KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B166 Development of Computer Games and Interactive Applications

One Small Favor

Group, Name and Surname:
IFF-9/11 Vidas Buivydas
IFF-9/4 Gytis Burbeckas
IFF-9/11 Matas Jonauskas

Kaunas, 2022

Tables of Contents

Tables of Images

Table of Tables/functions

Work Distribution Table:

Name/Surname	***Description of game development part***
Vidas Buivydas	*Creation of one main character and their abilities, weapons, armors. Rooms of the dungeon. Random room generator algorithm. Creation of enemies. Creation of experience system.*
Gytis Burbeckas	*Creation of one main character and their abilities, weapons, armors. Creation of enemies. Creation of enemies' animations. Creation of shop system. Creation of UI. Creation of collectibles.*
Matas Jonauskas	*Creation of one main character and their abilities, weapons, armors. Creation of enemies. Creation of animations and dialogues at the start and the ending of the game. Creation of drop system.*

Description of Your Game

Description of Your Game.

- 3D or 2D? 2D.
- What type is your game? roguelike.
- Platforms (mobile, PC, or both?) computer.
- Scenario description: the mother asks her son to take the jar from the basement. When the character enters the basement the door slams and the light goes out. Suddenly the light appears in the back of the room and an unknown person appears near the door. He warns about the dangers behind these doors and offers three weapons to choose from (these weapons determine the main character's race). When the main character chooses a weapon he enters the dungeon and nearby monsters start to attack him. After defeating many enemies he finally finds a jar and the final boss appears. When the final boss is defeated, he finally returns home and hands the jar to his mom but sadly he took the bad jar and has to return to the basement to take the right one.
- Game mechanics: there will be 3 different main character races with different weapons and attacks. Also, the main characters can use spells. Every race has its owecific type of weapons and armors which grants them more bonuses. Rooms are generated randomly. There will be different types of rooms like shops or rooms full of monsters. Defeating enemies grants experience to player. After reaching the next character level he gets skill points which he can use to upgrade character attributes. Enemies will drop loot like coins or potions, etc. … At the end of the dungeo,n the player has to defeat the boss in order to reach the next level.

Laboratory work #1

List of tasks

1. Create main characters (movement and animations)
2. Create collectibles
3. Create a room generator
4. Create lighting
5. Create a simple UI
6. Create traps

Solution

Task #1 Create main characters (movement and animations)

Four characters have been created for the game: boy, archer, wizard and knight. After that, the player's controller script was created. The players controller script includes player movement in all directiothe ns, player's weapon following the mouse cursor and the player's animations activation. Assets for characters were taken from the website [1].

Figure 1. Boy

Figure 2. Elf

Figure 3. Wizard

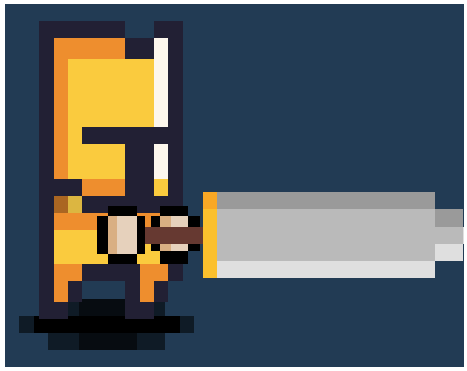

Figure 4. Knight

PlayersContoller script:

void Start() [Table 1] (Matas Jonauskas)
void Update() [Table 2] (Matas Jonauskas)
void PlayerMovement() [Table 3] (Matas Jonauskas)
void PlayerAndGunRotation() [Table 4] (Matas Jonauskas)
Void MovementAnimation() [Table 5] (Matas Jonauskas)

Start() function assigns camera when the game starts.

```
void Start()
{
    theCam = Camera.main;
}
```

Table 1. Function Start()

Update() function calls PlayerMovement() PlayerAndGunRotation(), MovementAnimation() functions each frame.

```
void Update()
{
    //Player movement
    PlayerMovement();

    //Player and gun rotation to mouse cursor
    PlayerAndGunRotation();

    //Movement animation
    MovementAnimation();
}
```

Table 2. Function Update()

PlayerMovement() function gets input axis data from player and assigns to game character's movement x and y axis. Normalize() function is needed to normalize player speed when moving diagonally. TheRB.velocity sets the player's speed.

```
void PlayerMovement(){

    moveInput.x = Input.GetAxisRaw("Horizontal");
    moveInput.y = Input.GetAxisRaw("Vertical");

    moveInput.Normalize();

    theRB.velocity = moveInput * moveSpeed;
}
```

Table 3. Function PlayerMovement()

PlayerAndGunRotation() function, gets mouse and player's positions vectors. Then according to these positions player is rotated either to the left or right. Then the angle is calculated for gun rotation according to the mouse cursor position.

```
void PlayerAndGunRotation(){

    // mouse position
    Vector3  mousePos = Input.mousePosition;
    // player position
    Vector3 screenPoint = theCam.WorldToScreenPoint(transform.localPosition);

     //rotate player to mouse pointer position
    if(mousePos.x < screenPoint.x){

        transform.localScale = new Vector3(-1f, 1f, 1f);
        gunArms.localScale = new Vector3(-1f, -1f, 1f);
    }else{

        transform.localScale = new Vector3(1f, 1f, 1f);
        gunArms.localScale = new Vector3(1f, 1f, 1f);
    }

    //rotate gun arm/arms
    Vector2 offset = new Vector2(mousePos.x - screenPoint.x, mousePos.y - screenPoint.y);
    float angle = Mathf.Atan2(offset.y, offset.x) * Mathf.Rad2Deg;
    gunArms.rotation = Quaternion.Euler(0, 0, angle);
}
```

Table 4. Function PlayerAndGunRotation

MovementAnimation() function activates or deactivates animations acording to different situations.

```
void MovementAnimation(){

    if(moveInput ≠ Vector2.zero && moveInput.x > 0 || moveInput.x < 0){
        anim.SetBool("IsMovingUpDOWN", false);
        anim.SetBool("IsMovingSides", true);
    }
    else if(moveInput ≠ Vector2.zero && moveInput.y > 0 || moveInput.y < 0 && moveInput.x == 0)
    {
        anim.SetBool("IsMovingSides", false);
        anim.SetBool("IsMovingUpDOWN", true);
    }

    else{
        anim.SetBool("IsMovingUpDOWN", false);
        anim.SetBool("IsMovingSides", false);
    }
}
```

Table 5. Function Movement Animation()

Characters animations creation:

Boy and Knight animations were created by Matas Jonauskas
Archer animations were created by Vidas Buivydas
Wizard animations were created by Gytis Burbeckas

All players have similar three animations: idle, running to sides and running upwards/downwards.

Idle animation:

All characters' idle animation is basic arm/weapon movement up and down.

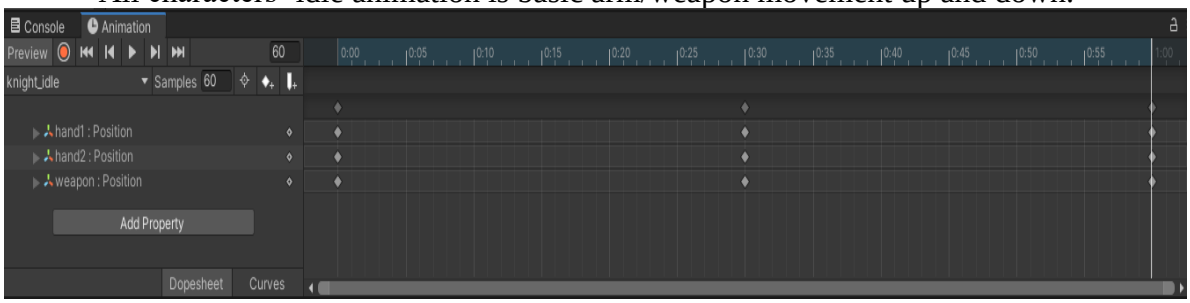

Figure 5. Arm movement Animation window

Running to sides animation:

All characters running animation consist of 4 different sprites:

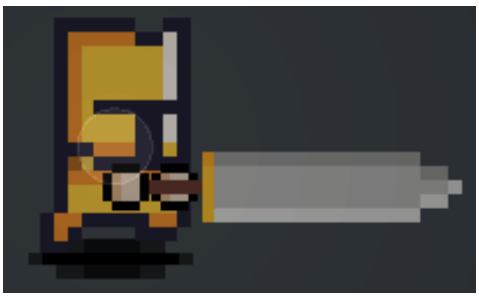

Figure 6. Run to sides sprite 1

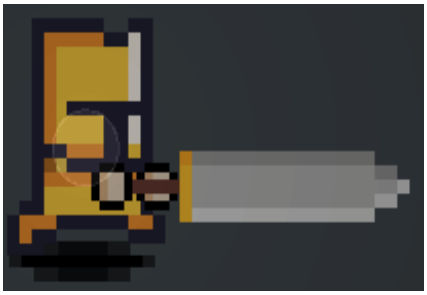

Figure 7. Run to sides sprite 2

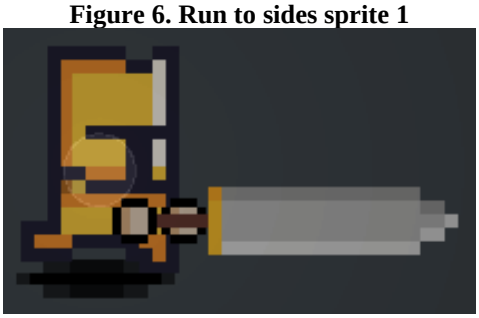

Figure 8. Run to sides sprite 3

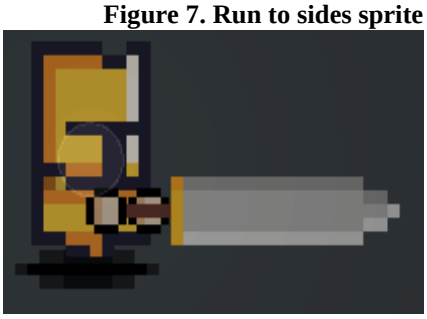

Figure. 9 Run to sides sprite 4

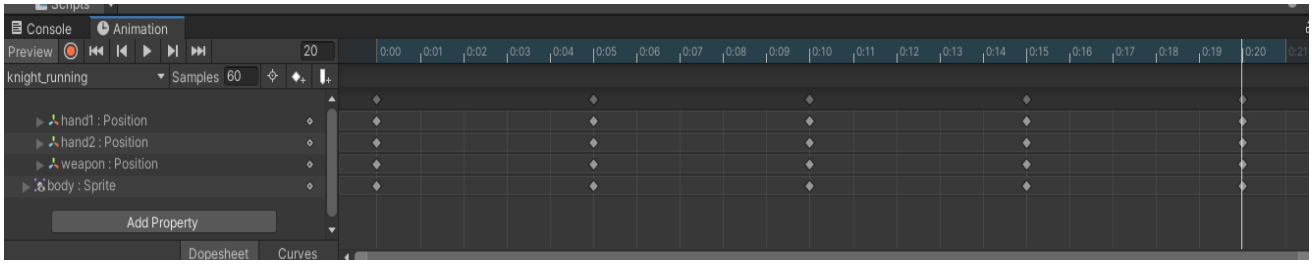

Figure 10 Character running to sides Animation window

Running up/down animation:

This animation consists of 3 different positions:

Figure 11 Running up/down sprite 1

Figure 12 Running up/down sprite 2

Figure 13 Running up/down sprite 3

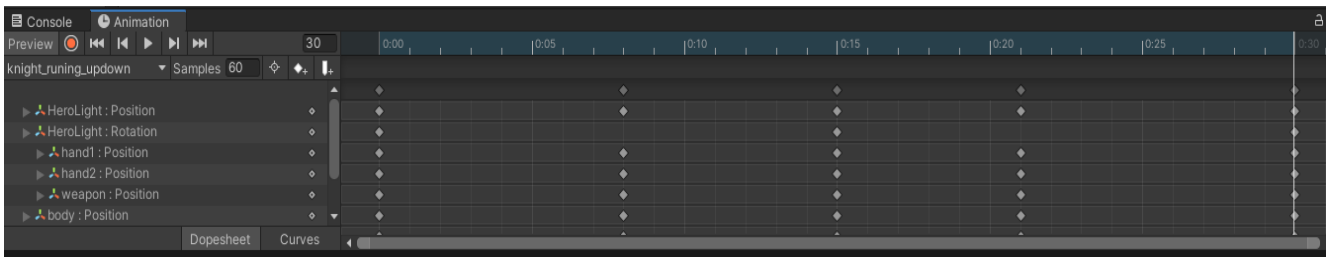

Figure 14. Character running up/down Animation window

Animator:

All characters have a similar animator configuration:

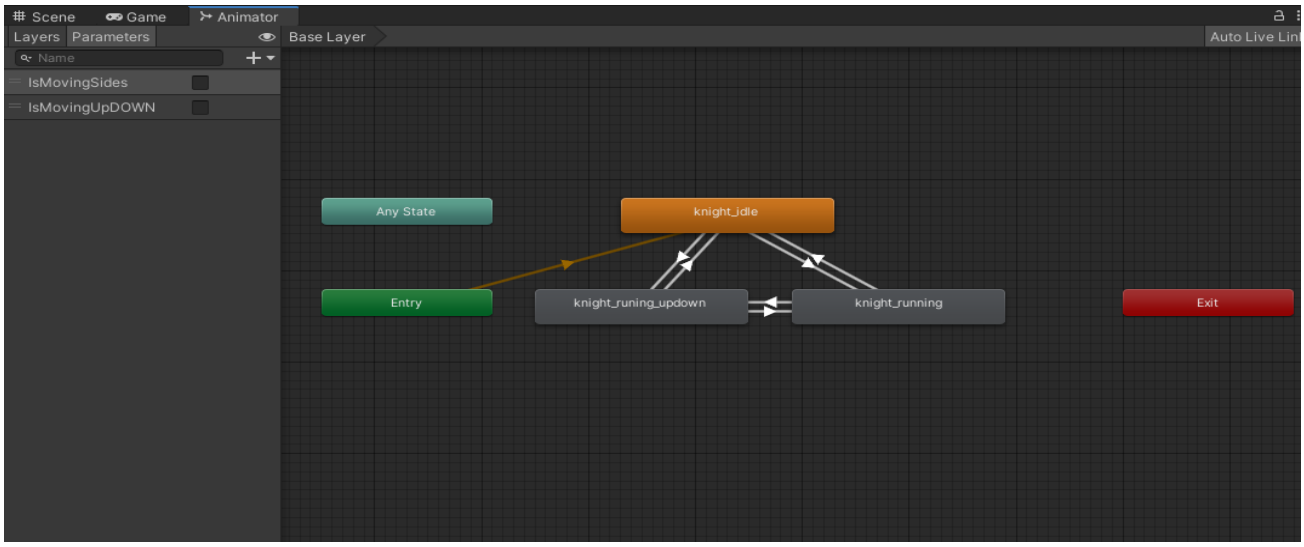

Figure 15. Knight Animator window

Task #2 Create collectibles

Firstly, health, mana, coin and trap objects were created. After scripts that control collectibles were created. These scripts destroy collectibles and add mana, health, coins or do damage when needed. Also, they display player health, mana and coins. Assets for collectibles were taken from the website [2]. Also floating animation was created for health and mana potion. (Gytis Burbeckas)

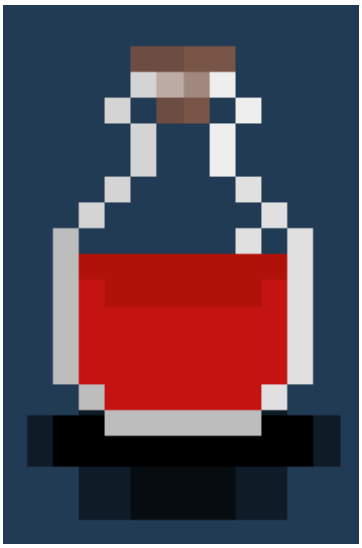

Figure 16. HealthPotion

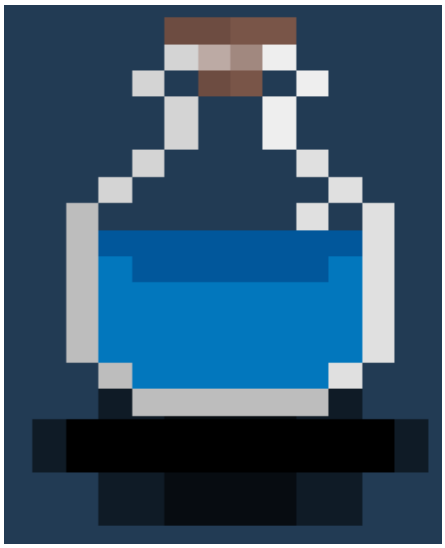

Figure 17. ManaPotion

Figure 18. Coin

void Start() [Table] (Gytis Burbeckas)
void UpdateLivesText() [Table] (Gytis Burbeckas)
void UpdateManaText() [Table] (Gytis Burbeckas)
void UpdateGoldText() [Table] (Gytis Burbeckas)
void TakeDamage(int amount) [Table] (Gytis Burbeckas)
void StopGame() [Table] (Gytis Burbeckas)
void AddHealth(int value) [Table] (Gytis Burbeckas)
void AddMana(int value) [Table] (Gytis Burbeckas)
void AddGold(int value) [Table] (Gytis Burbeckas)
void OnTriggerEnter2D(Collider2D object) [Table] (Gytis Burbeckas)
bool isHealthPotion(GameObject object) [Table] (Gytis Burbeckas)
bool isManaPotion(GameObject object) [Table] (Gytis Burbeckas)
bool isGold1(GameObject object) [Table] (Gytis Burbeckas)

This function sets the health, mana and gold of the player to default values when the game starts.

```
public void Start()
{
    UpdateLivesText();
    UpdateManaText();
    UpdateGoldText();
}
```

Table 6. Function Start()

This function updates the health value that the player can see in a game.

```
private void UpdateLivesText()
{
    healthText.text = $"Health: {health}";
}
```

Table 7. Function UpdateLivesText()

This function updates the mana value that the player can see in a game.

```
private void UpdateManaText()
{
    manaText.text = $"Mana: {mana}";
}
```

Table 8. Function UpdateManaText()

This function updates the gold amount that the player can see in a game.

```
private void UpdateGoldText()
{
    goldText.text = $"Gold: {gold}";
}
```

Table 9. Function UpdateGoldText()

This function subtracts the health of the player when needed and updates. Also, it stops the game when the player's health reaches 0.

```
public void TakeDamage(int amount)
{
    if (isVulnerabe == true)
    {
        if (health - amount <= 0 || health <= 0)
        {
            health = 0;
            UpdateLivesText();

            StopGame();
        }
        else
        {
            health -= amount;

            UpdateLivesText();
            StartCoroutine(StartInvurnebility());


        }
    }
}
```

Table 10. Function TakeDamage(int amount)

This function stops the game, hides the player's mana, health, gold and loads the "GameOver" scene.

```
private void StopGame()
{
    playerController.enabled = false;
    healthText.gameObject.SetActive(false);
    manaText.gameObject.SetActive(false);
    goldText.gameObject.SetActive(false);
    SceneManager.LoadScene("GameOver");

}
```

Table 11. Function StopGame()

This function adds health to the player. It adds health in a way that the player can't have more health than 100.

```
public void AddHealth(int value)
{
    if(health + value >= 100)
    {
        health = 100;
        UpdateLivesText();
    }
    else
    {
        health += value;
        UpdateLivesText();
    }

}
```

Table 12. Function AddHealth(int value)

This function adds health to the player and updates the text that displays health information. It adds mana in a way that the player can't have more than 100.

```
public void AddMana(int value)
{
    if(mana + value >= 100)
    {
        mana = 100;
        UpdateManaText();
    }
    else
    {
        mana += value;
        UpdateManaText();
    }
}
```

Table 13. Function AddMana(intv value)

This function adds gold and updates the text that displays gold information.

```
public void AddGold(int value)
{
    gold += value;
    UpdateGoldText();
}
```

Table 14. Function AddGold(int value)

This function detects what kind of object the player encountered and adds health, mana or gold to the player. After that, the object that the player hit is destroyed.

```
private void OnTriggerEnter2D(Collider2D other)
{
    var otherGameObject = other.gameObject;
    var collected = false;

    if (isHealthPotion(otherGameObject))
    {
        player.AddHealth(healthValue);
        collected = true;
    }
    else if (isManaPotion(otherGameObject))
    {
        player.AddMana(manaValue);
        collected = true;
    }
    else if(isGold1(otherGameObject))
    {
       player.AddGold(gold1Value);
        collected = true;
    }

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

    }
}
```

Table 15. Function OnTriggerEnter2D(Collider other)

This function is used to check if the object that the player encountered is a health potion.

```
private bool isHealthPotion(GameObject obj)
{
    return obj.CompareTag(healthPotionTag);
}
```

Table 16 Function isHealthPotion(GameObject obj)

This function is used to check if the object that the player encountered is a mana potion.

```
private bool isManaPotion(GameObject obj)
{
    return obj.CompareTag(manaPotionTag);
}
```

Table 17. Function isManaPotion(GameObject obj)

This function is used to check if the object that the player encountered is gold.

```
private bool isGold1(GameObject obj)
{
    return obj.CompareTag(gold1Tag);
}
```

Table 18. Function isGold1(GameObject obj)

Task #3 Create a room generator

Firstly, room templates with spawn points (for other rooms to spawn and conjunct) were created [Figure 1]. After that GameObject with room templates for each entrance direction were created [Figure]. Lastly, the script for room spawning was created and 2D box colliders were attached to walls so that players couldn't pass through walls. Assets for wall textures were taken from the website [4]. (Vidas Buivydas).

Room template example:

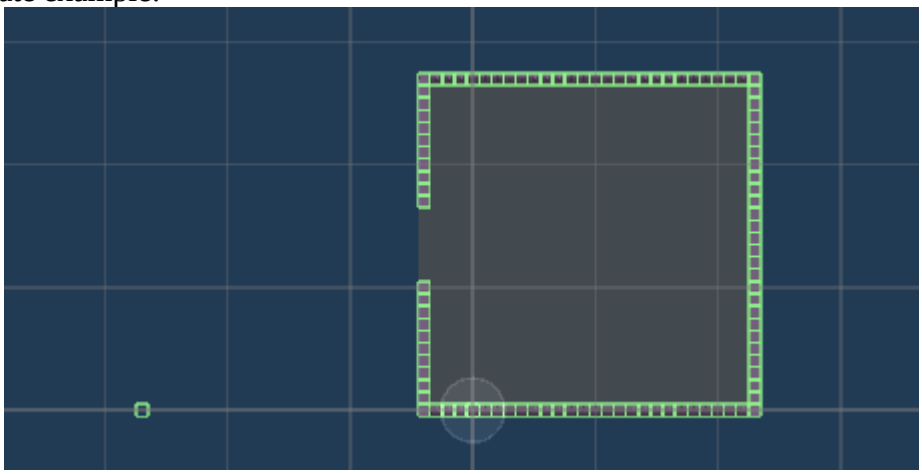

Figure 19. Room template example

List of room templates (Vidas Buivydas):

Figure 20. List of room templates

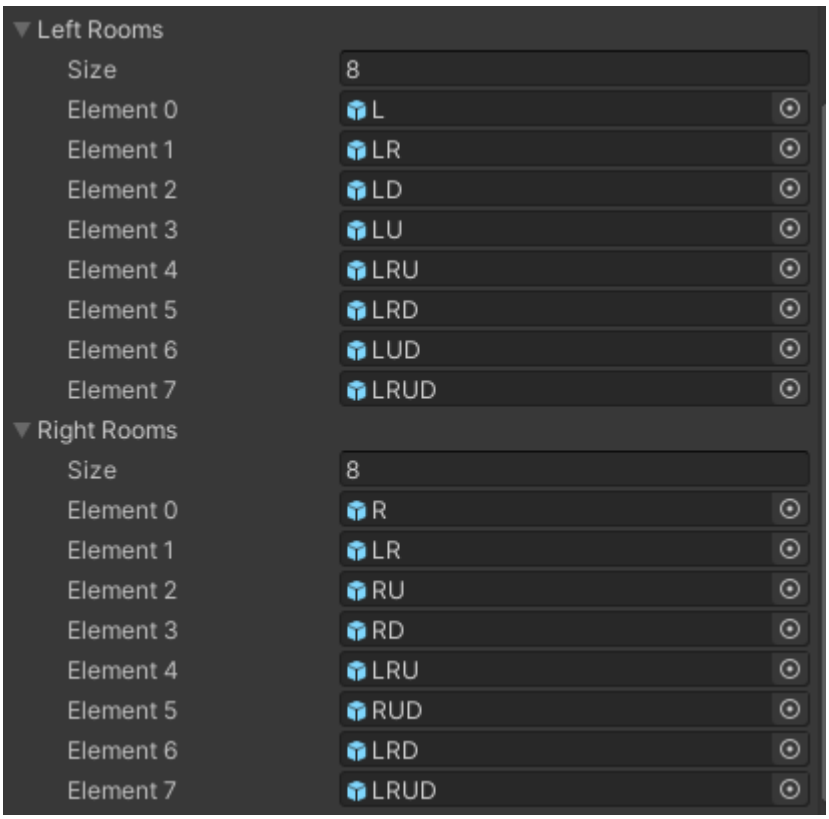

Figure 21. List of room templates

randomly generated rooms:

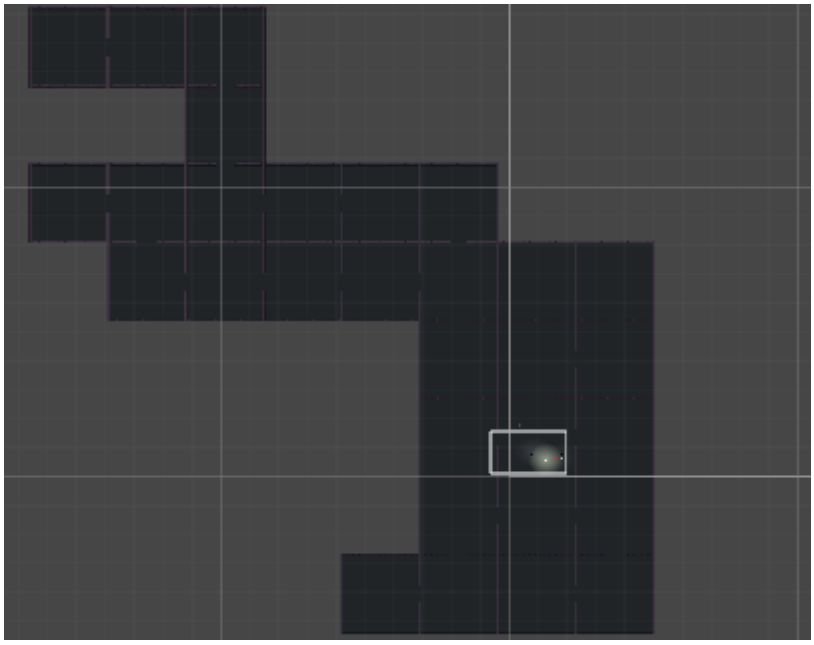

Figure 22. Randomly generated rooms

Functions used in RoomSpawner.cs:

1. void Start() [Table] (Vidas Buivydas)
2. void Spawn() [Table] (Vidas Buivydas)
3. int IndexGenerator(int generated, int length) [Table] (Vidas Buivydas)
4. void OnTriggerEnter2D(Collider2D other) [Table] (Vidas Buivydas)

This function gets all room templates and the count of rooms generated, after that, it calls “Spawn” function which begins spawning other rooms:

```
void Start()
{
    manager = GameObject.FindGameObjectWithTag("Manager").GetComponent<manager>();
    templates = GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();
    Invoke("Spawn", 1f);

}
```

Table 19. Function Start()

This function checks if the room wasn’t spawned on the colliding spawnpoint, then chooses the right room from list of templated based on needed entrance.

```
void Spawn()
{
    if (spawned == false)
    {
        if (openingDirection == 1)
        {
            //need to spawn a room with a bottom door
            rand = IndexGenerator(manager.roomCount, templates.bottomRooms.Length);
            Instantiate(templates.bottomRooms[rand], transform.position, templates.bottomRooms[rand].transform.rotation);
            manager.roomCount++;
        }
        else if (openingDirection == 2)
        {
            //need to spawn a room with a top door
            rand = IndexGenerator(manager.roomCount, templates.topRooms.Length);
            Instantiate(templates.topRooms[rand], transform.position, templates.topRooms[rand].transform.rotation);
            manager.roomCount++;
        }
        else if (openingDirection == 3)
        {
            //need to spawn a room with a left door
            rand = IndexGenerator(manager.roomCount, templates.leftRooms.Length);
            Instantiate(templates.leftRooms[rand], transform.position, templates.leftRooms[rand].transform.rotation);
            manager.roomCount++;
        }
        else if (openingDirection == 4)
        {
            //need to spawn a room with a right door
            rand = IndexGenerator(manager.roomCount, templates.rightRooms.Length);
            Instantiate(templates.rightRooms[rand], transform.position, templates.rightRooms[rand].transform.rotation);
            manager.roomCount++;
        }
        spawned = true;
    }
}
```

Table 20. Function Spawn()

This function is created for balance of rooms. In the beginning, it gives a greater chance of spawning rooms with more entrances and as more room spawns this chance is decreased to a minimum.

```
int IndexGenerator(int generated, int length)
{
    int index = (int)(length / System.Math.Pow(2, generated / 10));
    if (index < 3) index = 3;
    rand = Random.Range(0, index);
    return rand;

}
```

Table 21. Function IndexGenerator(int generated, int length)

This function activates when two spawn points spawn on top of each other leaving only one of them.

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("RoomPoint"))
    {
        spawned = true;
    }
    if (other.CompareTag("SpawnPoint"))
    {
        if (original == true)
        {
            original = false;
        }
        else
        {
            Destroy(gameObject);
            other.gameObject.GetComponent<RoomSpawmer>().original = true;
        }
    }
}
```

Table 22. Function OnTriggerEnter2D(Collider2D other)

Task #4 Create lighting

Firstly, Universal RP was installed into our project using package manager, project render settings were changed to work with URP. After this setup, we could easily create light sources. In the beginning, global light was created to provide the scene with some light, after that torch witch animation was created [Figure]. Finally, point lights were created and attached to objects (characters, collectibles, torches) in order to make them more visible. Assets for torch animation were taken from the website [3]. (Vidas Buivydas)

torch animation: (Vidas Buivydas)

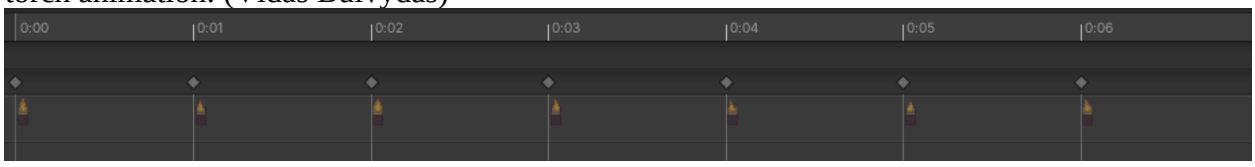

Figure 23. Torch animation

game preview with lighting:

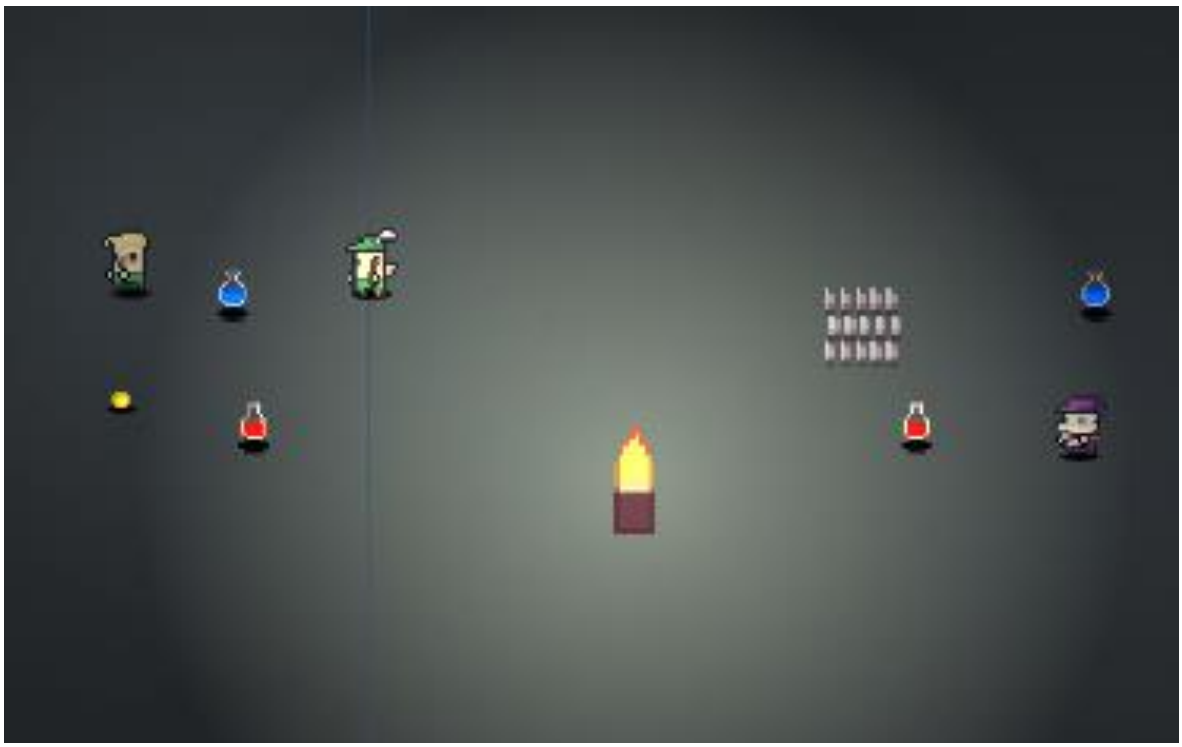

Figure 24. Game preview with lighting

Task #5 Create a simple UI

Two scenes with Canvas were created [Figure **Klaida! Nerastas nuorodos šaltinis.**] [Figure Figure]. After that scripts for changing the scenes were created. According to scripts, On click events were added to buttons. Also, empty objects with event components were added to every scene. (Gytis Burbeckas)

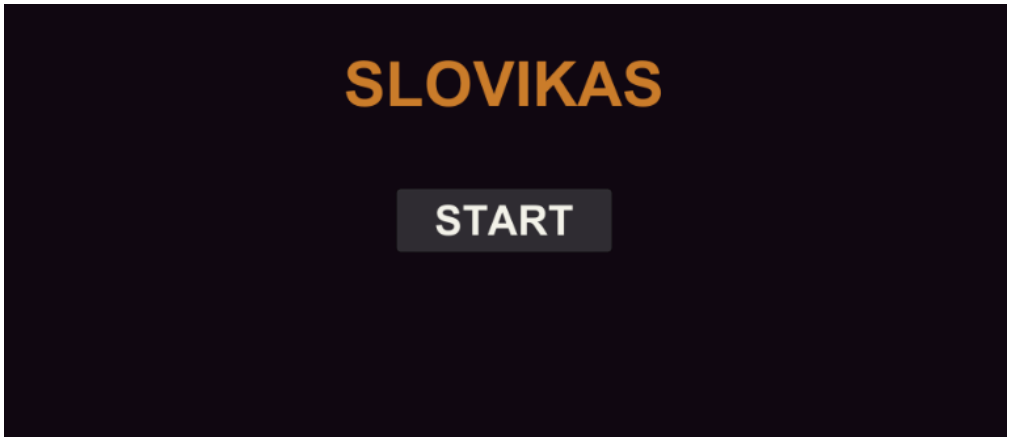

Figure 25. GameMenu scene

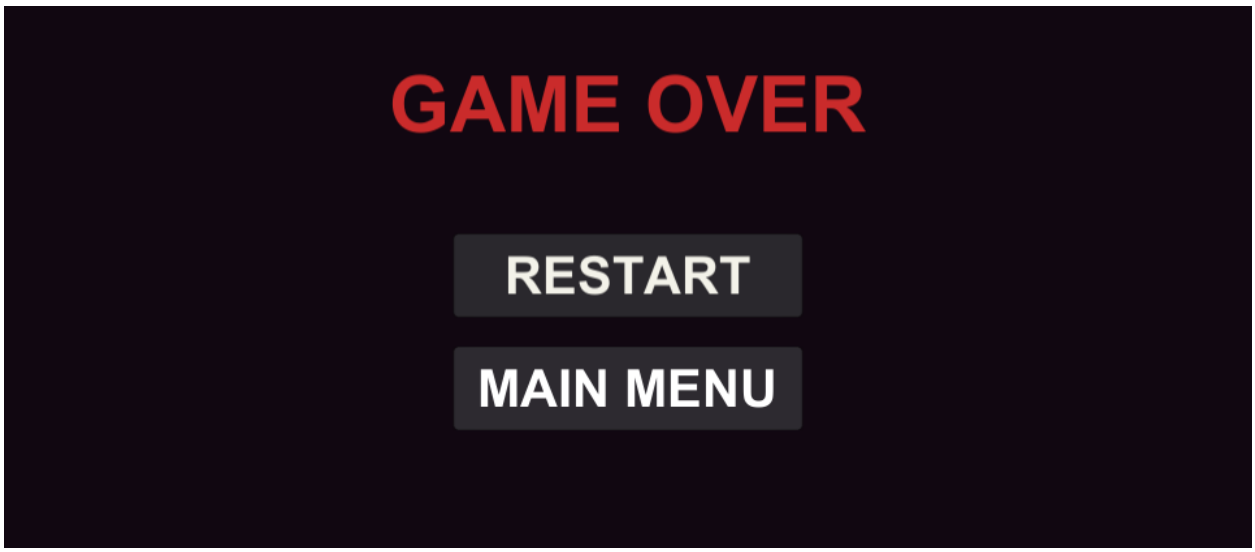

Figure 26. GameOver scene

1. void StartButton() [Table] (Gytis Burbeckas)
2. void RestartButton() [Table] (Gytis Burbeckas)
3. void MainMenuButton() [Table] (Gytis Burbeckas)

This function is created to load the game scene when the “START” button is pressed.

```
public void StartButton()
    {
        SceneManager.LoadScene("game");
    }
```

Table 23. Function StartButton()

This function is created to start a new game scene after losing when the “RESTART” button is pressed.

```
public void RestartButton()
{
    SceneManager.LoadScene("game");
}
```

Table 24. Function RestartButton()

This function is created to return to the main menu scene when the “MAIN MENU” button is pressed.

```
public void MainMenuButton()
{
    SceneManager.LoadScene("MainMenu");

}
```

Table 25. Function MainMenuButton()

Task #6 Create traps

Firstly, we created floor spikes animations using assets we downloaded earlier [Figure]. After that, we attached box collider 2D to traps and marked them as triggers. Finally, we created a

script and attached it to the player. It tells us when the player steps on floor spikes and calls a function that subtracts some health from the player. Assets for floor spikes animation were taken from the website [4]. (Vidas Buivydas)

Floor spikes animation: (Vidas Buivydas)

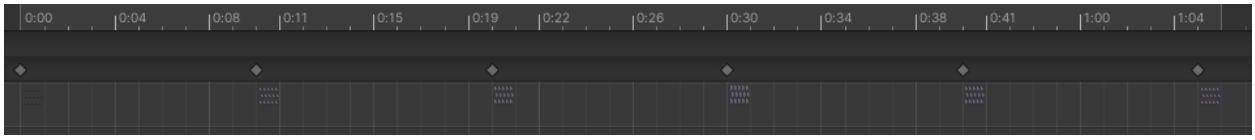

Figure 27. Floor spikes animation

Floor spikes in the game:

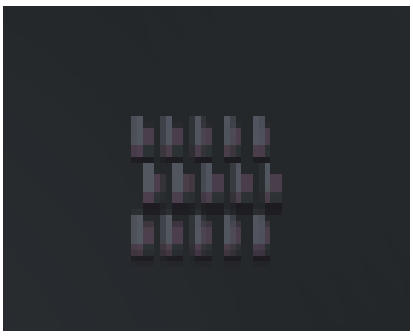

Figure 28. Floor spikes in game

Functions used in SpikesTrigger.cs:

1. void Awake() [Table] (Vidas Buivydas)
2. void OnTriggerEnter2D(Collider2D other) [Table] (Vidas Buivydas)
3. void OnTriggerStay2D(Collider2D other) [Table] (Vidas Buivydas)
4. bool IsSpikes(GameObject obj) [Table] (Vidas Buivydas)

Functions used in Player.cs:

1. IEnumerator StartInvulnerability() [Table] (Vidas Buivydas)
2. void TakeDamage(int amount) [Table] (Vidas Buivydas)

This function sets the object’s internal values to the corresponding player and its rigid body

```
private void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    player = GetComponent<Player>();
}
```

Table 26. Function Awake()

This function checks if a player stepped on floor spikes, if that's true it will call a method to reduce some health of the player

```
private void OnTriggerEnter2D(Collider2D other)
{
    var otherGameObject = other.gameObject;
    if (IsSpikes(otherGameObject))
    {
        player.TakeDamage(damage);
    }
}
```

Table 27. Functon OnTriggerEnter2D(Collider2D other)

This function checks if the player is staying on floor spikes, if that's true it will call a method to reduce some health of the player

```
void OnTriggerStay2D(Collider2D other)
{
    var otherGameObject = other.gameObject;
    if (IsSpikes(otherGameObject))
    {
        player.TakeDamage(damage);
    }
}
```

Table 28. Function OnTriggerStay2D(Collider2D other)

This function checks if the game object that the player collided with is floor spikes

```
private bool IsSpikes(GameObject obj)
{
    return obj.CompareTag(spikesTag);
}
```

Table 29. Function IsSpikes(GameObject obj)

This function starts invulnerability for a player, by default he can’t be damaged for one second

```
public IEnumerator StartInvulnerability()
{
    isVulnerabe = false;
    yield return new WaitForSeconds(invulnerabilityTime);
    isVulnerabe = true;
}
```

Table 30. Function StartInvulnerability()

This function checks if the player is vulnerable and if it is, his health is reduced by a certain amount and the invulnerability period is started.

```
public void TakeDamage(int amount)
{
    if (isVulnerabe == true)
    {
        if (health - amount <= 0 || health <= 0)
        {
            health = 0;
            UpdateLivesText();

            StopGame();
        }
        else
        {
            health -= amount;

            UpdateLivesText();
            StartCoroutine(StartInvulnerability());

        }
    }
}
```

Table 31. Function TakeDamage(int amount)

Laboratory work #2

List of tasks

1. Create main characters attacks
2. Improve the main menu
3. Create a floor generator
4. Create a settings menu
5. Update lighting, create shadows
6. Create health, mana, gold bars
7. Create destructible objects
8. Create a player selector
9. Create a level-up system
10. Create a door system

Solution

Task #1 Create main characters attacks

Added shooting animations for elf and wizard characters and particle effects for projectiles. The projectile is destroyed after hitting an object.

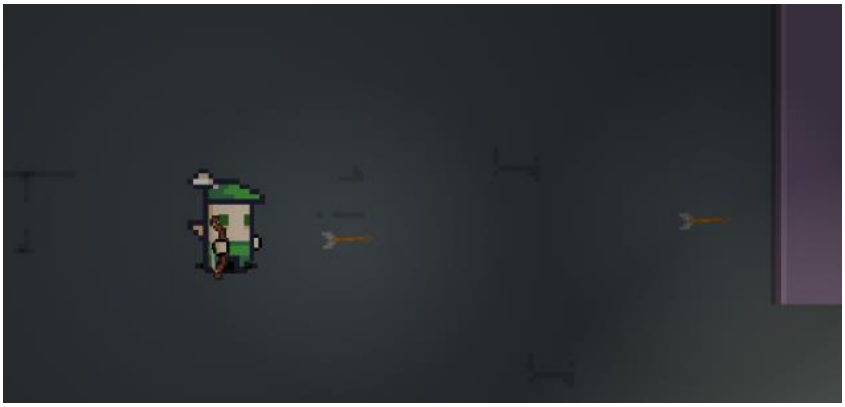

Figure 29 Elf shooting

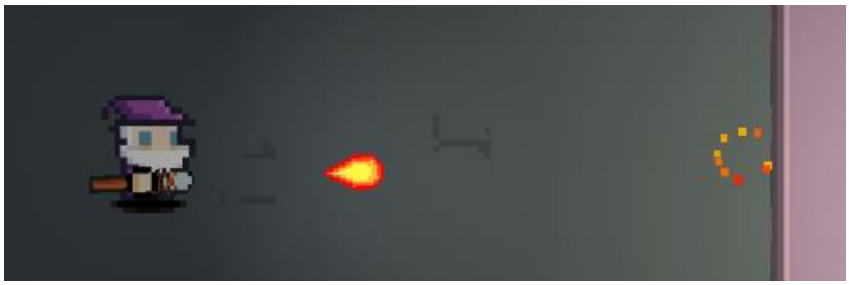

Figure 30 Wizard shooting

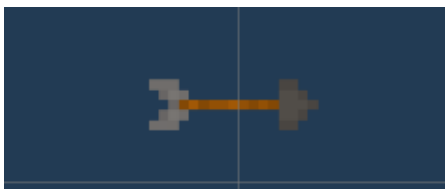

Figure 31 Arrow projectile

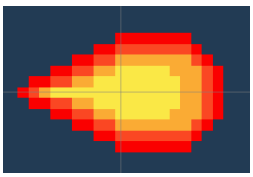

Figure 32 Fireball projectile

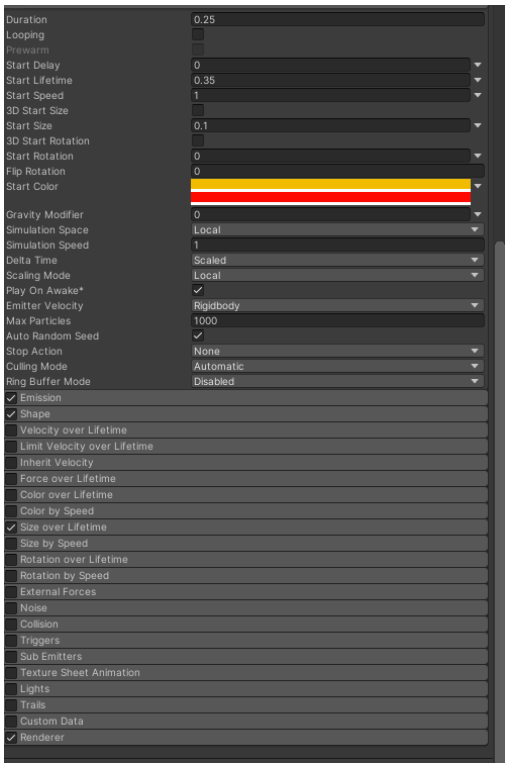

Figure 33 Projectile particle effect

OnTriggerEnter2D() function instantiates projectile particle effect after collision with other object. OnBecameInvisible() function destroys the projectile if it went out of field of view.

```
private void OnTriggerEnter2D(Collider2D other){
    Instantiate(impactEffect, transform.position, transform.rotation);
    Destroy(gameObject);
}

private void OnBecameInvisible(){
    Destroy(gameObject);
}
```

Figure 34 OnTriggerEnter2D() function

Void ShootingProjectiles() function instantiates projectile if left mouse trigger is clicked or held, shotCounter calculates time between shots.

```
void ShootingProjectiles(){

    if(Input.GetMouseButton(0)){
        shotCounter -= Time.deltaTime;

        if(shotCounter ≤ 0){
            Instantiate(bulletToFire, firepoint.position, firepoint.rotation);

            shotCounter = player.attackSpeed;
        }
    }
}
```

Figure 35 ShootingProjectiles() function

Task #2 Improve main menu

Game menu scene components were added to the main scene. The game menu scene is no longer needed. Additional buttons (Settings and Quit) were added. One script with functions [Table 32, Table 33, Table 34] was created to control all menu components and click events added to the buttons.

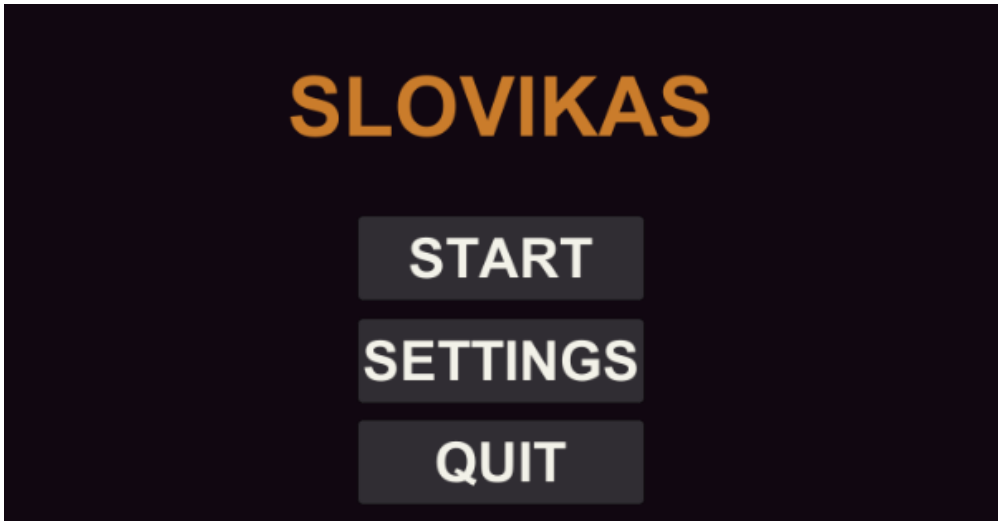

Figure 36. Improved main menu

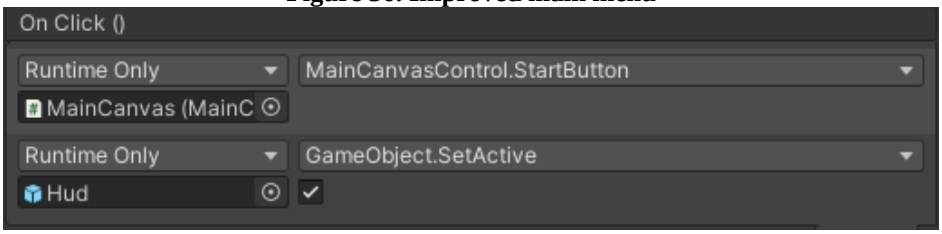

Figure 37. START button On click view

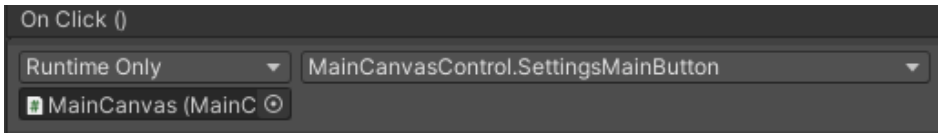

Figure 38. SETTINGS button On click view

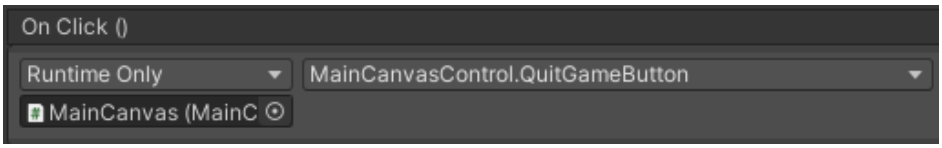

Figure 39. Quit button On click view

This function [Table 32] starts hides menus and starts the game. Also, it sets a timescale to 1 for the animation to work. (Gytis Burbeckas)

```
public void StartButton()
{
    Time.timeScale = 1;
    isGamePaused = false;
    place = "MAIN";
    settingsMenu.SetActive(false);
    mainMenu.SetActive(false);
    pauseMenu.SetActive(false);
}
```

Table 32. Function StartButton()

This function [Table 33] shows the settings menu for playing person. Also, it pauses the game so the player can't move and no animations are working. (Gytis Burbeckas)

```
public void SettingsMainButton()
{
    Time.timeScale = 0;
    isGamePaused = true;
    settingsMenu.SetActive(true);
    mainMenu.SetActive(false);
    pauseMenu.SetActive(false);
    hpxpgoldMenu.SetActive(false);
    place = "MAIN";
}
```

Table 33. Function SettingsManimenu()

This function [Table 34] quits the game application. (Gytis Burbeckas)

```
public void QuitGameButton()
{
    Application.Quit();
    Debug.Log("Game is exiting");
}
```

Table 34. Function QuitGameButton()

Task #3 Create floor generator

Firstly, floor templates were created [Figure] and added to the RoomTemplates game object [Figure]. Then RoomSpawner script was updated to generate a random floor when the room is generated. (Vidas Buivydas)

Figure 40. Floor templates

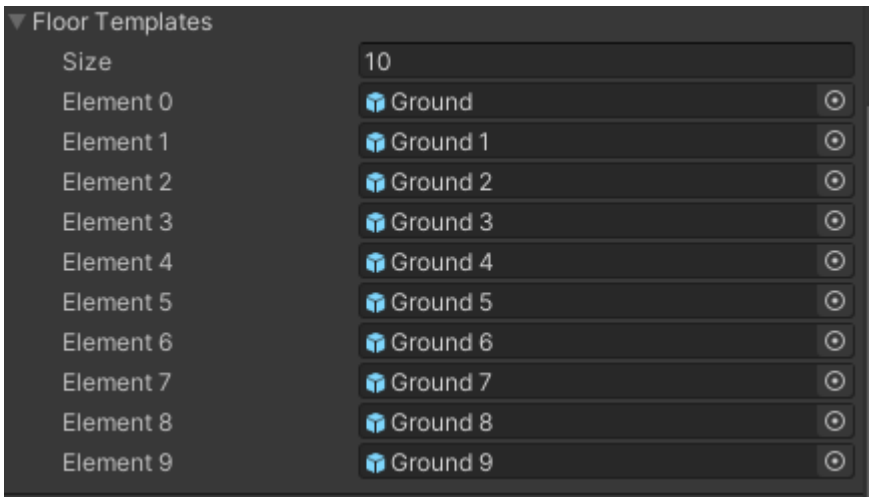

Figure 41. Floor templates assigned to a game object

Functions updated in RoomSpawner.cs:

1. void Spawn()[Figure](Vidas Buivydas)

This function was updated to call a CreateFloor() method every time a room is created

```
void Spawn()
{
    if (spawned == false)
    {
        if (openingDirection == 1)
        {
            //need to spawn a room with a bottom door
            rand = IndexGenerator(manager.roomCount, templates.bottomRooms.Length);
            Instantiate(templates.bottomRooms[rand], transform.position, templates.bottomRooms[rand].transform.rotation);
            manager.roomCount++;
            CreateFloor();
        }
        else if (openingDirection == 2)
        {
            //need to spawn a room with a top door
            rand = IndexGenerator(manager.roomCount, templates.topRooms.Length);
            Instantiate(templates.topRooms[rand], transform.position, templates.topRooms[rand].transform.rotation);
            manager.roomCount++;
            CreateFloor();
        }
        else if (openingDirection == 3)
        {
            //need to spawn a room with a left door
            rand = IndexGenerator(manager.roomCount, templates.leftRooms.Length);
            Instantiate(templates.leftRooms[rand], transform.position, templates.leftRooms[rand].transform.rotation);
            manager.roomCount++;
            CreateFloor();
        }
        else if (openingDirection == 4)
        {
            //need to spawn a room with a right door
            rand = IndexGenerator(manager.roomCount, templates.rightRooms.Length);
            Instantiate(templates.rightRooms[rand], transform.position, templates.rightRooms[rand].transform.rotation);
            manager.roomCount++;
            CreateFloor();
        }
        spawned = true;
    }
}
```

Figure 42. function void Spawn()

Functions created in RoomSpawner.cs:

1. void CreateFloor()[Figure](Vidas Buivydas)

This function selects a random floor template and generates it

```
void CreateFloor()
{
    int rnd = Random.Range(0, templates.floorTemplates.Length -1);
    Instantiate(templates.floorTemplates[rnd], transform.position, templates.floorTemplates[rnd].transform.rotation);
}
```

Figure 43. function void CreateFloor()

Task #4 Create a settings menu

Added new canvas to Main canvas to create settings page. The return button was created to go back to the main menu. The slider was used to create a volume controller. The toggle element was used to create the Fullscreen setting. The dropdown table was used to create the resolution setting. Also, a script [Table 35,Table 36,Table 37,Table 38,Table 39] was written to give these elements functionality.

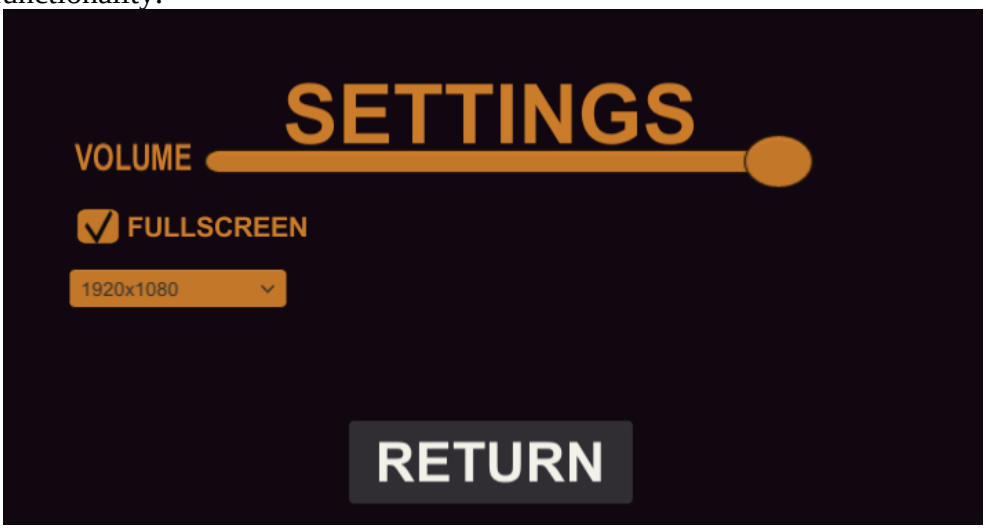

Figure 44. Settings page

This function [Table 35] is used to set the value of AudioMixer to the value of the volume slider. (Gytis Burbeckas)

```
public void SetVolume(float volume)
{
    audioMixer.SetFloat("volume", volume);
}
```

Table 35. Function SetVolume(float volume)

This function [Table 36] is used to make a game in full screen or disable full screen. (Gytis Burbeckas)

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

Table 36. Function SetFullscreen(bool isFullScreen)

This function [Table 37] is used to get all resolutions of the computer and add their values to the dropdown table of resolutions. (Gytis Burbeckas)

```
private void Start()
{
    Time.timeScale = 0;
    isGamePaused = true;
     resolutions = Screen.resolutions.Select(resolution => new Resolution { width = resolution.width, height = resolution.height }).Distinct().ToArray();
    resolutionDropdown.ClearOptions();
    List<string> options = new List<string>();
    int currentResolutionIndex = 0;

    for (int i = 0; i < resolutions.Count(); i++)
    {
        string option = resolutions.ElementAt(i).width + "x" + resolutions.ElementAt(i).height;
        options.Add(option);

        if (resolutions.ElementAt(i).width == Screen.currentResolution.width &&
            resolutions.ElementAt(i).height == Screen.currentResolution.height)
        {
            currentResolutionIndex = i;
        }
    }
    resolutionDropdown.AddOptions(options);
    resolutionDropdown.value = currentResolutionIndex;
    resolutionDropdown.RefreshShownValue();
}
```

Table 37. Function Start()

This function [Table 38] is used to set the screen resolution to chosen. (Gytis Burbeckas)

```
public void SetResolution(int resolutionIndex)
{
    Resolution resolution = resolutions.ElementAt(resolutionIndex);
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
}
```

Table 38. Function SetResolution(int resolutionIndex)

This function [Table 39] is used to go back to the game or main menu. If a player came to the settings menu from the main menu, he will go back to the main menu. If a player came to the settings menu from the pause menu, he will go back to the pause menu. (Gytis Burbeckas)

```
public void returnFromSettings()
{
    if (place == "MAIN")
    {
        settingsMenu.SetActive(false);
        mainMenu.SetActive(true);
        pauseMenu.SetActive(false);
        hpxpgoldMenu.SetActive(false);
        Time.timeScale = 0;
        isGamePaused = true;
        place = "MAIN";
    }
    else if (place == "PAUSE")
    {
        settingsMenu.SetActive(false);
        mainMenu.SetActive(false);
        pauseMenu.SetActive(true);
        Time.timeScale = 0;
        isGamePaused = true;
        place = "PAUSE";
    }
```

Table 39. Function returnFromSettings()

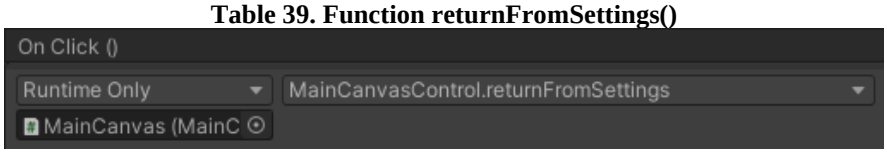

Figure 45. Return button On click view

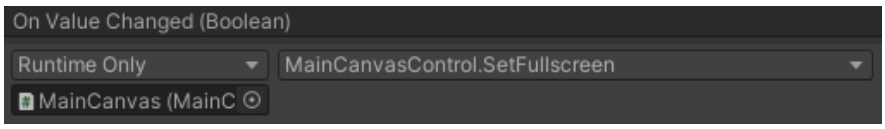

Figure 46. Fullscreen toggle On Value Changed view

Figure 47. Resolutions dropdown table On value Changed view

Figure 48. Volume slider On Value Changed view

Task #5 Update lighting, create shadows

Firstly, torch lighting was updated by adding a Flow Machine component [Figure] to change the light range and intensity every 0.1 seconds. After that Shadow Caster 2D component [Figure] was attached to every character and the shadow casting shape was adjusted. (Vidas Buivydas)

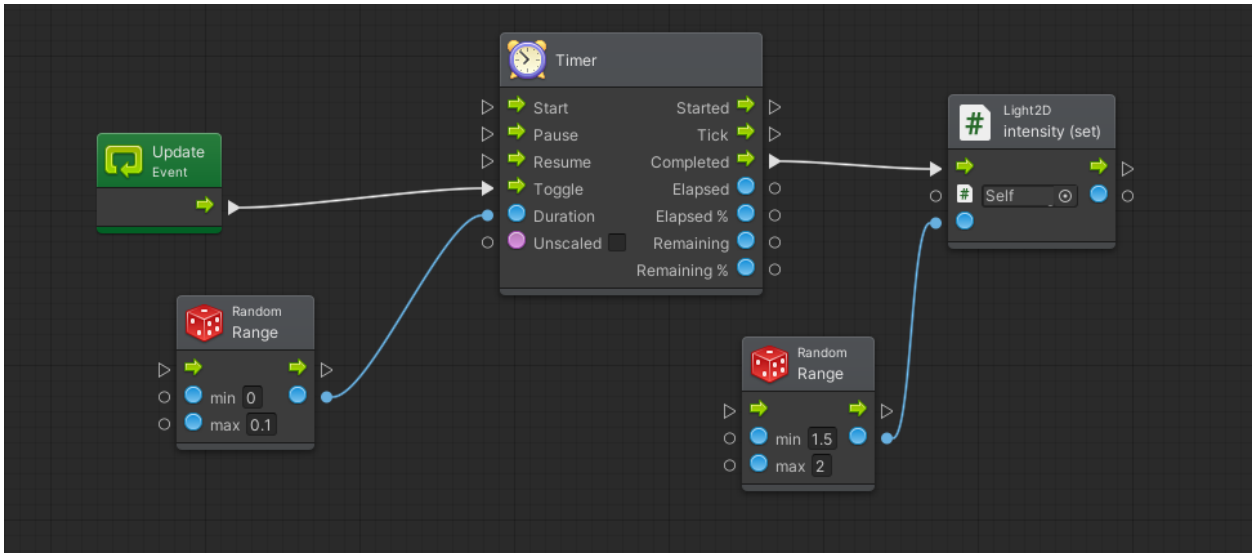

Figure 49. Torch Flow Machine

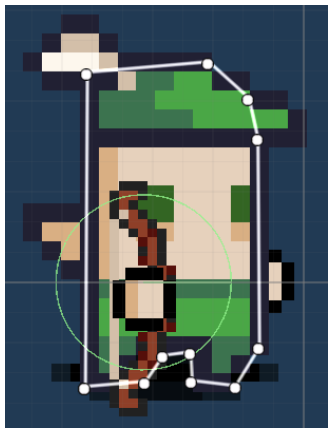

Figure 50. Shadow Caster 2D edited shape for the character

Task #6 Create health, mana, gold bars

A new hud canvas was created. A simple element with sprite was used to show the amount of gold. Sliders were used to show players mana and health. Scripts [Table 40, Table 41]were created to show the same value of health and mana of the player.

Figure 51. Health bar, mana bar, gold

This function [Table 40] is used to set the value of the health slider to the amount of health that the player has. (Gytis Burbeckas)

```
public void SetHealth(int health)
{
    healthSlider.value = health;
    textComp.text = healthSlider.value.ToString();
}
```

Table 40. Function SetHealth(int health)

This function [Table 41] is used to set the value of the health slider to the amount of health that the player has. (Gytis Burbeckas)

```
public void SetMana(int mana)
{
    manaSlider.value = mana;
    textComp.text = manaSlider.value.ToString();
}
```

Table 41. Function SetMana(int mana)

Task #7 Create destructable objects

Firstly, for each destructible (box, pot, barrel, explosive barrel (from [5])) hit [Figure52.] and destroy [Figure53.] animations were created and used certain in the animation controller [Figure54.]. After that script for destructibles was created and attached to every destructible object.
(Vidas Buivydas)

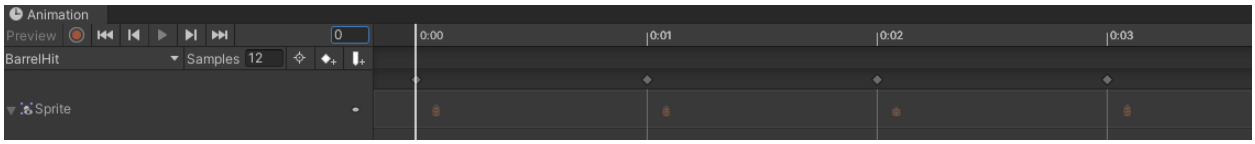

Figure52. Destructible is hit animation

Figure53. Destructible is destroyed animation

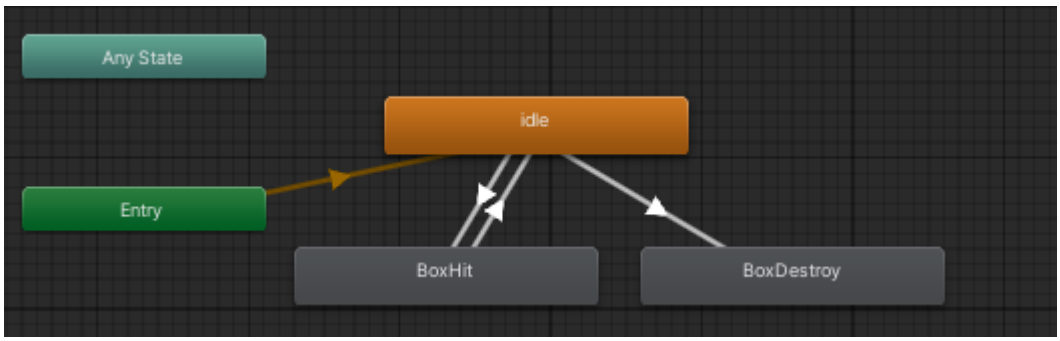

Figure54. Destructible animation controller

Functions used in Destructables.cs:

1. void Start() [Figure 55.] (Vidas Buivydas)
2. private void OnTriggerEnter2D(Collider2D other) [Figure 56.] (Matas Jonauskas)
3. private IEnumerator WaitForAnimation() [Figure 57.] (Vidas Buivydas)

This function assigns the Animator component before the first frame of the game.

```
void Start()
{
    animator = GetComponent<Animator>();
}
```

Figure 55. function void Start()

This function checks if the collided object is a weapon and if it is, subtracts health from the object and respectively runs hit or destruction animations.

```
private void OnTriggerEnter2D(Collider2D other){

    if(other.tag == "Weapon")
    {
        health -= Player.instance.damage;
        if(health > 0){
            animator.SetBool("IsDamaged", true);
            StartCoroutine(WaitForAnimation());
        }else{
            animator.SetBool("IsDestroyed", true);
            collider.enabled = false;
        }
    }
}
```

Figure 56. function void OnTriggerEnter2D(Collider2D other)

This function waits 0.05 seconds before changing animators values.

```
private IEnumerator WaitForAnimation()
{
    yield return new WaitForSeconds(0.05F);
    animator.SetBool("IsDamaged", false);
}
```

Figure 57. function IEnumerator WaitForAnimation()

Task #8 Create player selector

The player selector was created for three main characters. Initial character(Boy) must select one of the three different items with pressing the E button. Sword is for choosing a knight, the bow for an elf and the staff for a wizard.

Figure 58 Player selection

When a player comes near chosen item message appears ”Press “E” to select”(OnTrigerEnter2D function) and after the player presses the “E” key the function below assigns the player to a selected character, initial character(Boy) is deleted from the scene, a new character is instantiated, main camera and level system also is assigned to the new character.

```
void Update()
{
    if(canSelect){
        if(Input.GetKeyDown(KeyCode.E))
        {
            Vector3 playerPos = PlayerController.instance.transform.position;

            Destroy(PlayerController.instance.gameObject);

            PlayerController newPlayer = Instantiate(playerToSpawn, playerPos, playerToSpawn.transform.rotation);
            PlayerController.instance = newPlayer;

            gameObject.SetActive(false);


            CameraController.instance.target = newPlayer.transform;
            List<GameObject> goList = new List<GameObject>();
            goList = new List<GameObject>(GameObject.FindGameObjectsWithTag("Player"));
            levelSystem.assignPlayer(goList[1].GetComponent<Player>());
        }
    }
}
```

Figure. 59 Character selector Update() function

```
0 references
private void OnTriggerEnter2D(Collider2D other){
    if(other.tag == "Player"){
        canSelect = true;
        messanger.SetActive(true);
    }
}

0 references
private void OnTriggerExit2D(Collider2D other){
    if(other.tag == "Player"){
        canSelect = false;
        messanger.SetActive(false);
    }
}
```

Figure. 60 OnTriggerEnte2D and OnTriggerExit2D functions

Task #9 Create a level-up system

Firstly, XP bar [Figure 61] and fill [Figure 62] was created. After that canvas [Figure 63] using buttons from [7] for levelling up was created. Finally, script for levels and characters' attribute updates was created. (Vidas Buivydas)

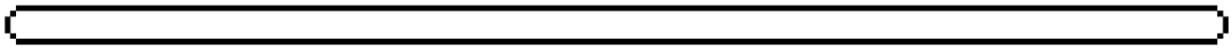

Figure 61. XP bar

Figure 62. XP fill

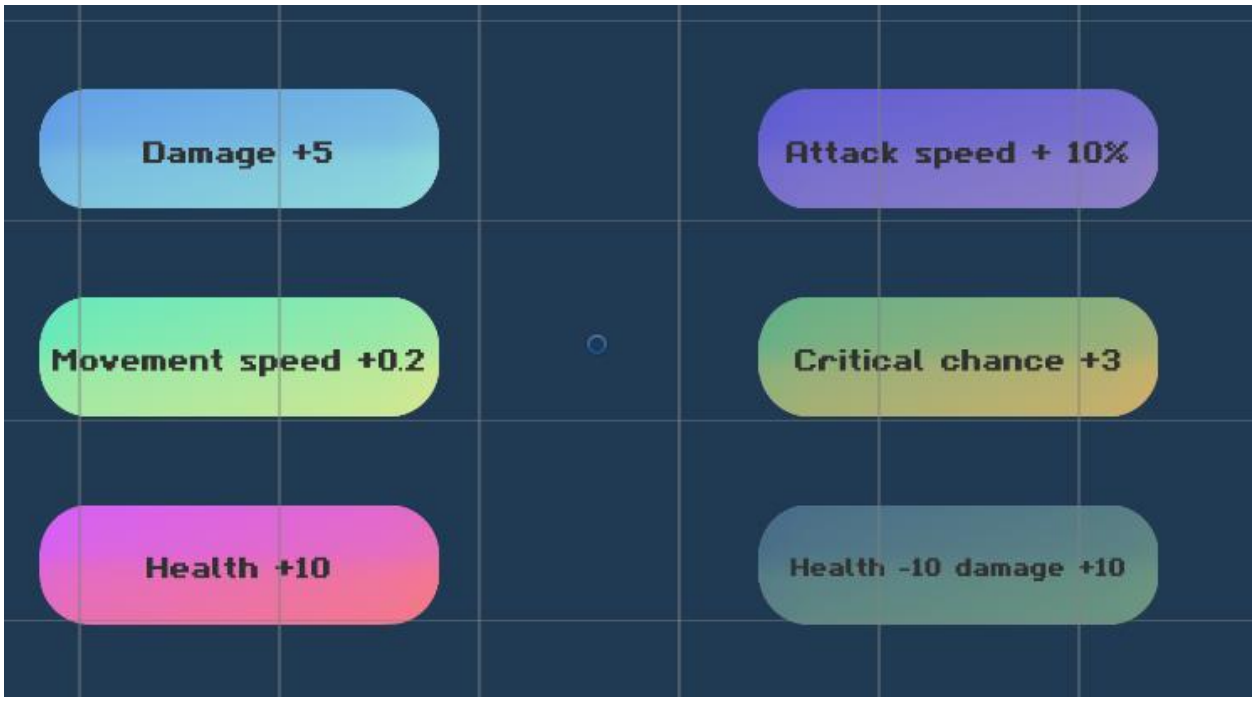

Figure 63. Level up canvas

Functions used in LevelSystem.cs:

1. void Start() [Figure 64] (Vidas Buivydas)
2. void Update() [Figure 65] (Vidas Buivydas)
3. public void UpdateXpUI() [Figure 66] (Vidas Buivydas)
4. public void GainExperienceFlatRate(int xpGained) [Figure 67] (Vidas Buivydas)
5. public void LevelUp() [Figure 68] (Vidas Buivydas)
6. public void assignPlayer(Player player) [Figure 69] (Vidas Buivydas)
7. public void updateText() [Figure 70] (Vidas Buivydas)
8. private void UpdateDMGButtonText() [Figure 71] (Vidas Buivydas)
9. private void UpdateHPButtonText() [Figure 72] (Vidas Buivydas)
10. private void UpdateMaxHPButtonText() [Figure 73] (Vidas Buivydas)
11. public void IncreaseDamage() [Figure 74] (Vidas Buivydas)
12. public void IncreaseHealth() [Figure 75] (Vidas Buivydas)
13. public void IncreaseAttackSpeed() [Figure 76] (Vidas Buivydas)
14. public void IncreaseCriticalChance() [Figure 77] (Vidas Buivydas)

15. public void IncreaseMovementSpeed() [Figure 78] (Vidas Buivydas)
16. public void IncreaseMaxHealth() [Figure 79] (Vidas Buivydas)

This function at the start of the game updates XP bar and assigns main canvas object

```
void Start()
{
    FIllXPImage.fillAmount = currentXP / requiredXP;
    UpdateXPImage.fillAmount = currentXP / requiredXP;
    mainCanvas = gameObject.transform.parent.parent.GetChild(0).transform.GetComponent<MainCanvasControl>();
}
```

Figure 64. function void Start()

This function calls XP UI updater method, adds XP to player and checks if he is able to level up

```
void Update()
{
    UpdateXpUI();
    if (Input.GetKeyDown(KeyCode.Equals))
    {
        //FIllXPImage.fillAmount += 0.2f;
        GainExperienceFlatRate(23);
    }

    if(currentXP >= requiredXP)
    {
        LevelUp();
    }
}
```

Figure 65. function void Update()

This function updates XP bar

```
public void UpdateXpUI()
{
    float xpFraction = currentXP / requiredXP;
    float FXP = FIllXPImage.fillAmount;
    if(FXP < xpFraction)
    {
        delayTimer += Time.deltaTime;
        UpdateXPImage.fillAmount = xpFraction;
        //if(delayTimer > 3)
       // {
            lerpTimer += Time.deltaTime;
            float percentComplete = lerpTimer / 4;
            FIllXPImage.fillAmount = Mathf.Lerp(FXP, UpdateXPImage.fillAmount, percentComplete);
        //}
    }
}
```

Figure 66. function void UpdateXpUI()

This function gives player XP

```
public void GainExperienceFlatRate(int xpGained)
{
    currentXP += xpGained;
    lerpTimer = 0f;
}
```

Figure 67. function void GainExperienceFlatRate(int xpGained)

This function updates player level, assigns new required XP ammount, calls other functions so that player can assign skill point.

```
public void LevelUp()
{
    level++;
    FIllXPImage.fillAmount = 0;
    UpdateXPImage.fillAmount = 0;
    currentXP = currentXP - requiredXP;
    requiredXP = Convert.ToInt32(requiredXP + 0.35 * requiredXP);
    LevelText.text = level.ToString();
    mainCanvas.GameLevelUp();
    updateText();
}
```

Figure 68. function void LevelUp()

This function assigns player

```
public void assignPlayer(Player player)
{
    this.player = player;
}
```

Figure 69. function void assignPlayer(Player player

This function calls methods that update text of level up canvas buttons

```
public void updateText()
{
    UpdateDMGButtonText();
    UpdateHPButtonText();
    UpdateMaxHPButtonText();
}
```

Figure 70. function void updateText()

This function updates damage button text

```
private void UpdateDMGButtonText()
{
    DMGBUttonText = GameObject.Find("DMGButtonText").GetComponent<Text>();
    DMGBUttonText.text = $"Damage + {level / 2 + 3}";
}
```

Figure 71. function UpdateDMGButtonText()

his function updates health button text

```
private void UpdateHPButtonText()
{
    HPButtonText = GameObject.Find("HPButtonText").GetComponent<Text>();
    HPButtonText.text = $"Health + {player.maxHealth / 3}";
}
```

Figure 72. function UpdateHPButtonText()

his function updates the max health button text

```
private void UpdateMaxHPButtonText()
{
    MaxHPButtonText = GameObject.Find("MaxHPButtonText").GetComponent<Text>();
    MaxHPButtonText.text = $"Max Health + {level * 2.5 + 5}";
}
```

Figure 73. function UpdateMaxHPButtonText()

This function increases player damage

```
public void IncreaseDamage()
{
    player.damage += level/2 + 3;
}
```

Figure 74. function IncreaseDamage()

This function increases player health

```
public void IncreaseHealth()
{
    player.health += player.maxHealth / 3;
}
```

Figure 75. function IncreaseHealth()

This function increases player attack speed

```
public void IncreaseAttackSpeed()
{
    player.attackSpeed -= (player.attackSpeed /10);
}
```

Figure 76. function IncreaseAttackSpeed()

This function increases player critical chance

```
public void IncreaseCriticalChance()
{
    player.criticalChance += 3;
}
```

Figure 77. function IncreaseCriticalChance()

This function increases player movement speed

```
public void IncreaseMovementSpeed()
{
    player.movementSpeed += 0.2f;
}
```

Figure 78. function IncreaseMovementSpeed()

This function increases max health

```
public void IncreaseMaxHealth()
{
    player.maxHealth += Convert.ToInt32(level * 2.5 + 5);
}
```

Figure 79. function IncreaseMaxHealth()

Task #10 Create a door system

Firstly, box colliders for entrance and doors (from [3]) were added to every room template [Figure 80]. After that script for the opening and closing the doors, destroying entrance colliders was created.

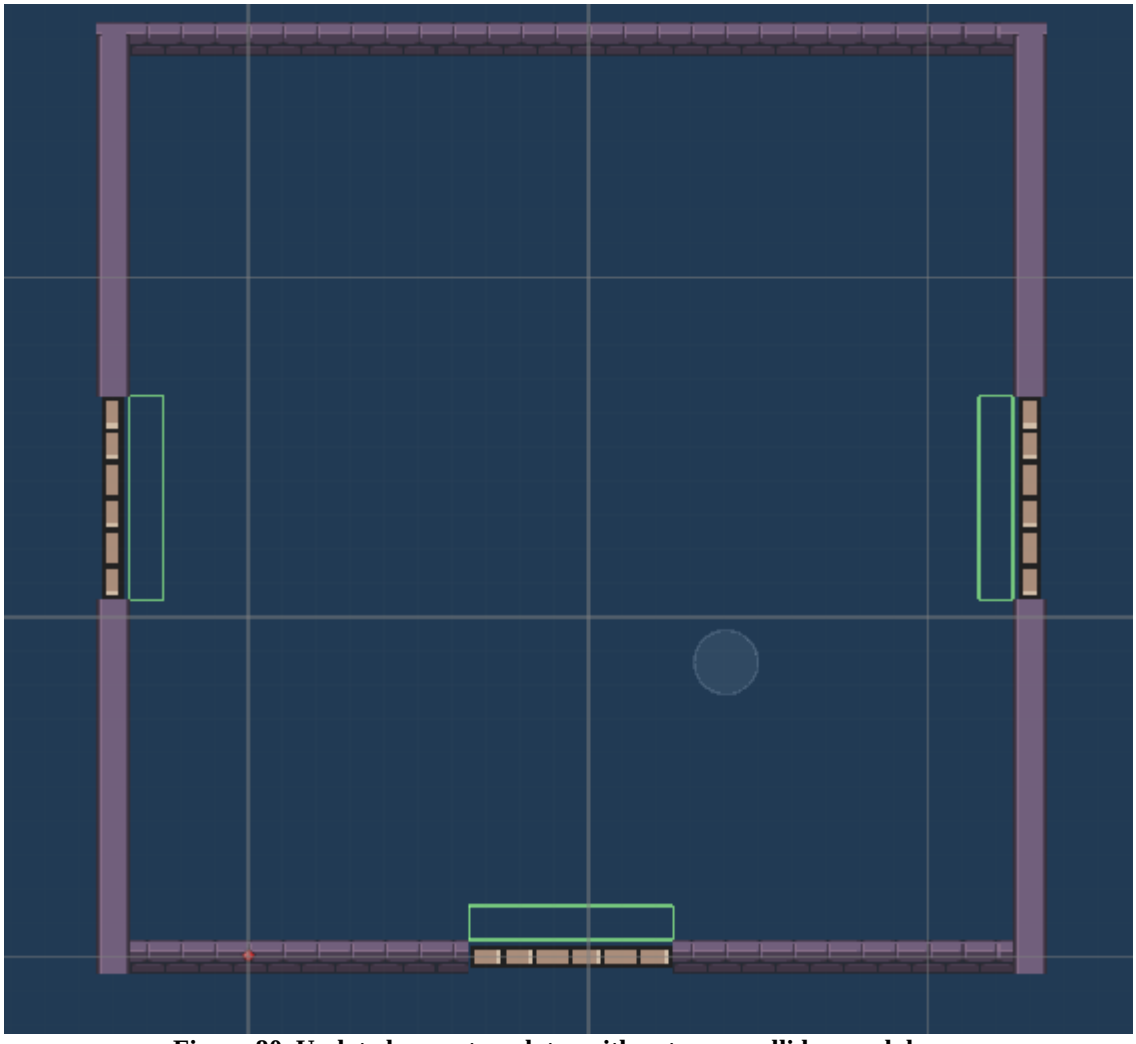

Figure 80. Updated room templates with entrance colliders and doors

Functions used in RoomClose.cs:

1. private void Awake() [Figure 81] (Vidas Buivydas)
2. private void OnTriggerEnter2D(Collider2D other) [Figure 82] (Vidas Buivydas)
3. private IEnumerator MakeVisible(GameObject obj) [Figure 83] (Vidas Buivydas)
4. private IEnumerator MakeInvisible(GameObject obj) [Figure 84] (Vidas Buivydas)
5. private bool IsEntrance(GameObject obj) [Figure 85] (Vidas Buivydas)
6. private bool IsDoorSprite(GameObject obj) [Figure 86] (Vidas Buivydas)

7. private bool IsDoorObject(GameObject obj) [Figure 87] (Vidas Buivydas)
8. void Update() [Figure 88] (Vidas Buivydas)

This method assigns RigidBody2D, player, SpriteRenderer to objects.

```
private void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    player = GetComponent<Player>();
    sprite = GetComponent<SpriteRenderer>();
}
```

Figure 81. function void Awake()

This method makes doors visible, activates doors‘ colliders and destroys entrance colliders.

```
private void OnTriggerEnter2D(Collider2D other)
{
    var otherGameObject = other.gameObject;
    if (IsEntrance(otherGameObject))
    {
        Destroy(otherGameObject.transform.parent.gameObject);
        List<GameObject> goList = new List<GameObject>();
        goList = new List<GameObject>(GameObject.FindGameObjectsWithTag(doorTag));
        if (goList.Count > 0)
        {
            for (int i = 0; i < goList.Count; i++)
            {
                GameObject go = goList[i];
                Transform[] ts = go.GetComponentsInChildren<Transform>(true);
                foreach (Transform o in ts)
                {
                    o.gameObject.active = true;
                    Transform[] ts2 = o.gameObject.GetComponentsInChildren<Transform>(true);
                    foreach (Transform o2 in ts2)
                    {
                        if (IsDoorSprite(o2.gameObject))
                        {
                            StartCoroutine(MakeVisible(o2.gameObject));
                        }
                    }
                }
            }
        }

    }
}
```

Figure 82. function OnTriggerEnter2D(Collider2D other)

This method makes game object visible gradually in 0.5 seconds

```
private IEnumerator MakeVisible(GameObject obj)
{
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.1f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.2f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.3f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.4f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.5f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.6f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.7f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.8f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.9f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 1);
}
```

Figure 83. function Ienumerator MakeVisible(GameObject obj)

This method makes game object invisible gradually in 0.5 seconds

```
private IEnumerator MakeInvisible(GameObject obj)
{
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 1);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.9f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.8f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.7f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.6f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.5f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.4f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.3f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.2f);
    yield return new WaitForSeconds(0.05F);
    obj.GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, 0.1f);
    obj.transform.parent.gameObject.active = false;
}
```

Figure 84. function IEnumerator MakeInvisible(GameObject obj)

This method checks if another game object is the entrance

```
private bool IsEntrance(GameObject obj)
{
    return obj.CompareTag(colliderTag);
}
```

Figure 85. function bool IsEntrance(GameObject obj)

This method checks if another game object is door sprite

```
private bool IsDoorSprite(GameObject obj)
{
    return obj.CompareTag(doorSpriteTag);
}
```

Figure 86. function bool IsDoorSprite(GameObject obj)

This method checks if other game object is the door object

```
private bool IsDoorObject(GameObject obj)
{
    return obj.CompareTag(doorObjectTag);
}
```

Figure 87. function bool IsDoorObject(GameObject obj)

This method makes the door invisible and removes the doors‘ colliders.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Equals))
    {

        List<GameObject> goList = new List<GameObject>();
        goList = new List<GameObject>(GameObject.FindGameObjectsWithTag(doorTag));
        if (goList.Count > 0)
        {
            for (int i = 0; i < goList.Count; i++)
            {
                GameObject go = goList[i];
                Transform[] ts = go.GetComponentsInChildren<Transform>(true);
                foreach (Transform o in ts)
                {
                    if (IsDoorObject(o.gameObject))
                    {
                        Transform[] ts2 = o.gameObject.GetComponentsInChildren<Transform>();
                        foreach (Transform o2 in ts2)
                        {
                            if (IsDoorSprite(o2.gameObject))
                            {
                                StartCoroutine(MakeInvisible(o2.gameObject));
                            }
                        }
                    }
                }
            }
        }

    }

}
```

Figure 88. function void Update()

Laboratory work #3

List of tasks

1. Music and sound effects
2. Enemies
3. Physics
4. Multiplayer

Solution

Task #1 Music and sound effects

Firstly, the music and sound effects were downloaded from the websites [7,9]. For the main menu, gameplay and defeat music there were created empty game objects with Audio source elements [Figure 89, Figure 90, Figure 91]. Each game object is set active based on the pressed buttons. Also, audio source was added to each destructible object and is played when the object is destroyed [Figure 92, Figure 93, Figure 94, Figure 95]. The audio source was added to fireball and arrow prefabs to play a sound when player shots [Figure 96, Figure 97]. Sound is played when these objects hit something. Scripts were written to control sounds and music. (Gytis Burbeckas)

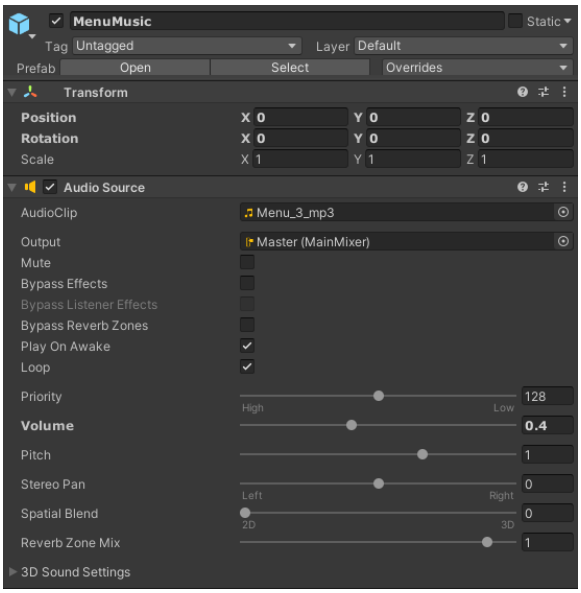

Figure 89. Main menu Audio Source

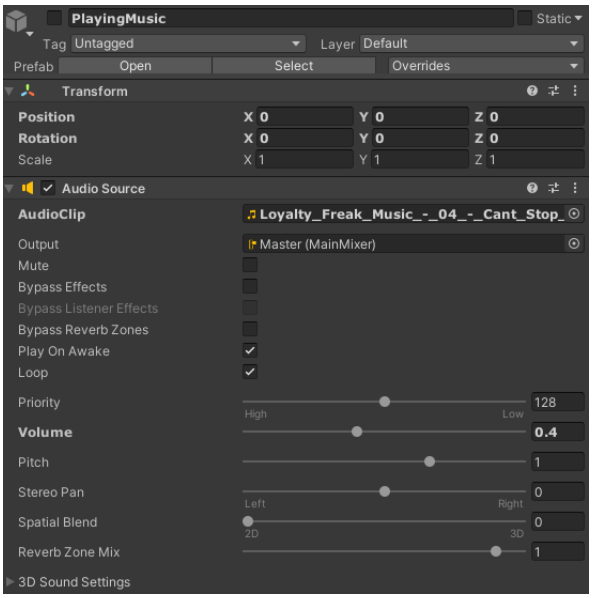

Figure 90. Gameplay music

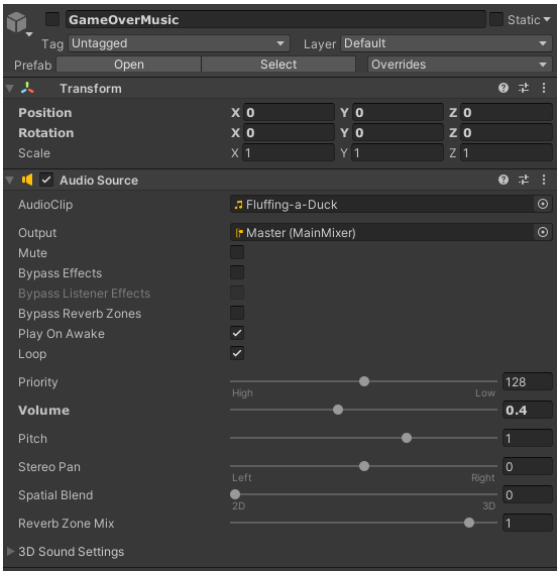

Figure 91. Game over music

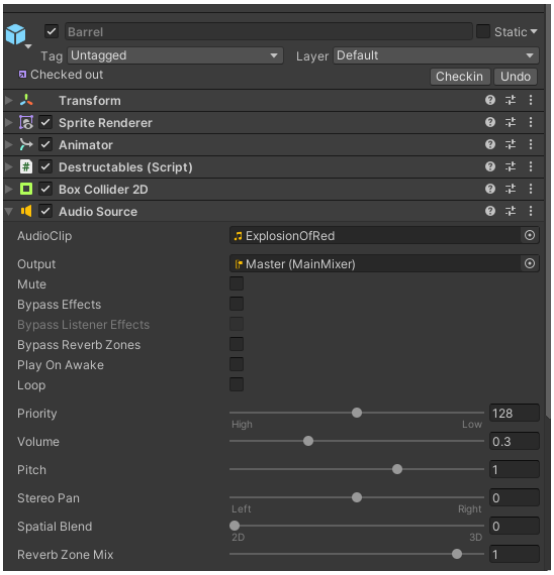

Figure 92. Barrel Audio Source

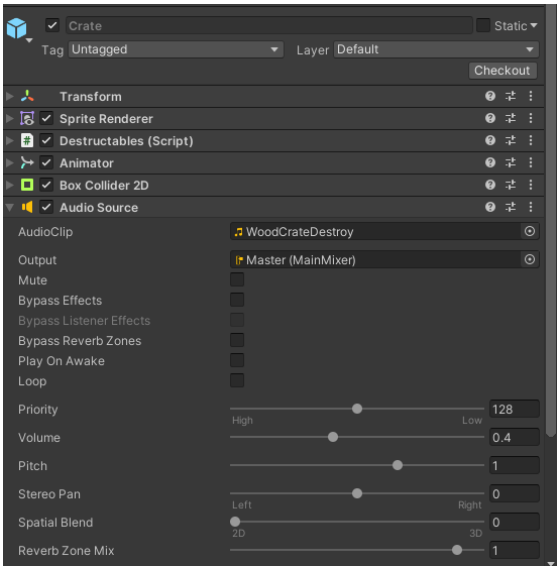

Figure 93. Crate Audio Source

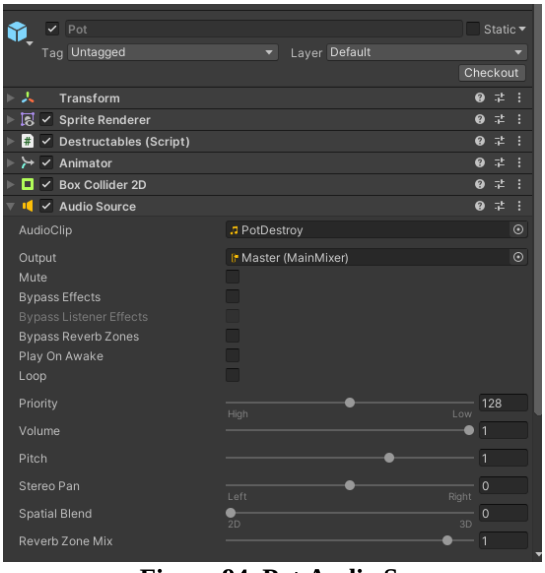

Figure 94. Pot Audio Source

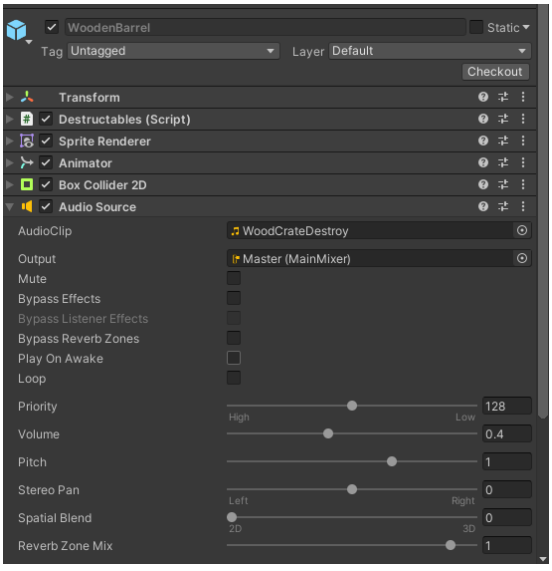

Figure 95. Wooden barrel Audio Source

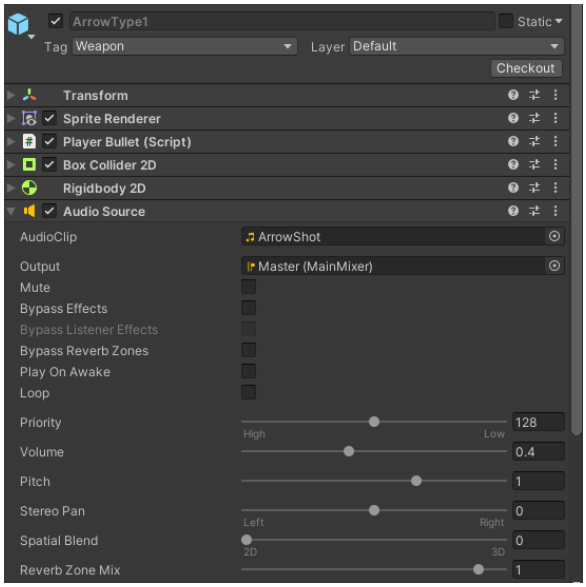

Figure 96. Arrow Audio Source

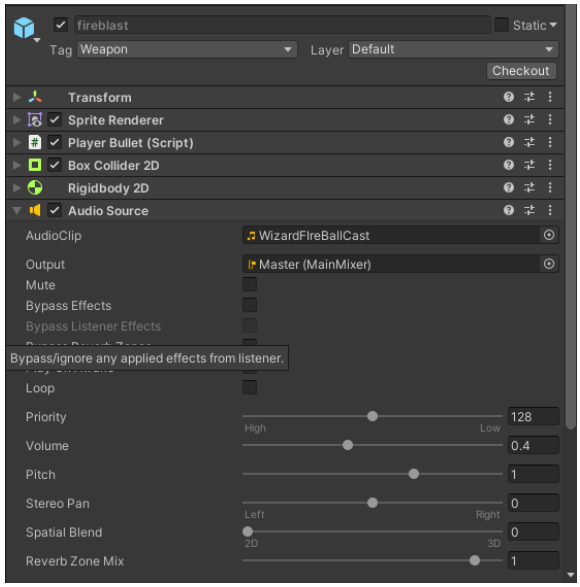

Figure 97. Fireblast Audio Source

This script was modified so now it stops shooting sound and activates fireball or arrow impact sound [Figure 98]. (Gytis Burbeckas)

```
private void OnTriggerEnter2D(Collider2D other){

    if(other.tag != "HealthPotion" && other.tag != "ManaPotion" && other.tag != "Gold1"){

        StartCoroutine(WaitForAnimation());
        GetComponent<AudioSource>().Stop();
        Instantiate(impactEffect, transform.position, transform.rotation);
        impactSound.Play();
    }
}
```

Figure 98. OnTriggerEnter2D(Collider2D other) script

This script is responsible for playing sound when player shots based on the bullet it plays fireball fire sound or arrow fire sound [Figure 99]. (Gytis Burbeckas)

```
void Start()
{
    Debug.Log(theRB.name);
    if (theRB.name == "ArrowType1(Clone)")
    {
        GameObject impactObject = GameObject.Find("ArrowHit");
        impactSound = impactObject.GetComponent<AudioSource>();
        GameObject shotSoundGameObject = GameObject.Find("ArrowShot");
        shotSound = shotSoundGameObject.GetComponent<AudioSource>();

        GetComponent<AudioSource>().Play();
    }
    else if(theRB.name == "fireblast(Clone)")
    {

        GameObject impactObject = GameObject.Find("FireBallHit");
        impactSound = impactObject.GetComponent<AudioSource>();
        Debug.Log(impactSound);
        GameObject shotSoundGameObject = GameObject.Find("WizardFireballCast");
        shotSound = shotSoundGameObject.GetComponent<AudioSource>();
        GetComponent<AudioSource>().Play();
    }
}
```

Figure 99. void Start()

Task #2 Enemies

There were created two types of enemies. Both enemy types start to run to the player if he steps in their visibility range. One enemy type attacks from the short distance [Figure 109], second type attacks from far distance [Figure 101], while shooting magma balls[Figure 102].

Figure 100 First enemy type

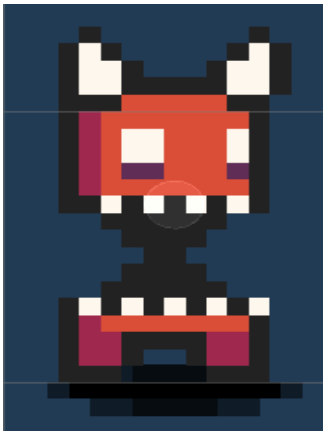

Figure 101 Second enemy type

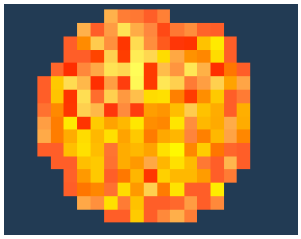

Figure 102 Magma ball

Then, added animations. For both enemies type there are two animations: idle and running.

Idle animation window of first type enemy(duration 30 s):

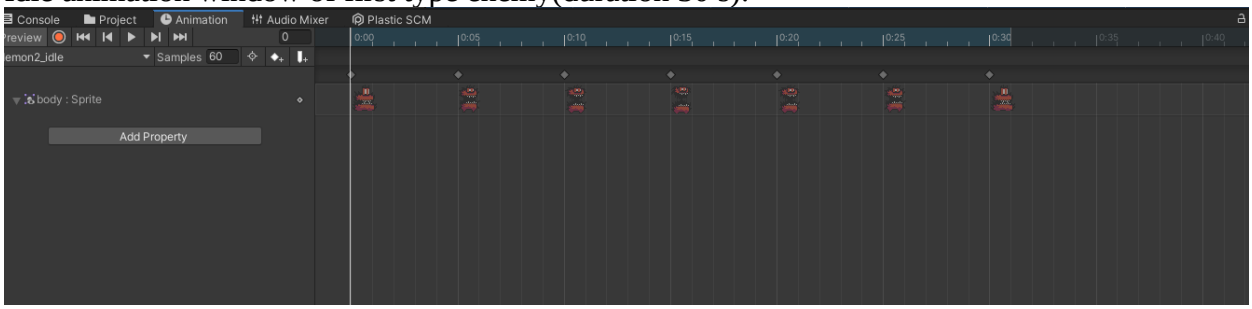

Figure 103 First enemy idle animation

Running animation window of first type enemy(duration 20s):

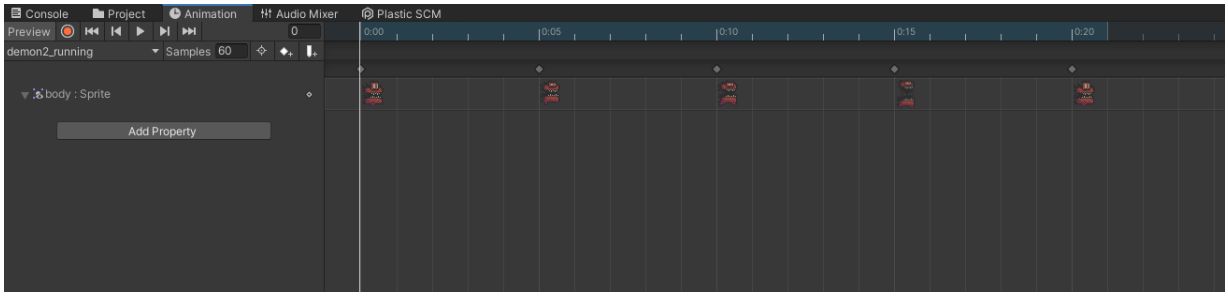

Figure 104 First enemy running animation

Idle animation window of second type enemy(duration 18):

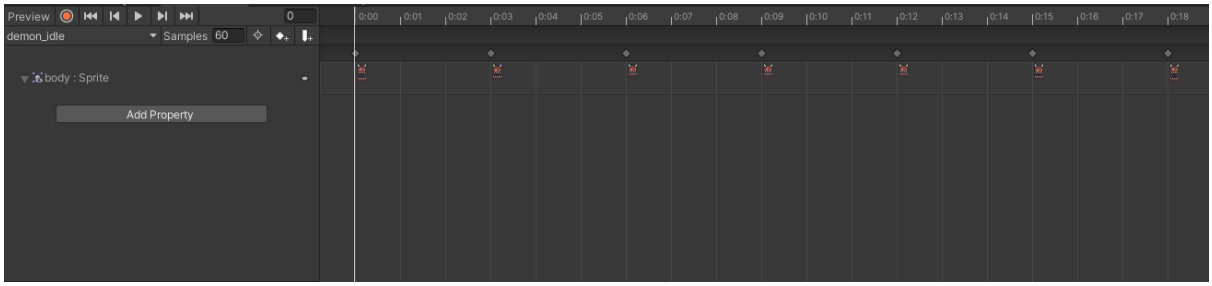

Figure 105 Second enemy idle animation

Running animation window of second type enemy(duration 24 s):

Figure 106 First enemy running animation

After animation, some effects for enemies were created: blood splatter after killing an enemy[Figure 107] and blood particle effects after successful hit[Figure 108].

Figure 107 Blood splatter in game

Figure 108 particle effect after hit

There were written two scripts: for controlling enemies(EnemyController.cs) and for their bullets(EnemyBullet.cs) .

Functions used in EnemyController.cs:

1. private void Awake() [Figure 109] (Matas Jonauskas)
2. void Start() [Figure 110] (Vidas Buivydas)
3. void Update() [Figure 111] (Matas Jonauskas)
4. private void OnTriggerEnter2D(Collider2D other) [Figure 112] (Matas Jonauskas)
5. private void OnCollisionStay2D(Collision2D other) [Figure 113] (Matas Jonauskas)

Functions used in EnemyBullet.cs:

1. void Start() [Figure 114] (Matas Jonauskas)
2. void update() [Figure 115] (Matas Jonauskas)
3. private void OnTriggerEnter2D(Collider2D other) [Figure 116] (Matas Jonauskas)
4. private void OnBecameInvisible() [Figure 117] (Matas Jonauskas)

```
private void Awake(){

    instance = this;
}
```

Figure 109 function void Awake()

This function at the start of the game gets RoomClose component from player and assigns it in script.

```
void Start()
{
    roomClose = GameObject.FindGameObjectWithTag("Player").GetComponent<RoomClose>();
}
```

Figure 110 function void Start()

Update() functions find direction in which enemies must go to attack the player. Firstly, checks if player is visible, after that calculate move direction, change enemy movement velocity, and set that enemy should look at player's position and last if statement enables or disables animation.

```
void Update()
{
    if(theBody.isVisible)
    {
        // jei bus 2 zaideju rezimas tai cia dar viena if idet kad du instance is karto ziuretu
        if(Vector3.Distance(transform.position, PlayerController.instance.transform.position) < rangeToChase){

            moveDirection = PlayerController.instance.transform.position - transform.position;
        }
        else{
            moveDirection = Vector3.zero;
        }

        moveDirection.Normalize();
        theRB.velocity = moveDirection * movementSpeed;


        // rotation
        Vector3  playerPos = PlayerController.instance.transform.position;
        Vector3  enemyPos = transform.position;

        if(playerPos.x < enemyPos.x){
            transform.localScale = new Vector3(-1f, 1f, 1f);
        }else{
            transform.localScale = new Vector3(1f, 1f, 1f);
        }

        /// shooting

        if(shouldShoot && Vector3.Distance(transform.position, PlayerController.instance.transform.position) < shootRange){

            fireCounter -= Time.deltaTime;

            if(fireCounter <= 0)
            {
                fireCounter = fireRate;
                Instantiate(bullet, firepoint.transform.position, firepoint.transform.rotation);
            }
        }
    }
    //movement animation

    if(moveDirection != Vector3.zero){

        anim.SetBool("isMoving", true);

    }else{
        anim.SetBool("isMoving", false);
    }
}
```

Figure 111 function void Update()

This function controls enemy ‘s health. If weapon hits the enemy, it ‘s health will decrease, particle effects will be instantiated. If enemy ‘s health if lower than zero, enemy will be destroyed, experience points assigned to the player, blood splatter instantiated and also there is a 50 % chance of getting health potion or mana potion or gold coin after killing an enemy.

```
private void OnTriggerEnter2D(Collider2D other){

    if(other.tag == "Weapon")
    {
        health -= Player.instance.damage;

        Instantiate(hitEffect, transform.position, transform.rotation);

        if(health < 0){
            roomClose.enemiesToKill--;
            Destroy(gameObject);
            LevelSystem.instance.LevelUpgrade(true);

            int selectedSplatter = Random.Range(3, 5);

            int rotation = Random.Range(0, 4);

            Instantiate(deathSplatters[selectedSplatter], transform.position, Quaternion.Euler(0f, 0f, rotation * 90f));

            if(Random.value < 0.5f){

                int drop = Random.Range(0, 3);
                Instantiate(deathSplatters[drop], transform.position, transform.rotation);
            }

        }
    }
}
```

Figure 112 function OnTriggerEnter2D(Collider2D other)

Player will loose health if enemy attacks him.

```
private void OnCollisionStay2D(Collision2D other){

    if(other.collider is CircleCollider2D && other.collider.tag == "Player"){

        Player.instance.TakeDamage(attackDamage);
    }
}
```

Figure 113 function OnCollisionStay2D(Collision2D other)

This function finds in which direction enemy should shoot it‘s bullets.

```
void Start()
{
    direction = PlayerController.instance.transform.position - transform.position;
    direction.Normalize();
}
```

Figure 114 function void Start()

This function changes bullet position according to the direction and given bullet speed

```
void Update()
{
    transform.position += direction * speed * Time.deltaTime;
}
```

Figure 115 function void Update()

If bullet hits the player, than the player will loose health.

```
private void OnTriggerEnter2D(Collider2D other){

    if(other.tag == "Player"){

        Player.instance.TakeDamage(projectileDamage);
    }

    Destroy(gameObject);
}
```

Figure 116 function void OnTriggerEnter2D(Collider2D other)

If enemy bullet is not visible on the screen than it will be destroyed.

```
private void OnBecameInvisible(){
    Destroy(gameObject);
}
```

Figure 117 function void OnBecameInvisible()

Task #3 Physics

Firsly, elements *RigidBody2D* were added to pot and wooden barrel elements [Figure 119]. Gravity was set to 0 and mass was added same as to arrow and fireblast prefabs [Figure 118]. When these destructibles are hit they are pushed. Script was modified to control simulation calculations of position and rotation when needed [Figure , Figure].

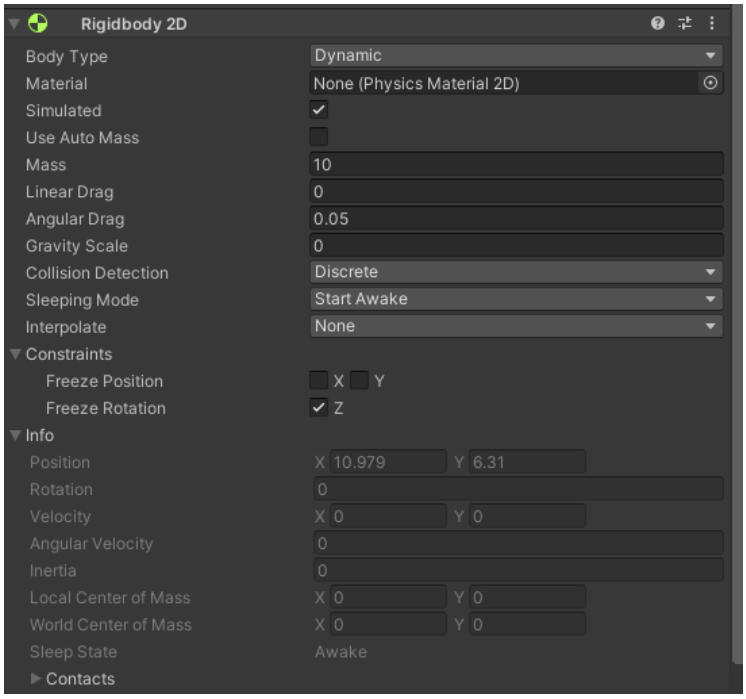

Figure 118. Fireblast and arrow Mass (identical)

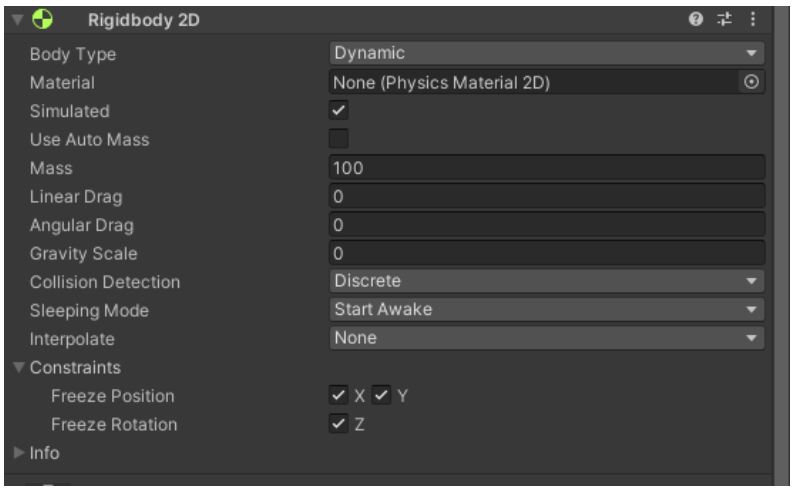

Figure 119. Pot and Wooden barrel Mass (identical)

This script controls when the calculations of simulation are made. When the object is destroyed it stops all calculations. If object is hit it starts calculations and stops after some time so object stops moving [Figure 120].

```
private void OnTriggerEnter2D(Collider2D other){

    if(other.tag == "Weapon")
    {
        health -= Player.instance.damage;
        if(health > 0){
            animator.SetBool("IsDamaged", true);


            if (GetComponent<Rigidbody2D>() != null)
            {
                Rigidbody2D rigidbody2d = GetComponent<Rigidbody2D>();
                rigidbody2d.constraints = RigidbodyConstraints2D.None;
                rigidbody2d.constraints = RigidbodyConstraints2D.FreezeRotation;
                StartCoroutine(SetConstraints());
            }
            StartCoroutine(WaitForAnimation());
        }else
        {
            if (explosionSound != null)
            {
                explosionSound.Play();
            }
            animator.SetBool("IsDestroyed", true);
            collider.enabled = false;

            if(gameObject.tag == "Barrel" && Vector3.Distance(transform.position, PlayerController.instance.transform.position) < 5)
            {
                Player.instance.TakeDamage(20);
            }
            if (GetComponent<Rigidbody2D>() != null)
            {
                Rigidbody2D rigidbody2d = GetComponent<Rigidbody2D>();
                rigidbody2d.constraints = RigidbodyConstraints2D.FreezeAll;
            }
        }
    }
}
```

Figure 120. private void OnTriggerEnter2D(Colider2D other)

This script stop simulation calculations of hit gameobject [Figure 121].

```
1 reference
private IEnumerator SetConstraints()
{
    yield return new WaitForSeconds(0.5f);

    if (GetComponent<Rigidbody2D>() != null)
    {
        Rigidbody2D rigidbody2d = GetComponent<Rigidbody2D>();
        rigidbody2d.constraints = RigidbodyConstraints2D.FreezeAll;
    }
}
```

Figure 121. private IEnumerator SetConstraits()

Task #4 Multiplayer

Firstly, we installed Mirror as our multiplayer engine. After that, we created the NetworkManager game object and assigned 4 components to it [Figure 123]. Later, we attached the Network Identity component [Figure 124] to the player Finally we created a script to follow new players.

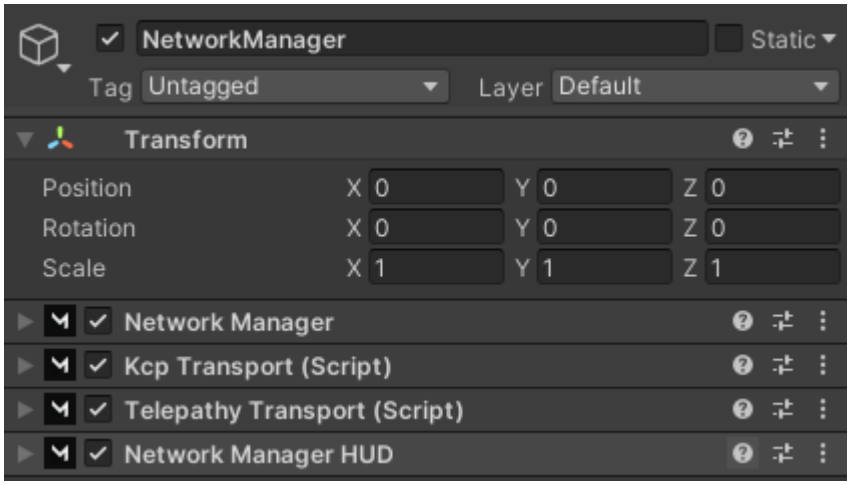

Figure 122 NetworkManager game object

Figure 123 NetworkIdentity component

Functions used in PlayerCamera.cs:

1. private void Awake() [Figure 124] (Vidas Buivydas)

   this method assigns a camera for each player

```
void Start()
{
    if (!isLocalPlayer)
    {
        camera = CameraController.FindObjectOfType<CameraController>();
        camera.target = this.gameObject.transform;
    }
}
```

Figure 124 function void Start()

User's manual

How to play?

Press the start button in the user interface and enjoy the game

Descriptions of the rules of the game.

Don’t let your character reach 0 health

Descriptions of the controls/keys.
Move to left – A
Move to right – D
Move up – W
Move down – S
Weapon direction control – Mouse location
Shoot – Left mouse button
Select player – E
Pause menu - Esc

Literature list

Resources list

1. *https://masterpose.itch.io/pixelduuuuudesmaker*
2. *https://thecrunkenstein.itch.io/roguelike-asset-pack*
3. *https://0x72.itch.io/dungeontileset-ii*
4. *https://rekkimaru.itch.io/dungeon-rpg-tileset*
5. *https://elthen.itch.io/pixel-art-destructible-objects*
6. *https://nick-noir.itch.io/pixel-art-exploding-barrel*
7. *https://assetstore.unity.com/packages/2d/gui/icons/sleek-essential-ui-pack-170650*
8. *https://freesound.org/*
9. *https://www.chosic.com/free-music/games/*