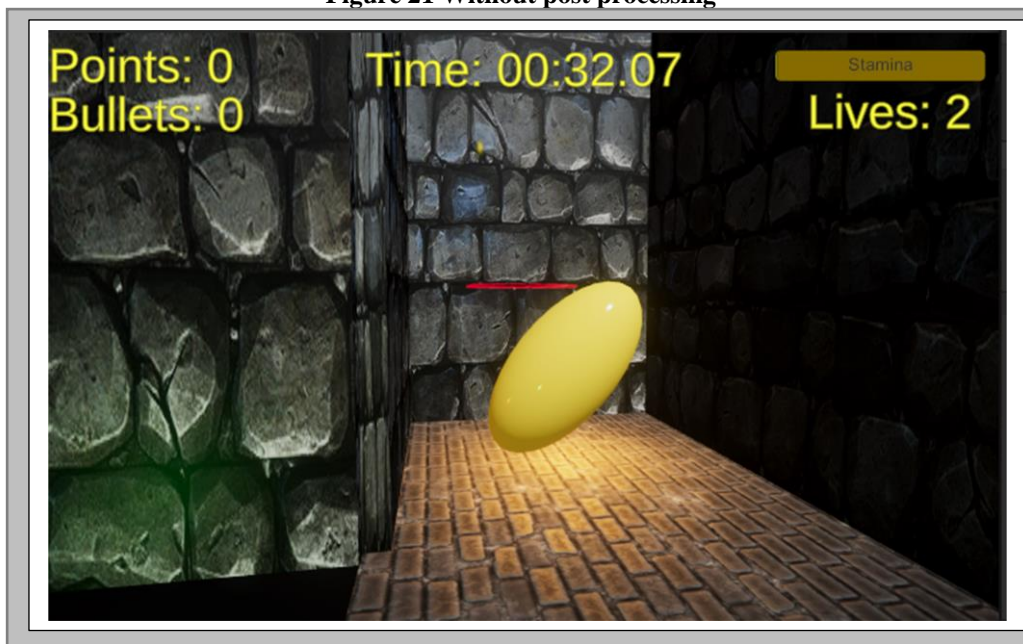


Figure 22 With post processing

Defense task - Save player score to disk - When the player finishes the game, show a list of previous player scores - Names are not important, can be random number or random string







Description of the implementation (3-5 sentences). I used PlayerPrefs to save the data to registry, with random numbers that was my mistake, should have used static names for scores, then I use the same method for retrieving them and simple GUI to show the score I retrieve from registry.

```

private void SaveScore()
{
    generatedNums = new List<int>();
    int rnd = Random.Range(0, 999);
    generatedNums.Add(rnd);
    PlayerPrefs.SetInt($"Score", CalculateScore(points,healthPoints));
    rnd++;
}
public void loadScores()
{
    score1 = PlayerPrefs.GetInt("Score718");
    scorefromDBText.text = $"Your score from DB = {score1}";
}

```

**Table 24 Defense task**

	Score1_h333546...	REG_DWORD	0x00000064 (100)
	Score211_h3071...	REG_DWORD	0x00000096 (150)
	Score42_h26961...	REG_DWORD	0x00000032 (50)
	Score718_h3071...	REG_DWORD	0x00000064 (100)
	Score966_h3071...	REG_DWORD	0x00000064 (100)
	Score988_h3071...	REG_DWORD	0x00000064 (100)

**Figure 23 Registry file with saved data**

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

**How to play?** *The goal of the game is to find the ending platform (red) and collect as many as you can points / health points as fast as you can. In order to start the game, you need to press PLAY, or go to LEVELS and choose the level you wish to play.*



Figure 24. Main Menu

*In the levels menu you can choose the level you wish to play. The higher the level, the larger the maze and the more enemies there are.*



Figure 25. Levels menu

**Descriptions of the rules of the game.** Rules is simple – finish the game as fast as possible without dying.

### Descriptions of the controls / keys.

- W,A,S,D or Arrow keys - Movement
- MOUSE – Direction the character is looking at
- SPACE – jumps
- SHIFT – runs

**You can download the game from :** <https://github.com/gviras/3DLab>

## Literature list

1. Source #1. <https://assetstore.unity.com/packages/3d/characters/humanoids/character-pack-free-sample-79870>
2. Source #2. <https://free3d.com/3d-model/valentine39s-day-heart-69073.html> - Heart model
3. Source #3. <https://assetstore.unity.com/packages/vfx/particles/spells/magic-vfx-ice-free-170242> - Ice particles
4. Source #4. <https://assetstore.unity.com/packages/vfx/particles/blood-gush-73426> - Blood particles
5. Source #5. <https://assetstore.unity.com/packages/2d/textures-materials/floors/20-man-made-ground-materials-12835> - Floor material
6. Source #6. <https://assetstore.unity.com/packages/audio/sound-fx/foley/footsteps-essentials-189879> - Sounds used
7. Source #7. <https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116> - Sounds used
8. Source #8. <https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633> - Skyboxes
9. Desert Caravan by Aaron Kenny  
Music promoted on <https://www.chosic.com/>
10. Art of silence by uniq  
<https://soundcloud.com/uniqofficial>  
Attribution 4.0 International (CC BY 4.0)  
<https://creativecommons.org/licenses/by/4.0/>  
Music promoted by <https://www.chosic.com/>
11. Time and Space by Keys of Moon | <https://soundcloud.com/keysofmoon>  
Attribution 4.0 International (CC BY 4.0)  
<https://creativecommons.org/licenses/by/4.0/>  
Music promoted by <https://www.chosic.com/>
12. Yugen - Emotional Ethnic Music by Keys of Moon | <https://soundcloud.com/keysofmoon>  
Attribution 4.0 International (CC BY 4.0)  
<https://creativecommons.org/licenses/by/4.0/>  
Music promoted by <https://www.chosic.com/>

## ANNEX

```
public class CameraCollision : MonoBehaviour
{
    private float minDistance = 0.1f;
    private float maxDistance = 0.8f;
    private float smooth = 15.0f;
    private float distance;
    Vector3 dollyDir;
    Ⓢ Unity Message | 0 references
    void Awake()
    {
        dollyDir = transform.localPosition.normalized;
        distance = transform.localPosition.magnitude;
    }

    // Update is called once per frame
    Ⓢ Unity Message | 0 references
    void Update()
    {
        Vector3 desiredCameraPos = transform.parent.TransformPoint(dollyDir * maxDistance);
        RaycastHit hit;

        if(Physics.Linecast(transform.parent.position, desiredCameraPos, out hit))
        {
            distance = Mathf.Clamp((hit.distance * 0.9f), minDistance, maxDistance);
        }
        else
        {
            distance = maxDistance;
        }
        transform.localPosition = Vector3.Lerp(transform.localPosition, dollyDir * distance, Time.deltaTime * smooth);
    }
}
```

Kodas 1 CameraCollision.cs



```

@ Unity Script | 0 references
public class CameraController : MonoBehaviour
{
    public GameObject CameraFollowObj;
    public GameObject PlayerObj;
    public Vector3 FollowPOS;
    public float CameraMoveSpeed = 120.0f;
    public float clampAngle = 80.0f;
    public float inputSensitivity = 150.0f;
    public float mouseX;
    public float mouseY;
    public float finalInputX;
    public float finalInputZ;
    private float rotY = 0.0f;
    private float rotX = 0.0f;

    @ Unity Message | 0 references
    private void Start()
    {
        Vector3 rot = transform.localRotation.eulerAngles;
        rotY = rot.y;
        rotX = rot.x;
    }

    @ Unity Message | 0 references
    private void Update()
    {
        float inputX = Input.GetAxis("RightStickHorizontal");
        float inputZ = Input.GetAxis("RightStickVertical");

        mouseX = Input.GetAxis("Mouse X");
        mouseY = Input.GetAxis("Mouse Y");

        finalInputX = inputX + mouseX;
        finalInputZ = inputZ + mouseY;

        rotY += finalInputX * inputSensitivity * Time.deltaTime;
        rotX += finalInputZ * inputSensitivity * Time.deltaTime;

        rotX = Mathf.Clamp(rotX, -clampAngle, clampAngle);

        Quaternion localRotation = Quaternion.Euler(rotX, rotY, 0.0f);
        transform.rotation = localRotation;
        PlayerObj.transform.Rotate(Vector3.up, mouseX * inputSensitivity * Time.deltaTime);
    }

    @ Unity Message | 0 references
    private void LateUpdate()
    {
        CameraUpdater();
    }

    1 reference
    private void CameraUpdater()
    {
        // set the target object to follow
        Transform target = CameraFollowObj.transform;
        // move towards the game object that is the target
        float step = CameraMoveSpeed * Time.deltaTime;

        transform.position = Vector3.MoveTowards(transform.position, target.position, step);
    }
}

```

Kodas 2 CameraController

```

[RequireComponent(typeof(TimerController))]

Unity Script | 1 reference
public class GameplayCanvasController : MonoBehaviour
{
    [Header("Objects")]
    [SerializeField] private Player player;
    [SerializeField] private TMP_Text scoreText;
    [SerializeField] private TMP_Text highScoreText;
    [Header("Menus")]
    [SerializeField] private RectTransform pausedMenu;
    [SerializeField] private RectTransform gameOverMenu;
    [SerializeField] private RectTransform winMenu;

    [Header("Maze renderer")]
    [SerializeField] private GameObject mazeRenderer;

    Scene scene = new Scene();

    private TimerController timerController;
    private int points;
    private int healthPoints;
    private int highScore;

    Unity Message | 0 references
    private void Awake()
    {
        timerController = GetComponent<TimerController>();
    }

    Unity Message | 0 references
    void Start()
    {
        scene = SceneManager.GetActiveScene();
        Hide(pausedMenu);
        Hide(gameOverMenu);
        Hide(winMenu);
        timerController.BeginTimer();

        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    Unity Message | 0 references
    private void Update()
    {
        points = player.getPoints();
        healthPoints = player.getHealthPoints();
        if (Input.GetButtonDown("Cancel"))
        {
            TogglePauseGame();
        }
    }
}

```

Kodas 3 GameplayCanvasController 1

```

0 references
private void OnDisable()
{
    Time.timeScale = 1;
}

0 references
public void ShowGameOverMenu()
{
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    Hide(pausedMenu);
    Hide(winMenu);
    Show(gameOverMenu);
    Time.timeScale = 0;
}

0 references
public void ShowWinMenu()
{
    int score = CalculateScore(points, healthPoints);
    string scoreString = $"Your score: {score}";
    scoreText.text = scoreString;
    Hide(pausedMenu);
    Show(winMenu);
    Hide(gameOverMenu);
    Time.timeScale = 0;

    SaveScore();
    loadScores();

    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
}

1 reference
public void ResumeGame()
{
    Hide(pausedMenu);
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
    Time.timeScale = 1;
}

0 references
public void RestartGame()
{
    Scenes.RestartScene();
}

0 references
public void ExitGame()
{
    SceneManager.LoadScene("Menu", LoadSceneMode.Single);
}

public void NextLevel()
{
    Scenes.LoadNextScene();
    mazeRenderer.gameObject.GetComponent<MazeRenderer>().generatePlayer();
}

```

#### Kodas 4 GameplayCanvasController 2

```

1 reference
private void TogglePauseGame()
{
    if (IsGameOver())
    {
        return;
    }
    if (IsGamePaused())
    {
        ResumeGame();
    }
    if (IsGameWon()){
        return;
    }
    else
    {
        PauseGame();
    }
}

1 reference
private void PauseGame()
{
    Show(pausedMenu);
    Time.timeScale = 0;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
}

1 reference
private bool IsGameOver()
{
    return gameOverMenu.gameObject.activeInHierarchy;
}

private bool IsGamePaused()
{
    return pausedMenu.gameObject.activeInHierarchy;
}

private bool IsGameWon()
{
    return winMenu.gameObject.activeInHierarchy;
}

private static void Show(Component component)
{
    component.gameObject.SetActive(true);
}

private static void Hide(Component component)
{
    component.gameObject.SetActive(false);
}

```

Kodas 5 GameplayCanvasController 3

```

4 references
private int CalculateScore(int points, int healthPoints)
{
    int minus = 0;
    double milliseconds = timerController.getTime().TotalMilliseconds;
    if (milliseconds > 10000)
    {
        minus = -20;
    } else if (milliseconds > 15000)
    {
        minus = -50;
    }
    else if (milliseconds > 20000)
    {
        minus = -70;
    }
    else if (milliseconds > 25000)
    {
        minus = -100;
    }
    else if (milliseconds > 30000)
    {
        minus = -200;
    }
    else
    {
        minus = 0;
    }

    int sc = points * 100 + healthPoints * 50 + minus;

    return sc;
}

1 reference
private void SaveScore()
{
    int highScore = PlayerPrefs.GetInt($"Score{scene.name}");
    if (highScore == 0)
    {
        PlayerPrefs.SetInt($"Score{scene.name}", CalculateScore(points, healthPoints));
    }
    else if (highScore < CalculateScore(points, healthPoints))
    {
        PlayerPrefs.SetInt($"Score{scene.name}", CalculateScore(points, healthPoints));
    }
}

1 reference
public void loadScores()
{
    highScore = PlayerPrefs.GetInt($"Score{scene.name}");
    highScoreText.text = $"Your highscore = {highScore}";
}

```

Kodas 6 GameplayCanvasController 4

```

@ Unity Script | 0 references
public class HeartController : MonoBehaviour
{
    @ Unity Message | 0 references
    void Update()
    {
        Spin();
    }

    1 reference
    void Spin()
    {
        transform.Rotate(0, 50*Time.deltaTime, 0);
    }
}

```

Kodas 7 HeartController

```

public class MainMenuCanvasController : MonoBehaviour
{
    [SerializeField] private RectTransform mainMenu;
    [SerializeField] private RectTransform optionsMenu;
    [SerializeField] private RectTransform levelsMenu;

    private static void Show(Component component)
    {
        component.gameObject.SetActive(true);
    }

    private static void Hide(Component component)
    {
        component.gameObject.SetActive(false);
    }

    private void Start()
    {
        ShowMainMenu();
    }

    public void StartGame()
    {
        SceneManager.LoadScene("Level", LoadSceneMode.Single);
    }

    - references
    public void ExitGame()
    {
        Scenes.ExitGame();
    }

    - references
    public void ShowMainMenu()
    {
        Show(mainMenu);
        Hide(optionsMenu);
        Hide(levelsMenu);
    }

    - references
    public void ShowOptionsMenu()
    {
        Show(optionsMenu);
        Hide(mainMenu);
    }

    - references
    public void ShowLevelsMenu()
    {
        Show(levelsMenu);
        Hide(mainMenu);
    }

    - references
    public void SelectLevel(string levelName)
    {
        SceneManager.LoadScene(levelName, LoadSceneMode.Single);
    }
}

```

Kodas 8 MainMenuCanvasController

```

[Flags]
38 references
public enum Wall
{
    //0000 -> No walls
    //1111 -> Left,right,up,down
    LEFT = 1, //0001
    RIGHT = 2, //0010
    UP = 4, //0100
    DOWN = 8, //1000

    VISITED = 128 //1000 0000
}

8 references
public struct Position
{
    public int X;
    public int Y;
}

6 references
public struct Neighbour
{
    public Position Position;
    public Wall SharedWall;
}

1 reference
public static class MazeGenerator
{
    1 reference
    private static Wall getOppositeWall(Wall wall)
    {
        switch (wall)
        {
            case Wall.RIGHT: return Wall.LEFT;
            case Wall.LEFT: return Wall.RIGHT;
            case Wall.UP: return Wall.DOWN;
            case Wall.DOWN: return Wall.UP;
            default: return Wall.LEFT;
        }
    }
}

```

Kodas 9 MazeGenerator 1

```

1 reference
private static Wall[,] generateMaze(Wall[,] maze, int width, int height, int seed)
{
    var rng = new System.Random(seed);
    var positionStack = new Stack<Position>();
    var position = new Position { X = rng.Next(0, width), Y = rng.Next(0, height) };

    maze[position.X, position.Y] |= Wall.VISITED; // 1000 1111
    positionStack.Push(position);

    while(positionStack.Count > 0)
    {
        var current = positionStack.Pop();
        var neighbours = getUnvisitedNeighbours(current, maze, width, height);
        if(neighbours.Count > 0)
        {
            positionStack.Push(current);

            var randIndex = rng.Next(0, neighbours.Count);
            var randomNeighbour = neighbours[randIndex];

            var nPosition = randomNeighbour.Position;
            maze[current.X, current.Y] &= ~randomNeighbour.SharedWall;
            maze[nPosition.X, nPosition.Y] &= ~getOppositeWall(randomNeighbour.SharedWall);

            maze[nPosition.X, nPosition.Y] |= Wall.VISITED;

            positionStack.Push(nPosition);
        }
    }

    return maze;
}

```

Kodas 10 MazeGenerator 2

```

1 reference
private static List<Neighbour> getUnvisitedNeighbours(Position p, Wall[,] maze, int width, int height)
{
    var list = new List<Neighbour>();

    if (p.X > 0) //left
    {
        if (!maze[p.X - 1, p.Y].HasFlag(Wall.VISITED))
        {
            list.Add(new Neighbour
            {
                Position = new Position
                {
                    X = p.X - 1,
                    Y = p.Y
                },
                SharedWall = Wall.LEFT
            });
        }
    }

    if (p.Y > 0) //DOWN
    {
        if (!maze[p.X, p.Y - 1].HasFlag(Wall.VISITED))
        {
            list.Add(new Neighbour
            {
                Position = new Position
                {
                    X = p.X,
                    Y = p.Y - 1
                },
                SharedWall = Wall.DOWN
            }); ;
        }
    }

    if (p.Y < height - 1) //UP
    {
        if (!maze[p.X, p.Y + 1].HasFlag(Wall.VISITED))
        {
            list.Add(new Neighbour
            {
                Position = new Position
                {
                    X = p.X,
                    Y = p.Y + 1
                },
                SharedWall = Wall.UP
            }); ;
        }
    }

    if (p.X < width - 1) //RIGHT
    {
        if (!maze[p.X + 1, p.Y].HasFlag(Wall.VISITED))
        {
            list.Add(new Neighbour
            {
                Position = new Position
                {
                    X = p.X + 1,
                    Y = p.Y
                },
                SharedWall = Wall.RIGHT
            }); ;
        }
    }

    return list;
}

```

Kodas 11 MazeGenerator 3



```

1 reference
public static Wall[,] Generate(int width, int height, int seed)
{
    Wall[,] maze = new Wall[width, height];
    Wall initial = Wall.RIGHT | Wall.LEFT | Wall.UP | Wall.DOWN;
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < height; j++)
        {
            maze[i, j] = initial; //all values 1111
        }
    }

    return generateMaze(maze,width,height,seed);
}

```

Kodas 12 MazeGenerator 4

```

@ Unity Script | 1 reference
public class MazeRenderer : MonoBehaviour
{
    //VARS

    [SerializeField][Min(0f)] private int Width = 10;
    [SerializeField][Min(0f)] private int Height = 10;
    [SerializeField][Min(0f)] private int Seed = 0;
    [SerializeField][Min(0f)] private int NumOfSaws;
    [SerializeField][Min(0f)] private int NumOfPoints;
    [SerializeField] [Min(0f)] private int NumOfHearts;

    private float size = 1f;

    [SerializeField] private Transform wallPrefab = null;
    [SerializeField] private Transform floorPrefab = null;
    [SerializeField] private Transform icyFloorPrefab = null;
    [SerializeField] private Transform honeyPrefab = null;
    [SerializeField] private Transform sawPrefab = null;
    [SerializeField] private Transform playerPrefab = null;
    [SerializeField] private Transform pointPrefab = null;
    [SerializeField] private Transform heartPrefab = null;

    [SerializeField] private Transform startObject = null;
    [SerializeField] private Transform endObject = null;

    @ Unity Message | - references
    void Start()
    {
        var maze = MazeGenerator.Generate(Width,Height,Seed);
        Draw(maze);
    }

    1 reference
    void GenerateSaws()
    {
        var r = new System.Random();
        for (int i = 0; i < NumOfSaws; i++)
        {
            int rX = r.Next((-Width - 1) / 2, Height / 2);
            int rY = r.Next((-Height - 1) / 2, Height / 2);
            var sawblade = Instantiate(sawPrefab, transform);
            var sawController = GetComponentInChildren<SawMovement>();
            if( i % 2==0 ) sawController.toggleXAxis();
            sawblade.position = new Vector3(rX, 0.5f, rY);
        }
    }

    void GeneratePoints()
    {
        var r = new System.Random();
        for (int i = 0; i < NumOfPoints; i++)
        {
            int rX = r.Next((-Width - 1) / 2, Height / 2);
            int rY = r.Next((-Height - 1) / 2, Height / 2);
            var point = Instantiate(pointPrefab, transform);
            point.position = new Vector3(rX, 0.25f, rY);
        }
    }
}

```

Kodas 13 MazeRenderer 1

```

1 reference
void GenerateHearts()
{
    var r = new System.Random(23);
    for (int i = 0; i < NumOfHearts; i++)
    {
        int rX = r.Next((-Width - 1) / 2, Height / 2);
        int rY = r.Next((-Height - 1) / 2, Height / 2);
        var point = Instantiate(heartPrefab, transform);
        point.position = new Vector3(rX, 0f, rY);
    }
}

1 reference
void generateStartEnd()
{
    var startPlatform = Instantiate(startObject, transform);
    startPlatform.position = new Vector3(0, 0, 0);

    var endPlatform = Instantiate(endObject, transform);
    var rnd = Random.Range(1, 4);
    switch (rnd)
    {
        case 1:
            endPlatform.position = new Vector3(-Width / 2, 0, (Height - 1) / 2);
            break;
        case 2:
            endPlatform.position = new Vector3((Width - 1) / 2, 0, -Height / 2);
            break;
        case 3:
            endPlatform.position = new Vector3((Width - 1) / 2, 0, (Height - 1) / 2);
            break;
        case 4:
            endPlatform.position = new Vector3(-Width / 2, 0, -Height / 2);
            break;
    }
}

2 references
public void generatePlayer()
{
    playerPrefab.position = new Vector3(0, 0, 0);
}

1 reference
void placeIceFloor(Vector3 position)
{
    var icyFloor = Instantiate(icyFloorPrefab, transform);
    icyFloor.localScale = new Vector3(icyFloor.localScale.x, icyFloor.localScale.y, icyFloor.localScale.z);
    icyFloor.position = position + new Vector3(0, -1, 0);
}

1 reference
void placeHoney(Vector3 position)
{
    var honeyFloor = Instantiate(honeyPrefab, transform);
    honeyFloor.localScale = new Vector3(honeyFloor.localScale.x, honeyFloor.localScale.y, honeyFloor.localScale.z);
    honeyFloor.position = position + new Vector3(0, -1, 0);
}

1 reference
void placeFloor(Vector3 position)
{
    var floor = Instantiate(floorPrefab, transform);
    floor.localScale = new Vector3(floor.localScale.x, floor.localScale.y, floor.localScale.z);
    floor.position = position + new Vector3(0, -1, 0);
}

```

## Kodas 14 MazeRender 2

```

private void Draw(Wall[,] maze)
{
    generatePlayer();
    generateStartEnd();
    GenerateSaws();
    GeneratePoints();
    GenerateHearts();

    for (int i = 0; i < Width; i++)
    {
        for (int j = 0; j < Height; j++)
        {
            var cell = maze[i, j];
            var position = new Vector3(-Width / 2 + 1, 1f, -Height / 2 + j); //OFFSET of the middle of the cell, so there's a path
            var rnd = Random.Range(1, 11);

            if(rnd > 6 && rnd <=9)
            {
                placeIceFloor(position);
            }
            else if(rnd > 9)
            {
                placeMoney(position);
            }
            else
            {
                placeFloor(position);
            }

            if (cell.HasFlag(Wall.UP))
            {
                var topWall = Instantiate(wallPrefab, transform) as Transform;
                topWall.position = position + new Vector3(0, 0, size/2);
                topWall.localScale = new Vector3(size, topWall.localScale.y, topWall.localScale.z);
            }

            if (cell.HasFlag(Wall.LEFT))
            {
                var leftWall = Instantiate(wallPrefab, transform) as Transform;
                leftWall.position = position + new Vector3(-size / 2, 0, 0);
                leftWall.localScale = new Vector3(size, leftWall.localScale.y, leftWall.localScale.z);
                leftWall.eulerAngles = new Vector3(0, 90, 0);
            }

            if (i == Width - 1)
            {
                if (cell.HasFlag(Wall.RIGHT))
                {
                    var rightWall = Instantiate(wallPrefab, transform) as Transform;
                    rightWall.position = position + new Vector3(size / 2, 0, 0);
                    rightWall.localScale = new Vector3(size, rightWall.localScale.y, rightWall.localScale.z);
                    rightWall.eulerAngles = new Vector3(0, 90, 0);
                }
            }

            if (j == 0)
            {
                if (cell.HasFlag(Wall.DOWN))
                {
                    var bottomWall = Instantiate(wallPrefab, transform) as Transform;
                    bottomWall.position = position + new Vector3(0, 0, -size / 2);
                    bottomWall.localScale = new Vector3(size, bottomWall.localScale.y, bottomWall.localScale.z);
                }
            }
        }
    }
}

```

Kodas 15 MazeRenderer 3

```

[RequireComponent(typeof(CharacterController))]
// Unity Script | 6 references
public class MovementController : MonoBehaviour
{
    // VARS
    [Min(0f)][SerializeField] private float movementSpeed;
    [Min(0f)][SerializeField] private float walkingSpeed;
    [Min(0f)][SerializeField] private float runningSpeed;
    [Min(0f)][SerializeField] private float jumpHeight;
    [Range(0,10)][SerializeField] private float stamina=10;
    [Range(0, 10)] [SerializeField] private float maxStamina = 10;
    [Min(0f)][SerializeField] private float cooldown=2;
    [Min(0f)][SerializeField] private float slideSpeed = 0.35f;
    [Min(0f)][SerializeField] private float honeySpeed = 2f;

    [SerializeField] private GameObject snowParticles;

    [SerializeField] private AudioClip jumpingSound;

    AudioSource SoundSource;

    private Vector3 moveDirection;
    private Vector3 velocity;
    private Vector3 lastMoveDirection = Vector3.zero;
    private float StaminaRegenTimer = 0.0f;
    private bool icy = false;
    private bool honey = false;
    private bool canJump = true;

    private const float staminaDecrease = 2.0f;
    private const float staminaIncrease = 5.0f;

    private bool isParticleRunning = false;

    [SerializeField] private bool isGrounded;
    [Min(0f)][SerializeField] private float groundCheckDistance;
    [SerializeField] private LayerMask groundMask;
    [SerializeField] private float gravity;

    // REFS
    private CharacterController controller;
    private Animator anim;

    // Unity Message | 0 references
    void Awake()
    {
        controller = GetComponent<CharacterController>();
        anim = GetComponentInChildren<Animator>();
        SoundSource = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    // Unity Message | 0 references
    void Update()
    {
        Move();
    }
}

```

Kodas 16 MovementController 1

```

1 reference
private void Move()
{
    isGrounded = Physics.CheckSphere(transform.position, groundCheckDistance, groundMask);

    if(isGrounded && velocity.y < 0)
    {
        velocity.y = -2f;
    }

    float moveZ = Input.GetAxis("Vertical");
    float moveX = Input.GetAxis("Horizontal");
    float inputMagnitude = Mathf.Min(new Vector3(moveX, 0, moveZ).sqrMagnitude, 1f);

    moveDirection = new Vector3(moveX, 0, moveZ);
    moveDirection = transform.TransformDirection(moveDirection);
    if(inputMagnitude > 0.225f)
    {
        lastMoveDirection = moveDirection;
    }
    if (isGrounded)
    {
        if (icy)
        {
            moveDirection = lastMoveDirection * slideSpeed;
            if (!isParticleRunning)
            {
                Instantiate(snowParticles, transform.localPosition, Quaternion.identity);
                isParticleRunning = true;
            }
            canJump = true;
        }
        else if(honey)
        {
            DestroyAll("Snow");
            isParticleRunning = false;
            moveDirection *= honeySpeed;
            canJump = false;
        }
        else
        {
            DestroyAll("Snow");
            isParticleRunning = false;
            canJump = true;
            if(moveDirection != Vector3.zero && !Input.GetKey(KeyCode.LeftShift))
            {
                Walk();
            }
            else if(moveDirection != Vector3.zero && Input.GetKey(KeyCode.LeftShift)){
                Run();
            }
            else if(moveDirection == Vector3.zero)
            {
                Idle();
            }
            moveDirection *= movementSpeed;
        }
    }

    if (Input.GetKeyDown(KeyCode.Space))
    {
        Jump(canJump);
    }
}

```

Kodas 17 MovementController 2

```

        if (Input.GetKeyDown(KeyCode.X))
        {
            jumpHeight = 10;
        }
        if (Input.GetKeyDown(KeyCode.C))
        {
            jumpHeight = 1.25f;
        }
    }
    controller.Move(moveDirection * Time.deltaTime); //movement
    velocity.y += gravity * Time.deltaTime;
    controller.Move(velocity * Time.deltaTime); //gravity
}

2 references
private void Walk()
{
    movementSpeed = walkingSpeed;
    anim.SetFloat("Speed", 0.25f, 0.1f, Time.deltaTime);
    RegenerateStamina();
}

1 reference
private void Run()
{
    if (stamina > 0)
    {
        stamina = Mathf.Clamp(stamina - (staminaDecrease * Time.deltaTime), 0.0f, maxStamina);
        movementSpeed = runningSpeed;
        anim.SetFloat("Speed", 0.5f, 0.1f, Time.deltaTime);
        StaminaRegenTimer = 0.0f;
    }
    else
    {
        Walk();
    }
}

1 reference
private void Idle()
{
    anim.SetFloat("Speed", 0, 0.1f, Time.deltaTime);
    RegenerateStamina();
}

1 reference
private void Jump(bool canJump)
{
    if (canJump)
    {
        velocity.y = Mathf.Sqrt(jumpHeight * -2 * gravity);
        movementSpeed = walkingSpeed;
        SoundSource.PlayOneShot(jumpingSound);
        anim.SetFloat("Speed", 0.75f);
    }
}

```

Kodas 18 MovementContoller 3

```

private void RegenerateStamina()
{
    if (StaminaRegenTimer >= cooldown)
    {
        stamina = Mathf.Clamp(stamina + (staminaIncrease * Time.deltaTime), 0.0f, maxStamina);
    }
    else
    {
        StaminaRegenTimer += Time.deltaTime;
    }
}

public float getMaxStamina()
{
    return maxStamina;
}

public float getStamina()
{
    return stamina;
}

public float getCooldown()
{
    return StaminaRegenTimer - cooldown;
}

void OnControllerColliderHit(ControllerColliderHit hit)
{
    icy = hit.collider.CompareTag("Ice");
    honey = hit.collider.CompareTag("Honey");
}

private void createSnowyParticles(Vector3 position)
{
    var iceParticles = Instantiate(snowParticles, position, Quaternion.identity);
    iceParticles.SetActive(true);
}

void DestroyAll(string tag)
{
    GameObject[] items = GameObject.FindGameObjectsWithTag(tag);
    for (int i = 0; i < items.Length; i++)
    {
        Destroy(items[i]);
    }
}

```

Kodas 19 MovementContoller 4

```

public class MusicSelector : MonoBehaviour
{
    public AudioSource Track1;
    public AudioSource Track2;
    public AudioSource Track3;

    public int TrackSelector;
    public int TrackHistory;

    void Start()
    {
        TrackSelector = Random.Range(0, 3);
        if(TrackSelector == 0)
        {
            Track1.Play();
            TrackHistory = 0;
        }
        else if(TrackSelector == 1)
        {
            Track2.Play();
            TrackHistory = 1;
        }
        else if(TrackSelector == 2)
        {
            Track3.Play();
            TrackHistory = 2;
        }
    }

    void Update()
    {
        if (Track1.isPlaying == false && Track2.isPlaying == false && Track3.isPlaying == false)
        {
            TrackSelector = Random.Range(0, 3);
            if (TrackSelector == 0 && TrackHistory != 0)
            {
                Track1.Play();
            }
            else if (TrackSelector == 1 && TrackHistory != 1)
            {
                Track2.Play();
            }
            else if(TrackSelector == 2 && TrackHistory != 2)
            {
                Track3.Play();
            }
        }
    }
}

```

Kodas 20 MusicSelector



```

[RequireComponent(typeof(MovementController))]

public class Player : MonoBehaviour
{
    [SerializeField, Min(0)] private int points = 0;
    [SerializeField, Min(0)] private int healthPoints = 1;
    [SerializeField, Min(1)] private int maxHealthPoints = 3;
    [SerializeField] private TMP_Text pointsText;
    [SerializeField] private TMP_Text healthPointsText;
    [SerializeField] private Slider staminaBar;

    [SerializeField] UnityEvent onStopGame;
    [SerializeField] UnityEvent onWinGame;

    private MovementController movementController;

    private GameplayCanvasController gameplayController;

    AudioSource playerAudioSource;

    private void Awake()
    {
        movementController = GetComponent<MovementController>();
    }

    private void Start()
    {
        playerAudioSource = GetComponent<AudioSource>();
        pointsText.gameObject.SetActive(true);
        staminaBar.gameObject.SetActive(true);
        healthPointsText.gameObject.SetActive(true);
        UpdatePointsText();
        UpdateHealthPointsText();
    }

    public void AddPoints(int value)
    {
        points += value;
        UpdatePointsText();
    }

    public void AddHealthPoint(int value)
    {
        if(healthPoints < maxHealthPoints)
        {
            healthPoints += value;
            UpdateHealthPointsText();
        }
    }
}

```

Kodas 21 Player 1

```

}
1 reference
public void removeHealthPoint()
{
    if(healthPoints > 0)
    {
        healthPoints -= 1;
        UpdateHealthPointsText();
    }
    else
    {
        Lose();
    }
}

private void UpdatePointsText()
{
    pointsText.text = $"Points: {points}";
}

private void UpdateHealthPointsText()
{
    healthPointsText.text = $"Lives: {healthPoints}";
}

public void Win()
{
    onWinGame.Invoke();
}

public void Lose()
{
    onStopGame.Invoke();
}

private void PlayerFootstepSound()
{
    playerAudioSource.Play();
}

public int getPoints()
{
    return points;
}

public int getHealthPoints()
{
    return healthPoints;
}

```

Kodas 22 Player 2

```

public class SawMovement : MonoBehaviour
{
    [SerializeField] private float speed = 1.0f;
    [SerializeField] private float blockMovement = 3;

    [SerializeField] private bool xAxis = false;
    private Vector3 pos;
    private Vector3 posX;
    private Vector3 posY;

    0 references
    void Start()
    {
        pos = transform.position;
        posX = new Vector3(pos.x + blockMovement, pos.y, pos.z);
        posY = new Vector3(pos.x, pos.y, pos.z + blockMovement);
    }

    0 references
    void Update()
    {
        if (xAxis)
        {
            xAxisMovement();
        }
        else
        {
            yAxisMovement();
        }
    }

    1 reference
    void xAxisMovement()
    {
        transform.position = Vector3.Lerp(pos, posX, Mathf.PingPong(Time.time * speed, 1.0f));
    }

    1 reference
    void yAxisMovement()
    {
        transform.position = Vector3.Lerp(pos, posY, Mathf.PingPong(Time.time * speed, 1.0f));
    }

    1 reference
    public void toggleXAxis()
    {
        xAxis = true;
    }
}

```

Kodas 23 SawMovement

```

public static class Scenes
{
    public static void RestartScene()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public static void LoadNextScene()
    {
        LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

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

    public static void ExitGame()
    {
        #if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false;
        #else
            Application.Quit();
        #endif
    }

    private static void LoadScene(int buildIndex)
    {
        SceneManager.LoadScene(buildIndex);
    }
}

```

Kodas 24 Scenes

```

@ Unity Script | 0 references
public class Settings : MonoBehaviour
{
    public AudioManager musicAudioMixer;
    public AudioManager effectsAudioMixer;

    public Slider musicSlider;
    public Slider effectSlider;

    0 references
    public void SetVolumeMusic(float volume)
    {
        musicSlider.value = volume;
        musicAudioMixer.SetFloat("MusicVolume", volume);
    }

    0 references
    public void SetVolumeEffects(float volume)
    {
        effectSlider.value = volume;
        effectsAudioMixer.SetFloat("EffectVolume", volume);
    }

    0 references
    public void SetFullscreen(bool isFullscreen)
    {
        Screen.fullScreen = isFullscreen;
    }
}

```

Kodas 25 Settings

```

[RequireComponent(typeof(MovementController))]

public class StaminaBarController : MonoBehaviour
{
    [SerializeField] private Slider staminaBar;
    [SerializeField] private TMP_Text staminaText;

    private float stamina;
    private float maxStamina;

    //REFERENCES
    private MovementController movController;

    void Awake()
    {
        movController = GetComponent<MovementController>();
    }

    // Update is called once per frame

    void Start()
    {
        staminaBar.maxValue = movController.getMaxStamina();
        staminaBar.value = movController.getStamina();
    }

    private void Update()
    {
        staminaBar.value = movController.getStamina();
        if (staminaBar.value == 0)
        {
            staminaText.text = $"{System.Math.Round(movController.getCooldown() *-1)}s. cooldown";
        }
        else
        {
            staminaText.text = "Stamina";
        }
    }
}

```

Kodas 26 StaminaBarController

```

public class TimerController : MonoBehaviour
{
    public static TimerController instance;

    [SerializeField] private TMP_Text timer;

    private TimeSpan timePlaying;
    private bool isRunning;

    private float elapsedTime;

    private void Awake()
    {
        instance = this;
        timer.gameObject.SetActive(true);
        timer.text = "Time: 00:00:00";
        isRunning = false;
    }

    public void BeginTimer()
    {
        isRunning = true;
        elapsedTime = 0f;

        StartCoroutine(UpdateTimer());
    }

    public void EndTimer()
    {
        isRunning = false;
    }

    private IEnumerator UpdateTimer()
    {
        while (isRunning)
        {
            elapsedTime += Time.deltaTime;
            timePlaying = TimeSpan.FromSeconds(elapsedTime);
            string timePlayingStr = "Time: " + timePlaying.ToString("mm':'ss'.'ff");
            timer.text = timePlayingStr;

            yield return null;
        }
    }

    public TimeSpan getTime()
    {
        return timePlaying;
    }

    // Update is called once per frame
}

```

Kodas 27 TimerController

```

[RequireComponent(typeof(Player))]

public class TriggerController : MonoBehaviour
{
    [SerializeField, Min(0)] private int pointValue = 1;
    [SerializeField, Min(0)] private int heartValue = 1;
    [SerializeField] private AudioClip heartSound;
    [SerializeField] private AudioClip pointSound;
    [SerializeField] private AudioClip finishSound;
    [SerializeField] private AudioClip sawSound;

    [SerializeField] private GameObject heartParticlePrefab;
    [SerializeField] private GameObject bloodPrefab;

    AudioSource soundSource;
    private Player player;

    private void Awake()
    {
        player = GetComponent<Player>();
        soundSource = GetComponent<AudioSource>();
    }

    // Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        var otherGameObject = other.gameObject;
        var collected = false;

        if(other.tag == "Point")
        {
            player.AddPoints(pointValue);
            soundSource.PlayOneShot(pointSound);
            collected = true;
        } else if(other.tag == "Finish")
        {
            player.Win();
            soundSource.PlayOneShot(finishSound);
        }
        else if(other.tag == "Heart")
        {
            player.AddHealthPoint(heartValue);
            soundSource.PlayOneShot(heartSound);
            createHeartParticles(otherGameObject.transform.position);
            collected = true;
        }

        if (collected)
        {
            otherGameObject.SetActive(false);
            Destroy(otherGameObject);
        }
    }

    void OnCollisionEnter(Collision other)
    {
        var otherGameObject = other.gameObject;
        if (otherGameObject.tag == "Saw")
        {
            player.RemoveHealthPoint();
            createBloodParticles(other.transform.position);
            soundSource.PlayOneShot(sawSound);
        }
    }

    private void createHeartParticles(Vector3 position)
    {
        Instantiate(heartParticlePrefab, position, Quaternion.identity);
    }

    private void createBloodParticles(Vector3 position)
    {
        Instantiate(bloodPrefab, position, Quaternion.identity);
    }
}

```

Kodas 28 TriggerController