KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

T120B166 Development of Computer Games and Interactive Applications

Game: The Ones That Remain

Group, Name and Surname:
IFIN-0/2,
Erikas Venckus
Mindaugas Ruškys
Eimantas Kancevičius

Date: *2022.04.04*

Kaunas, 2022

Tables of Contents

Tables of Images

Table of Tables/functions

Work Distribution Table:

Name/Surname	***Description of game development part***
Erikas Venckus	Level design, tester, programmer, artist
Eimantas Kancevičius	Game design, tester, programmer, artist
Mindaugas Ruškys	Sound engineer, tester, producer, programmer, artist

Description of Your Game

Description of Your Game.

1. 3D or 2D?
   2.5D
2. What type is your game?
   Zombie survival, Co-Op
3. What genre is your game?
   Action RPG
4. Platforms (mobile, PC or both?)
   PC
5. Scenario Description.
   The action is set in 2030s. 5 years ago there was a massive breakthrough in health science - an injection that made every injury heal faster and regrow lost limbs, while also temporarily enhancing the user's senses and abilities. Medical trials for this injection were successful with both animals and humans - healing time from injuries, ranging from minor cuts to missing limbs and organs was decreased by 10-25 times, while enhancements were rather random for every individual. It worked by rapidly renewing and regrowing cells, but due to corruption and bribery from the company producing it, the injection was poorly tested for long term, common use. Production of this injection was very easy and cheap, that's why it didn't take long until its usage spread all around the world. About two years ago, an increasing amount of people, who often used this injection, started to mutate. They were slowly losing their mobility and senses. It didn't take long until they were no longer in control of their body. Then the visual changes started, and the sick were becoming more like enhanced, senseless and aggressive humanoid creatures, attacking every one that stood in their way and starting a war for human survival. You play as a human survivor trying to scavenge and survive in this new post-apocalyptic world. Your enemies are other humans trying to survive and various types of mutants. You have a home base witch you can upgrade and sometimes need to defend. Upon the death of a character, you play as a new survivor who found your old base. The scavenging stage is done by choosing a location on a map and being transported to that area where you can enter and loot various buildings.

Laboratory work #1

List of tasks (main functionality of your project)

1. Player character creation
2. Main Menu and Pause Menu creation
3. Music for main menu creation and implementation
4. Creation of player movement script with animations

Solution

Task #1. *Player character creation*

A human character model was downloaded, and a human rig was created using blender. Then textures were created and export as a baked texture images of diffuse, roughness and ambient occlusion. Textures of diffuse and roughness were combined, and everything was combined into a texture material in Unity. Then an avatar was created in unity and basic movement animations were downloaded and implemented.

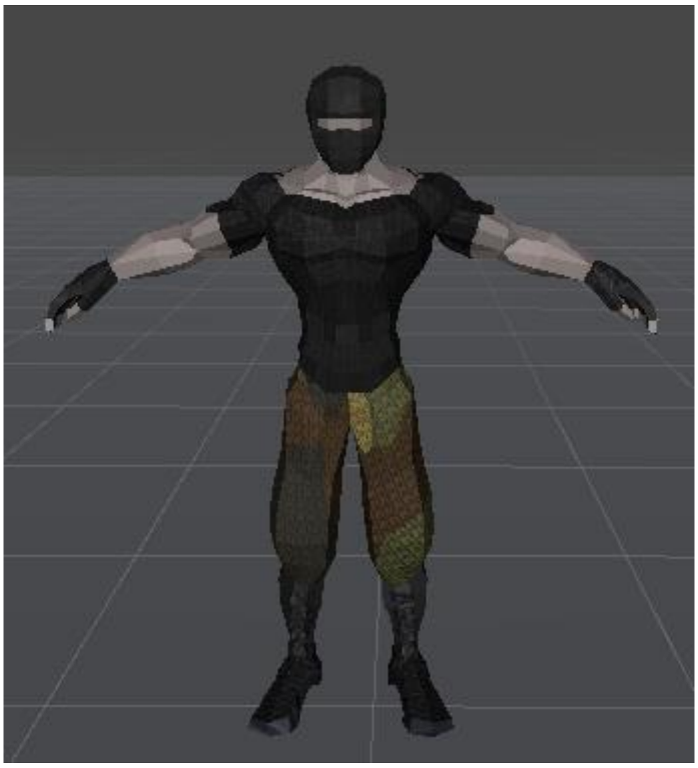


Figure 1. Player character

Task #2. *Main Menu and Pause Menu creation*

Main Menu was created containing buttons for different functions such as New Game, Load Game, Options and Quit basic functionality to load game scene and exit to desktop was implemented. Pause Menu was created containing 4 buttons Resume, Save Game, Options, Quit to Main Menu. The Resume button has the expected functionality to close the Pause Menu also the UI can be triggered by pressing ESC key. Quit to Main Menu also has working functionality which unloads the game scene and loads the Main Menu instead.

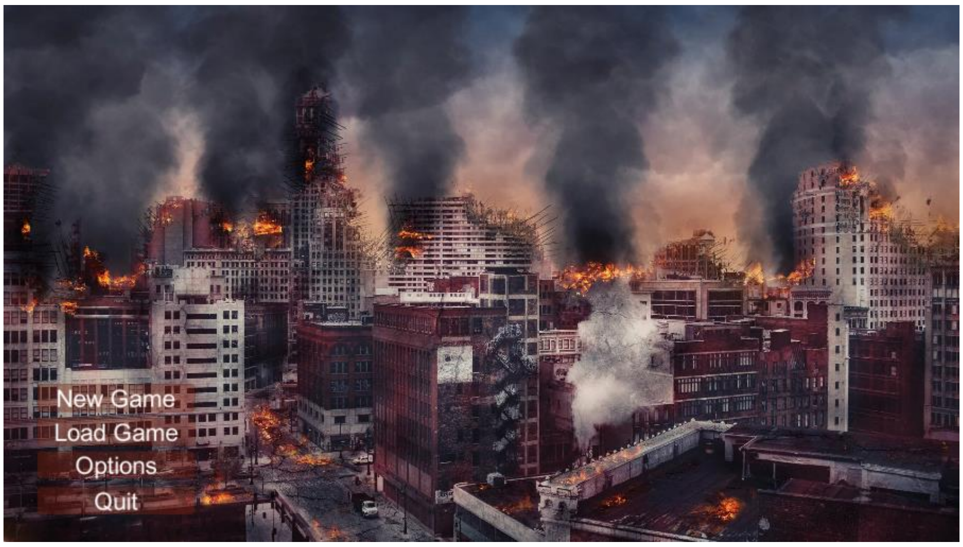


Figure 2. Main Menu UI

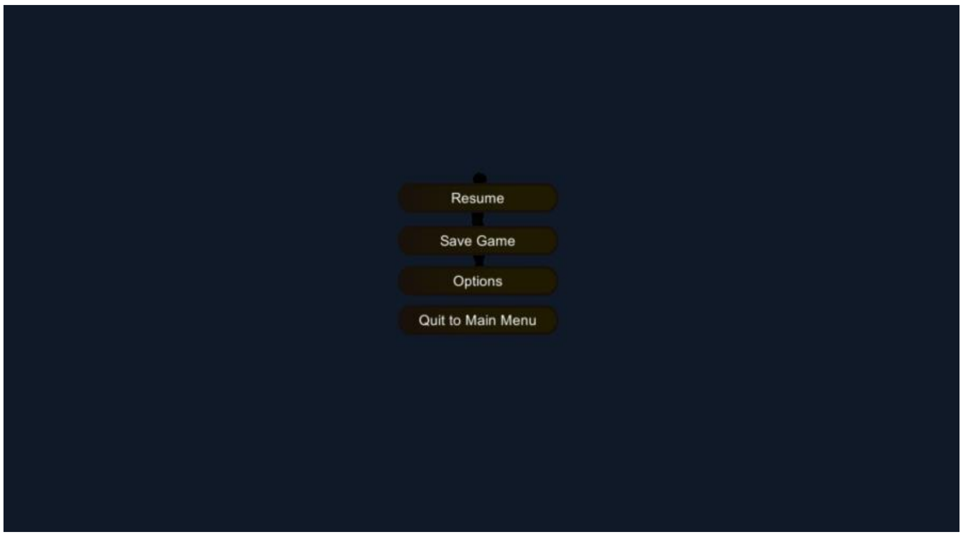


Figure 3. Pause Menu UI

```
public class SceneLoader : MonoBehaviour
{
    public void LoadScene(string scenename)
    {
        SceneManager.LoadScene(scenename);
    }
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

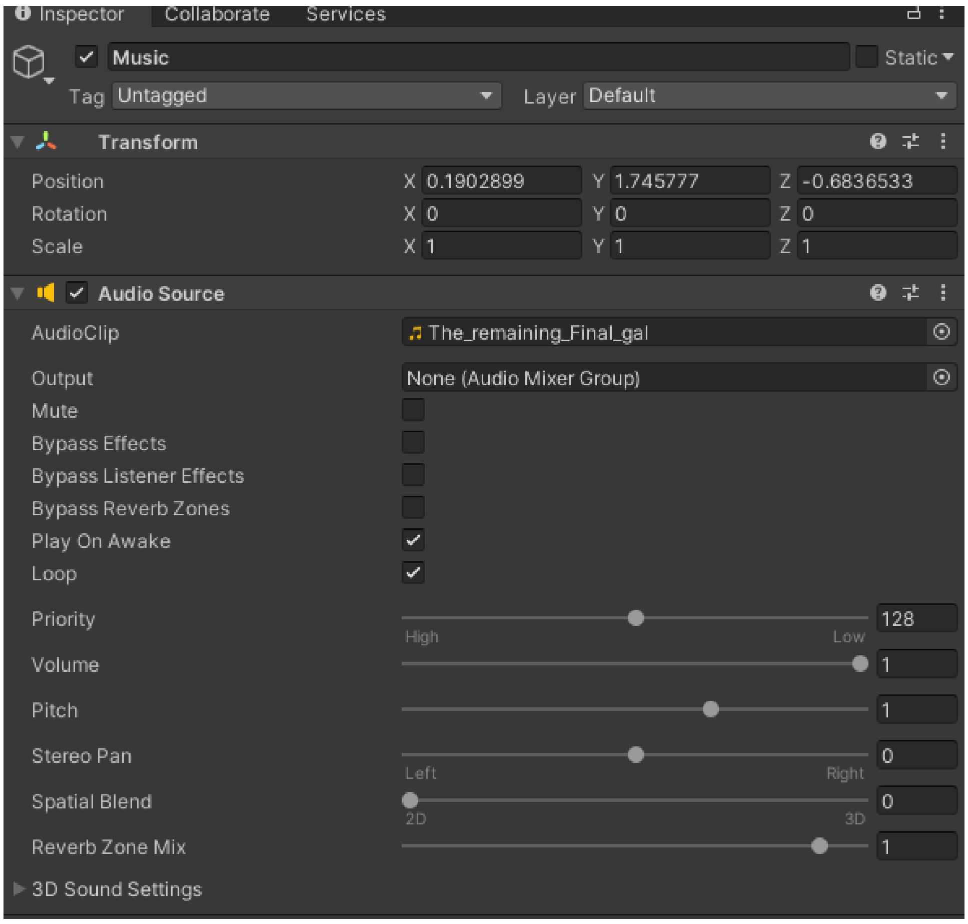
Table 1. SceneLoader script to load scenes

```
public class PauseMenu : MonoBehaviour
{
    [SerializeField]
    private GameObject pause_UI;

    bool is_paused = false;
    bool status_change = true;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            is_paused = !is_paused;
            status_change = true;
        }

        if(status_change)
        {
            if(is_paused)
            {
                Time.timeScale = 0;
                pause_UI.SetActive(true);
                status_change = false;
            }
            else
            {
                Time.timeScale = 1;
                pause_UI.SetActive(false);
                status_change = false;
            }
        }
    }
    /// <summary>
    /// Change game state to unpaused
    /// </summary>
    public void resume_btn_click_fn()
    {
        is_paused = false;
        status_change = true;
    }
}
```

Table 2. Pause Menu script

Task #3. *Music for main menu creation and implementation*

Music for the main menu was created and recorded. Some editing was done to reduce background tones and reverb was added. Then an empty object was created in main menu scene for the music to play in the scene.


Inspector Collaborate Services

☑ Music ☐ Static

Tag Untagged Layer Default

Transform

Position	X 0.1902899	Y 1.745777	Z -0.6836533
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

☑ Audio Source

AudioClip	The_remaining_Final_gal
Output	None (Audio Mixer Group)
Mute	☐
Bypass Effects	☐
Bypass Listener Effects	☐
Bypass Reverb Zones	☐
Play On Awake	✓
Loop	✓
Priority (High – Low)	128
Volume	1
Pitch	1
Stereo Pan (Left – Right)	0
Spatial Blend (2D – 3D)	0
Reverb Zone Mix	1

3D Sound Settings

Figure 4. Music empty object

Task #4. Player movement with animations

First, all five of our animations had to be connected to the Animator, using transitions to make animations transition between themselves, when certain conditions, which are specified in the properties tab, are met. Secondly, to make our character react to these animations, a script called AnimationStateController was created. In short, this script reacts to movement and makes the character perform animation from the input that's being received. Lastly, a movement script was created to move character in the scene, with the animations character reacts to, when movement is active.

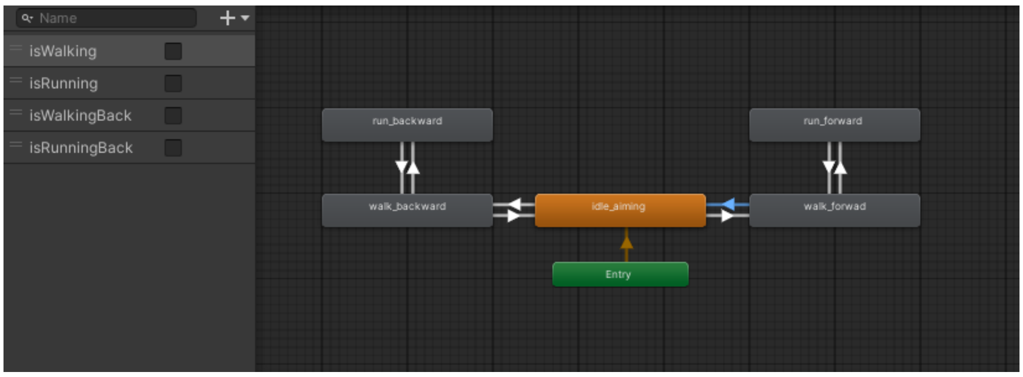


Figure 5. Human Controller (Animator Controller)

```
        //running
        if (!isRunning && (forwardPressed && runPressed))
        {
            anim.SetBool(isRunningHash, true);
        }
        if (isRunning && (!forwardPressed || !runPressed))
        {
            anim.SetBool(isRunningHash, false);
        }

        //walking backwards
        if(!isWalkingBack && backwardPressed)
        {
            anim.SetBool(isWalkingBackHash, true);
        }

        if(isWalkingBack && !backwardPressed)
        {
            anim.SetBool(isWalkingBackHash, false);
                                                                   }
```

Table 3. AnimationStateController script for changing animations

```
private void UpdatePosition()
    {
        var positionMovement = transform.forward *
            (movementAxis.x * moveSpeed * Time.deltaTime);

        var currentPosition = rb.position;
        var newPosition = currentPosition + positionMovement;

        rb.MovePosition(newPosition);
                                                     }
```

Table 4. Player Controller script for changing positions

Laboratory task

Make an enemy like cube that loses health on collision with the player.

```
public class Enemy : MonoBehaviour
{
    [Header("Player state")]
    [SerializeField, Min(0)]
    private int lives = 3;

    [Header("UI")]
    [SerializeField]
    private Text livesText;

    private void Start()
    {
        UpdateLivesText();
    }

    private void UpdateLivesText()
    {
        livesText.text = $"Lives: {lives}";
    }

    public void TakeDamage()
    {
        lives--;
        UpdateLivesText();

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

Table 5. Enemy cube script

```
public class Colide : MonoBehaviour
{
    [SerializeField]
    private string playerTag = "Player";

    [SerializeField, Min(0f)]
    private float minExplosionForce = 1f;

    [SerializeField, Min(0f)]
    private float maxExplosionForce = 1f;

    private Rigidbody rb;
    private Enemy enemy;

    private void Awake()
    {
        rb = GetComponent<Rigidbody>();
        enemy = GetComponent<Enemy>();
    }

    private void OnCollisionEnter(Collision other)
    {
        var otherGameObject = other.gameObject;
        if (IsPlayer(otherGameObject))
        {
            AddExplosionForce();
            enemy.TakeDamage();
        }
    }

    private bool IsPlayer(GameObject obj)
    {
        return obj.CompareTag(playerTag);
    }

    private void AddExplosionForce()
    {
    var force = Random.Range(minExplosionForce, maxExplosionForce);
    rb.AddForce(Vector3.up * force, ForceMode.Impulse);
    }
}
```

Table 6. Collision script

Laboratory work #2

List of tasks (main functionality of your project)

1. Character animations improvements
2. Aiming and shooting
3. First level outside zone creation
4. Major UI improvements
5. Enemy creation
6. Creation of health, armor characteristics
7. Ragdoll implementation
8. Player and weapon rotation using IK constraints
9. Camera improvements

Solution

Task #1. Character animations improvements

After implementation of the animations, we noticed that animations didn't transition between each other like we wanted, so instead of using bools to change animation states, we started to use a float to increase velocity and make animations change according to the current velocity. Every animation was added to a blend tree. Also, there is acceleration and deceleration to the animations, so player doesn't just stop when movement key is no longer pressed. The result of these improvements is quite satisfying because animation transition became smoother, and animations got sort of a timer to make animations move faster when movement speed is higher.

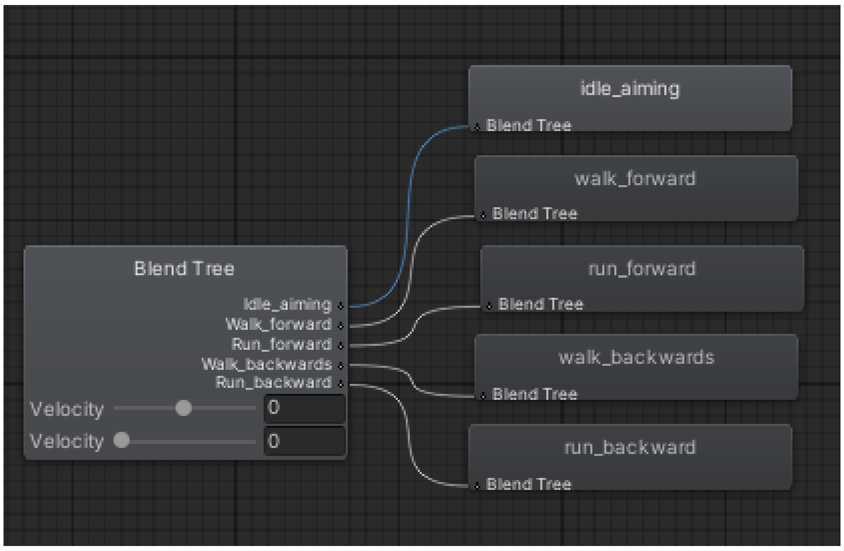


Figure 6. New animation blend tree

Task #2. Aiming and shooting

One of the most important tasks was to make character aim at cursor. We wanted aiming to include both head and weapon looking at the target. Once this complicated process was finished, we proceeded to make character able to shoot. For this task, we made a bullet prefab with its own Bullet script, and every time the fire button is clicked, a bullet would shoot out of guns muzzle at a direction where weapon is aiming. Of course, You can only shoot a certain number of shots because ammo was made limited.

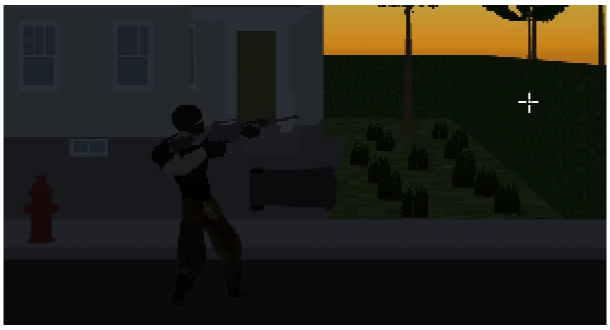


Figure 7. Player weapon following the cursor

```
    public void Fire(Vector3 position, Vector3 euler, int layer)
    {
        lifeTimer = Time.time;
        transform.position = position;
        transform.eulerAngles = euler;
        transform.position = new Vector3(transform.position.x, transform.position.y,
0);
        Vector3 vop = Vector3.ProjectOnPlane(transform.forward, Vector3.forward);
        transform.forward = vop;
        transform.rotation = Quaternion.LookRotation(vop, Vector3.forward);
    }
```

Table 5. Fragments of the shooting script

```
private void Hit(Vector3 position, Vector3 direction, Vector3 reflected, Collider
collider)
    {
        Debug.Log(collider.name);
        if(collider.tag == "Enemy")
        {
            collider.GetComponent<MobHealth>().TakeDamage(damage);
        }
        Destroy(gameObject);

    }
```

Table 6. Fragments of the bullet script

Task #3. First level outside zone creation

Using CITY package assets, we built an outside are of the suburban level zone including a few houses, gas station and a shop.

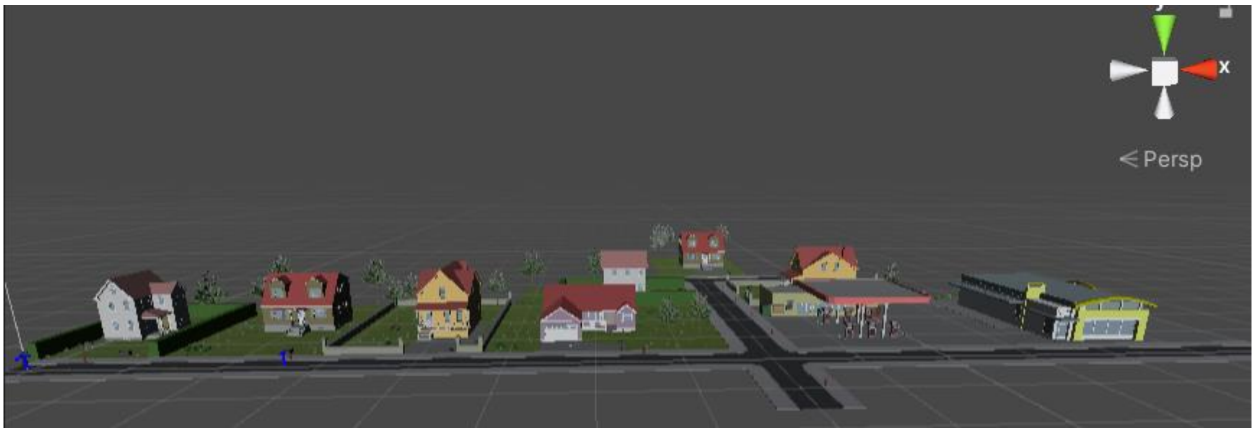


Figure 8. Suburban level

Task #4. Major UI improvements

Main menu and Pause menu button styles have been changed to match style everywhere. A simple options menu has been added to change simple settings such as audio level, resolution, and game window modes. A menu to select the level has been added after which player is presented with the main controls of the game. A HUD with essential information for gameplay was also created containing armor, health, ammo, and selected weapon information. A game over screen was created that is displayed when player's health hits 0. A crosshair has been added to help with aiming. Camera now moves to left and right based on mouse location.

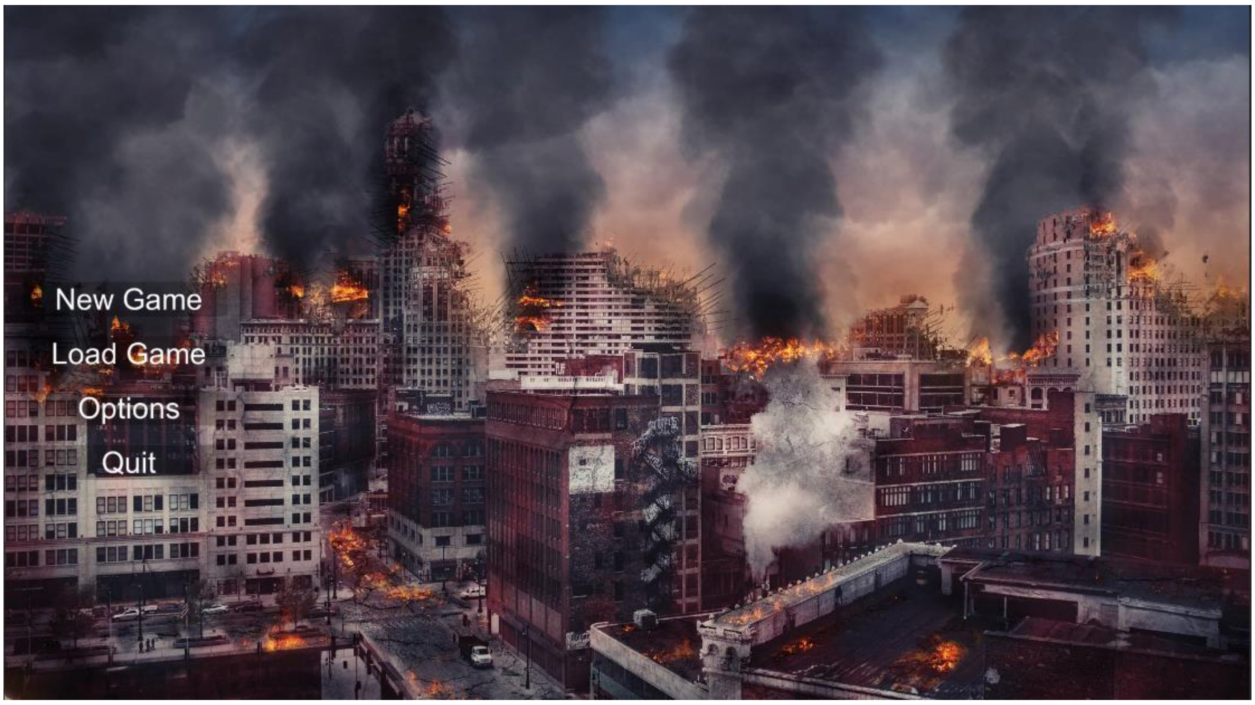


Figure 9. Main menu

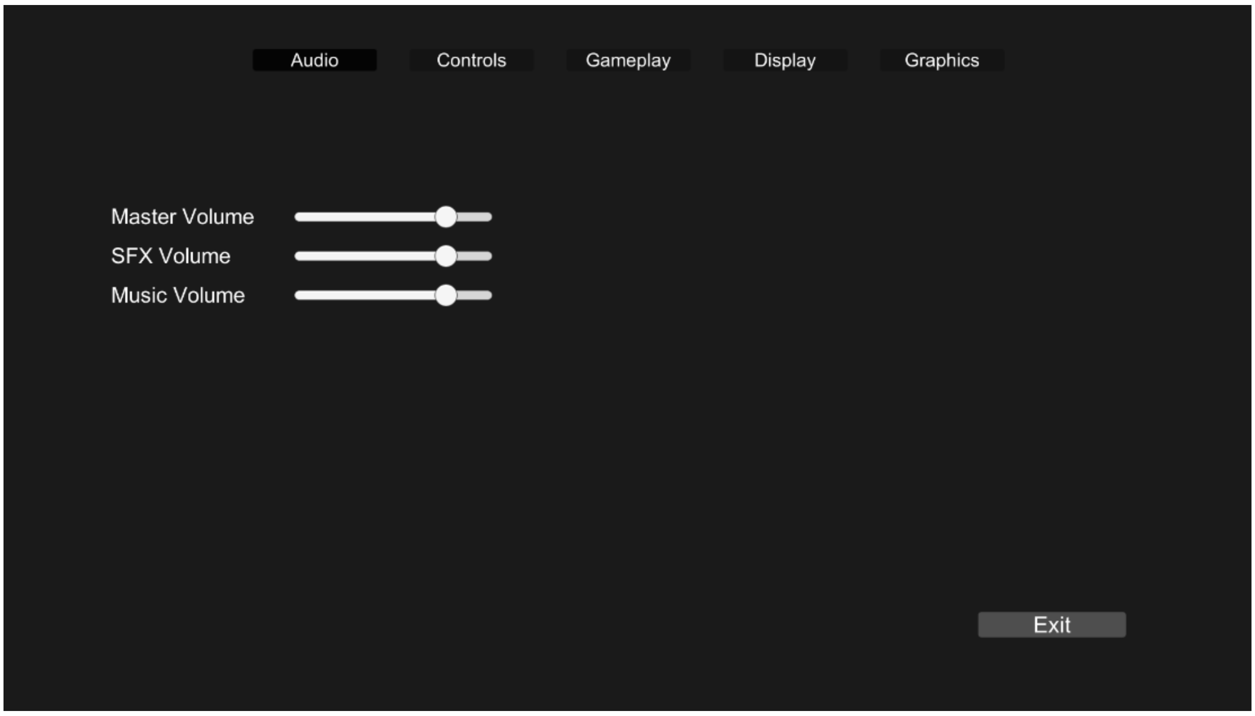
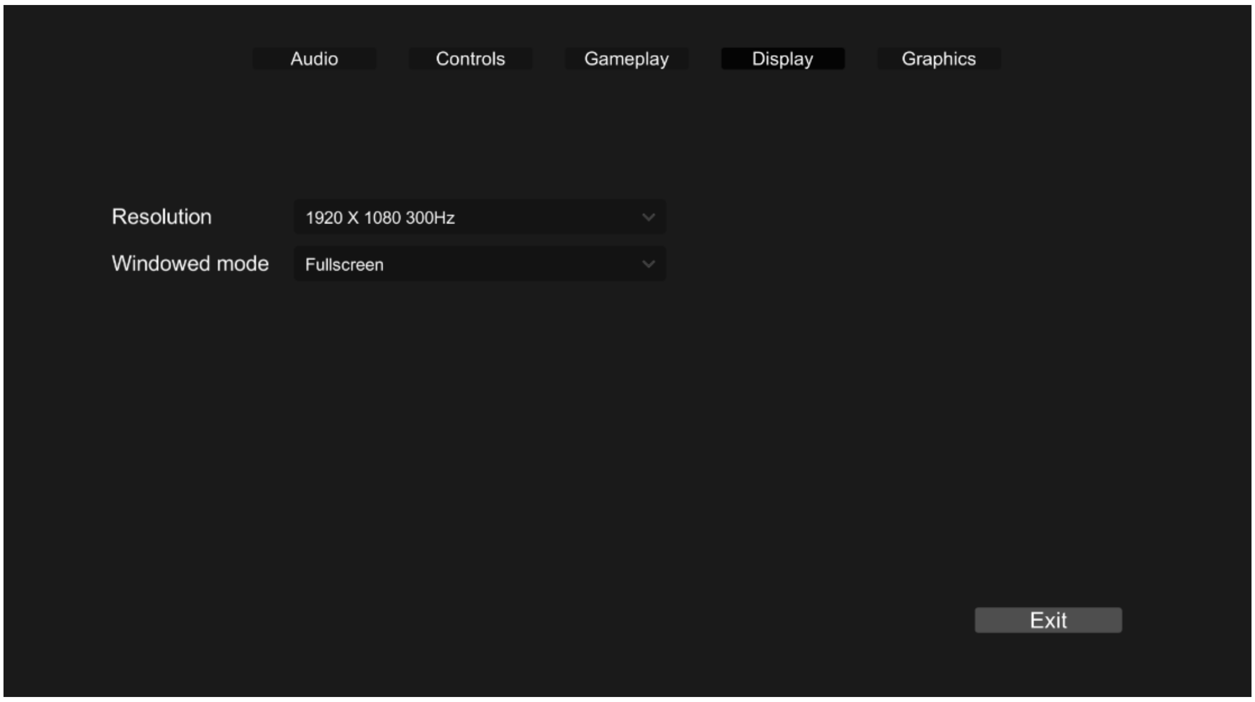


Figure 10. Sound option menu



Resolution 1920 X 1080 300Hz

Windowed mode Fullscreen

Figure 11. Display option menu

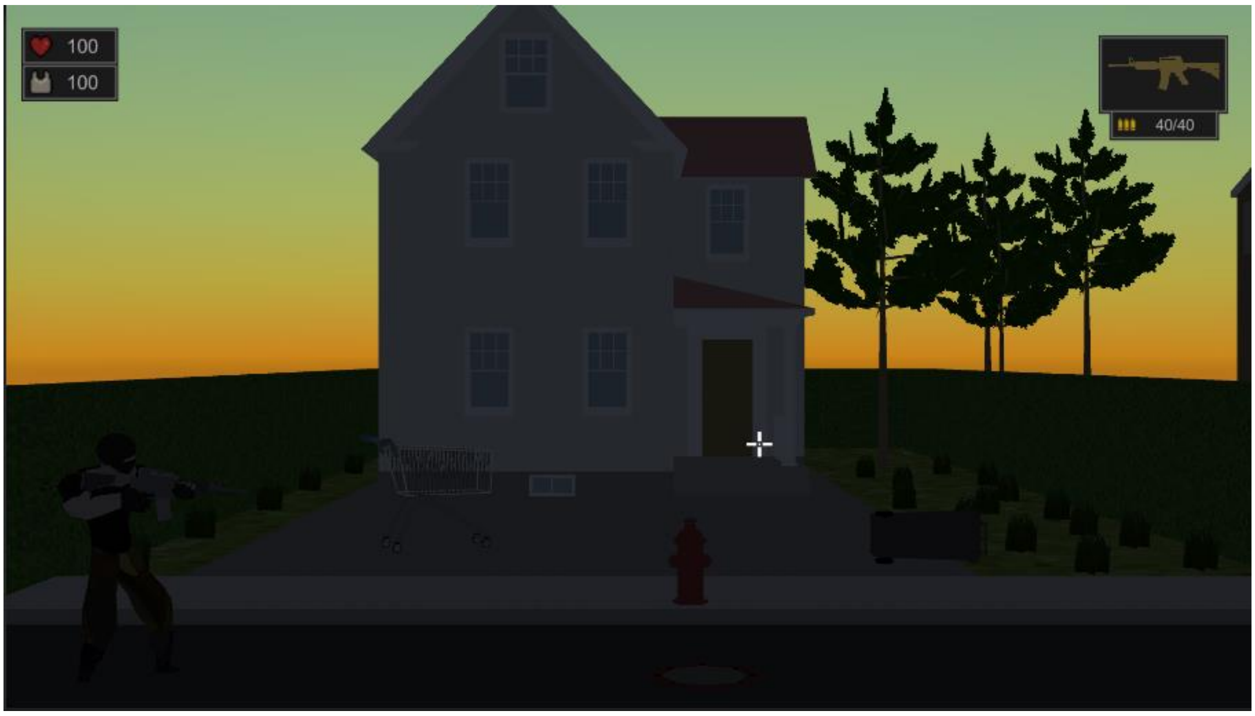


Figure 12. Health, armor, and weapon UI

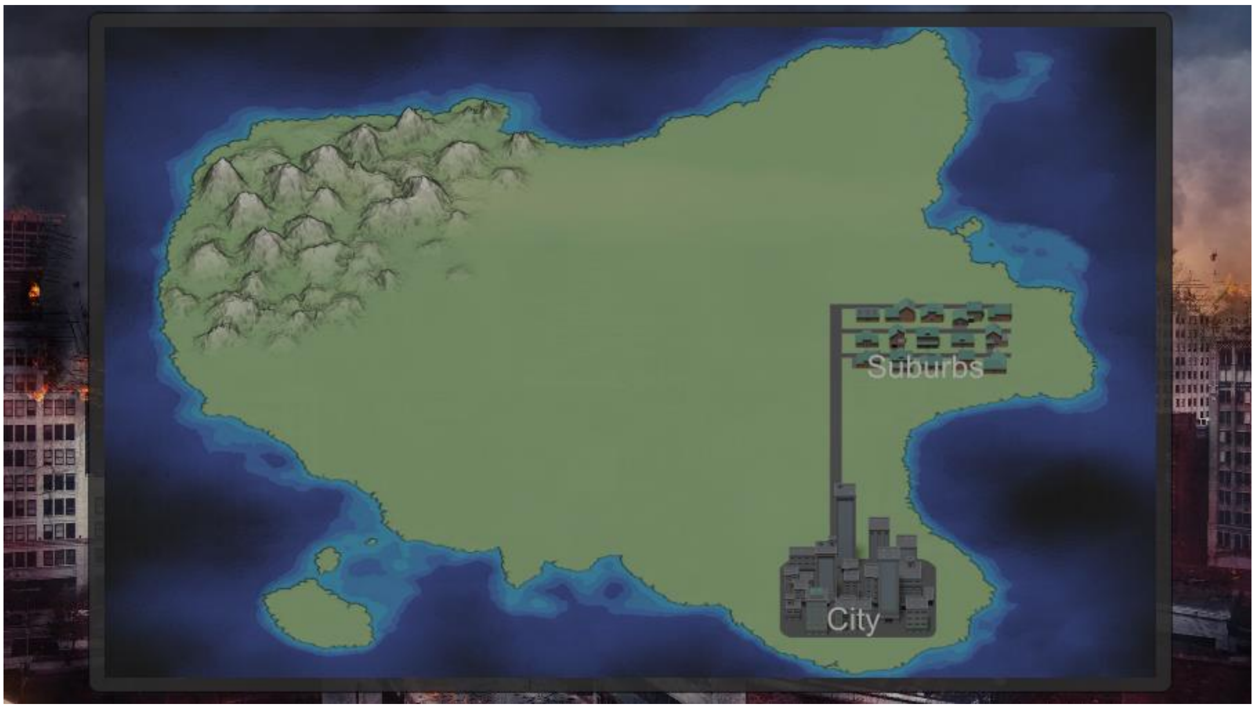


Figure 13. Map selection screen

```
// Audio
    public AudioMixer masterMixer;

    public void SetMasterVolume (float MasterVolume)
    {
        masterMixer.SetFloat("MasterVolume", MasterVolume);
    }

    public void SetSFXVolume(float SFXVolume)
    {
        masterMixer.SetFloat("SFXVolume", SFXVolume);
    }

    public void SetMusicVolume(float MusicVolume)
    {
        masterMixer.SetFloat("MusicVolume", MusicVolume);
    }
    private void UpdateSliders()
    {
        float value;
        masterMixer.GetFloat("MasterVolume", out value);
        masterSlider.value = value;

        masterMixer.GetFloat("SFXVolume", out value);
        sfxSlider.value = value;

        masterMixer.GetFloat("MusicVolume", out value);
        musicSlider.value = value;
    }

    // Display
    public static FullScreenMode fullScreenMode;
    public void SetScreenMode(int screenmodeIndex)
    {
        if(screenmodeIndex == 0)
        {
            Screen.fullScreenMode = FullScreenMode.ExclusiveFullScreen;
        }
        else if(screenmodeIndex == 1)
        {
            Screen.fullScreenMode = FullScreenMode.Windowed;
        }
        else if (screenmodeIndex == 2)
        {
            Screen.fullScreenMode = FullScreenMode.FullScreenWindow;
        }
    }

    public Slider masterSlider;
    public Slider sfxSlider;
    public Slider musicSlider;

    public Dropdown resolutionDropdown;

    Resolution[] resolutions;

    void Start()
    {
        UpdateSliders();

        resolutions = Screen.resolutions;

        resolutionDropdown.ClearOptions();

        List<string> options = new List<string>();

        int currentResolutionIndex = 0;
        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + " X " + resolutions[i].height;

            if (resolutions[i].refreshRate != 60)
            {
                option = resolutions[i].width + " X " + resolutions[i].height + " " 
+ resolutions[i].refreshRate + "Hz";
            }

            if (CheckRefreshRate(resolutions[i].refreshRate))
            {
                options.Add(option);
            }

            if(resolutions[i].width == Screen.currentResolution.width && 
resolutions[i].height == Screen.currentResolution.height)
            {
                currentResolutionIndex = i;
            }
        }

        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();
    }
```

Table 7. Fragment of script for options menu

```
        // Updates texts.
        armorText.text = $"{(int)armor}";
        healthText.text = $"{(int)health}";
```

Table 8. Fragment of script for health and armor containing HUD display part

Task #5. Enemy creation

Using assets from “Polygon Zombies with Animations Free Pack”, we implemented a basic enemy with an AI. Using state machines and NavMesh agent we made the enemy follow a basic patrol route and react to the player when he is close or firing at the enemy. The enemy also has a health bar and turn in to a ragdoll if he dies. He can also attack and do damage to the player.

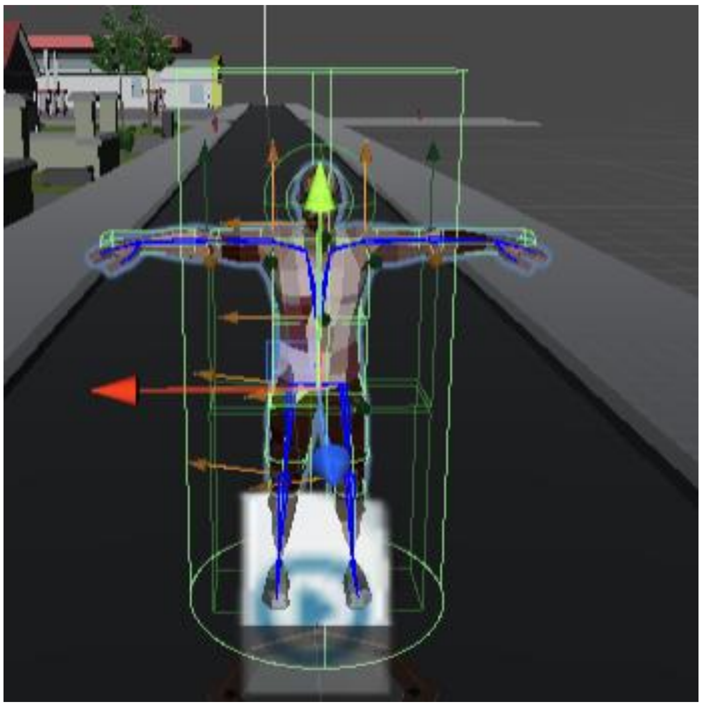


Figure 14. Enemy model with colliders

```
public class MobHealth : MonoBehaviour
{
    [Min(1)]
    public float HP = 100;
    private float currentHP;
    private float timer = 0f;
    private Collider[] hitColliders;
    private Animator stateController;

    // Start is called before the first frame update
    void Start()
    {

        currentHP = HP;
        hitColliders = GetComponents<Collider>();
        stateController = GetComponent<Animator>();
        Invoke("ActivateCol", 1);


    }

    private void ActivateCol()
    {
        foreach (Collider col in hitColliders)
        {
            col.enabled = true;
        }
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void TakeDamage(float damage)
    {
        stateController.SetBool("IsChasing", true);
        currentHP -= damage;

        if(currentHP < 0) Die();

    }

    private void Die()
    {
        foreach (Collider col in GetComponents<Collider>())
        {
            col.enabled = false;
        }
        GetComponent<MutantRagdollDeath>().Die();
    }
}
```

Table 9. Enemy health script

Task #6. Creation of health and armor characteristics

Player has health and armor that deplete if he gets attacked. Armor blocks 80% of the damage. If health reaches zero, the player dies and turns into a ragdoll and a death UI appears.

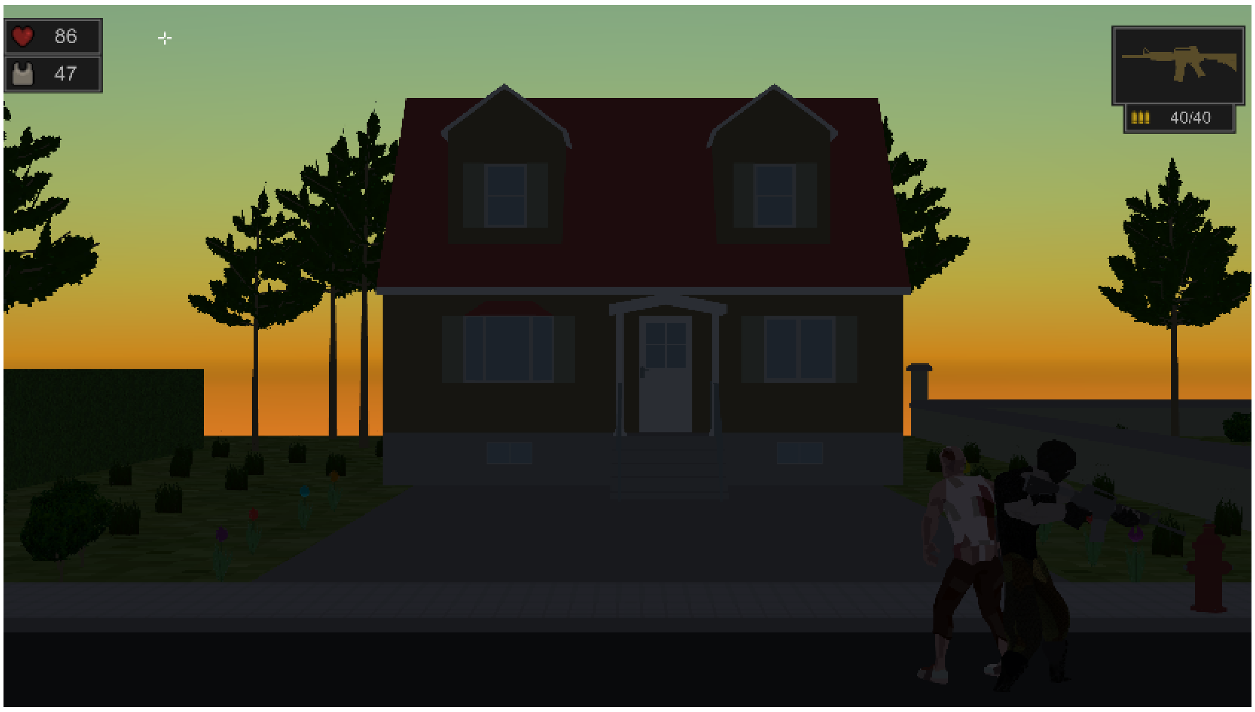


Figure 15. Enemy attacking the player

```
public class HealthAndArmor : MonoBehaviour
{
    // Armor Text
    [SerializeField]
    private Text armorText;

    //Health Text
    [SerializeField]
    private Text healthText;

    //Health Value
    private float health = 100;

    //Armor Value
    private float armor = 100;

    // Game Over Script
    private GameOver gOver;

    /// <summary>
    /// Gets the GameOver script to end the game if the payer dies.
    /// </summary>
    private void Start()
    {
        gOver = GetComponent<GameOver>();
    }

    /// <summary>
    /// Takes dammage based on amount of armor you have and ends the game if you have
no health left.
    /// </summary>
    /// <param name="damage">Amount of dammage given</param>
    public void TakeDamage(float damage)
    {
        // Calculates dammage based on armor/
        if(armor >= damage *0.8)
        {
            armor -= (damage * 0.8f);
            health -= (damage * 0.2f);
        }
        else
        {
            damage -= armor;
            armor = 0;
            health -= damage;
        }

        // Updates texts.
        armorText.text = $"{(int)armor}";
        healthText.text = $"{(int)health}";

        // Ends the game if the player has no health left.
        if(health <= 0)
        {
            gOver.Over();
        }
    }

    /// <summary>
    /// Adds health to player.
    /// </summary>
    /// <param name="lives">Amount of health to add</param>
    private void AddHealth(float lives)
    {
        health += lives;
        healthText.text = $"{(int)health}";
    }
}
```

Table 10. Player health and armour script

Task #7. Ragdoll implementation

Player and enemies turn into ragdolls when they die. This is done through adding rigidbodys and colliders to all the bones.

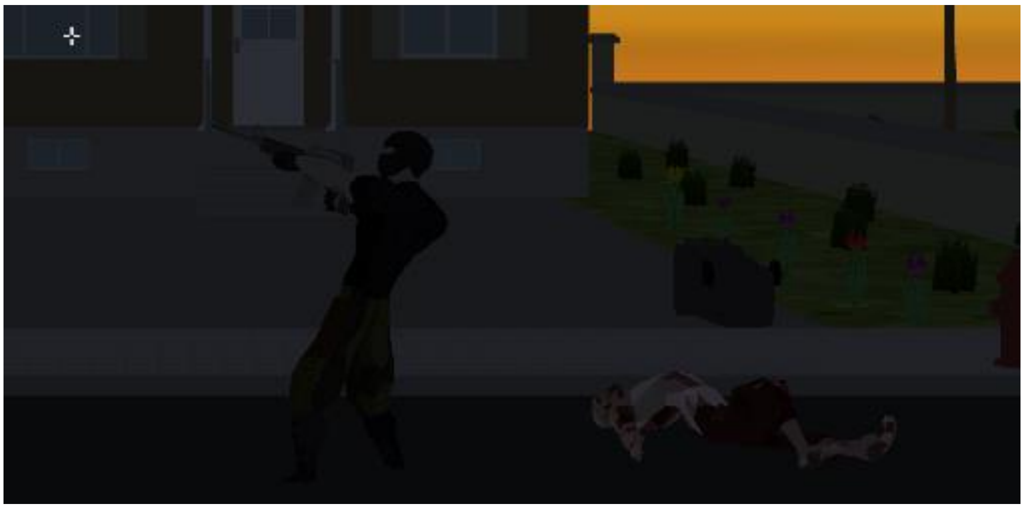


Figure 16. Enemy fell turned into a ragdoll after being killed

```
public class RagdollDeathScript : MonoBehaviour
{
    // Player animator.
    private Animator animator;

    // Players gun.
    private GameObject gun;

    // Guns Rigidbody
    private Rigidbody gunsRgb;

    // Guns Collider.
    private Collider gunsCol;

    // Ragdolls RigidBody array.
    private Rigidbody[] ragdollBodies;

    // Ragdolls colider array.
    private Collider[] ragdollColliders;

    /// <summary>
    /// Gets all the components that are used when activating the ragdoll
    /// </summary>
    void Start()
    {
        animator = GetComponent<Animator>();
        ragdollBodies = GetComponentsInChildren<Rigidbody>();
        ragdollColliders = GetComponentsInChildren<Collider>();

        ToggleRagdoll(false);
    }

    /// <summary>
    /// Players ragdoll activates.
    /// </summary>
    public void Die()
    {
        ToggleRagdoll(true);
    }
    /// <summary>
    /// Toggles ragdoll on and off;
    /// </summary>
    /// <param name="state">State of the ragdoll. True = on, false = off</param>
    private void ToggleRagdoll(bool state)
    {
        //Gets the current gun held by the player and its components.
        gun = GameObject.FindGameObjectWithTag("Gun");
        gunsRgb = gun.GetComponent<Rigidbody>();
        gunsCol = gun.GetComponent<Collider>();

        animator.enabled = !state;

        // Enables gun physics and collider.
        gunsRgb.isKinematic = !state;
        gunsRgb.useGravity = state;
        gunsCol.enabled = state;


        // Activates Rigidbody of all bones
        foreach (Rigidbody rb in ragdollBodies)
        {
            rb.isKinematic = !state;
        }

        // Activates colliders of all bones.
        foreach (Collider collider in ragdollColliders)
        {
            collider.enabled = state;
        }
    }
}
```

Table 11. Script to activate the ragdoll

Task #8. Player and weapon rotation using IK constraints

Using IK constraints, we were able to make the players body and the weapon rotate more fluidly based on the position of the cursor. Player's spine, head and arms move according to the position of the cursor and the weapon.

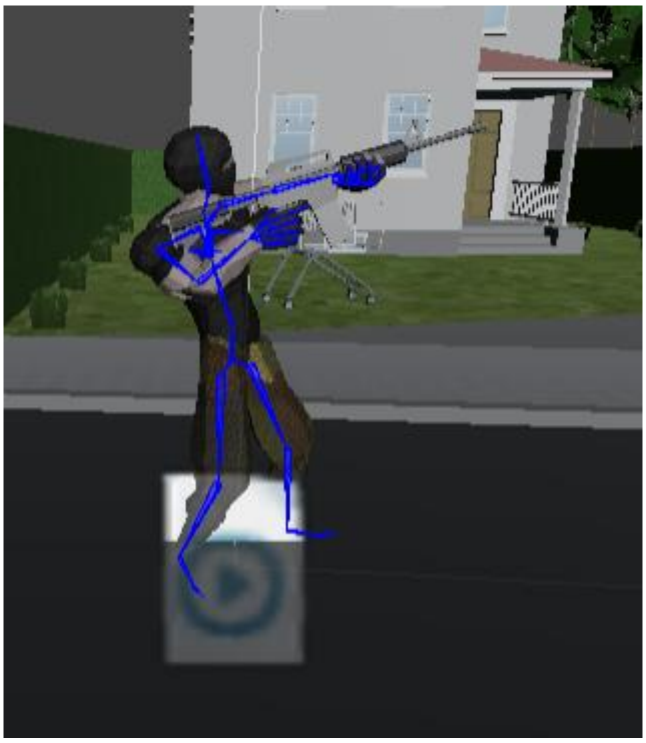


Figure 17. Player weapon following the cursor

```
    private void OnAnimatorIK(int layerIndex)
    {
        anim.SetIKPositionWeight(AvatarIKGoal.RightHand, rightHandWeight);
        anim.SetIKPositionWeight(AvatarIKGoal.LeftHand, leftHandWeight);

        anim.SetIKPosition(AvatarIKGoal.RightHand, rightHandTarget.position);
        anim.SetIKPosition(AvatarIKGoal.LeftHand, leftHandTarget.position);

        anim.SetIKRotationWeight(AvatarIKGoal.RightHand, rightHandWeight);
        anim.SetIKRotationWeight(AvatarIKGoal.LeftHand, leftHandWeight);

        anim.SetIKRotation(AvatarIKGoal.RightHand, rightHandTarget.rotation);
        anim.SetIKRotation(AvatarIKGoal.LeftHand, leftHandTarget.rotation);

        anim.SetIKHintPositionWeight(AvatarIKHint.RightElbow, rightHandWeight);
        anim.SetIKHintPositionWeight(AvatarIKHint.LeftElbow, leftHandWeight);

        anim.SetIKHintPosition(AvatarIKHint.RightElbow, rightElbowTarget.position);
        anim.SetIKHintPosition(AvatarIKHint.LeftElbow, leftElbowTarget.position);

        this.lookPos = pl.GetLookPosition();
        lookPos.z = transform.position.z;

        float distanceFromPlayer = Vector3.Distance(lookPos, transform.position);

        if (distanceFromPlayer > updateLookPosThreshold || distanceFromPlayer <
updateLookPosThreshold)
        {
            targetPos = lookPos;
        }

        IK_lookPos = Vector3.Lerp(IK_lookPos, targetPos, Time.deltaTime * lerpRate);

        anim.SetLookAtWeight(lookWeight, bodyWeight, headWeight, headWeight,
clampWeight);
        anim.SetLookAtPosition(IK_lookPos);

                                                }}
```

Table 12. IK controller script

Task #9. Camera improvements

Camera now follows the cursor but has two limits: the distance it can be away from the player and level constraints so it wouldn't be able to move outside the level.

```
public class CameraScript : MonoBehaviour
{
    private Transform player;

    //private Vector3 offset;

    [SerializeField]
    private float speed = 0.15f;

    [SerializeField]
    private float movementrange = 9.5f;

    public float xMin;
    public float xMax;
    private void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }
    Vector3 lastmouse = new Vector3(0, 0, 0);
    private void LateUpdate()
    {
        Vector3 mouse = new Vector3(0,0,0);

        if ((Input.mousePosition.x <= Screen.width && Input.mousePosition.x > 0) &&
(Input.mousePosition.y <= Screen.height && Input.mousePosition.y > 0))
        {
            mouse = new Vector3(Input.mousePosition.x - (Screen.width / 2f), 0, 0);
        }
        else
        {
            mouse = lastmouse;
        }

        Vector3 offset = new Vector3(player.position.x, 2.3f, -10);
        Vector3 desiredPosition = offset + mouse / (Screen.width / movementrange);

        if (mouse != Vector3.zero)
        {
            lastmouse = mouse;
        }

        desiredPosition.x = Mathf.Clamp(desiredPosition.x, xMin, xMax);
        transform.position = Vector3.Lerp(transform.position, desiredPosition, speed);
    }
}
```

Table 13. Camera controller script

Laboratory task

Create a sneaking mechanic, where enemy detection distance is lowered, when player is sneaking.

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

public class idleState : StateMachineBehaviour
{
    float timer;
    Transform playerTransform;
    PlayerController pc;
    float chaseRange = 5;

    // OnStateEnter is called when a transition starts and the state machine starts to
evaluate this state
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo,
int layerIndex)
    {
        timer = 0;
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        playerTransform = player.transform;
        pc = player.GetComponent<PlayerController>();
    }

    // OnStateUpdate is called on each Update frame between OnStateEnter and
OnStateExit callbacks
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo,
int layerIndex)
    {
        timer += Time.deltaTime;
        if(timer > 5)
        {
            animator.SetBool("IsPatrolling", true);
        }

        float cR;

        // New feature
        if (pc.isSneaking)
        {
            cR = chaseRange * 0.3f;
        }
        else cR = chaseRange;
        //
        float distance = Vector3.Distance(playerTransform.position,
animator.transform.position);
        if (distance < cR)
        {
            animator.SetBool("IsChasing", true);
        }

    }
}
```

Table 14. Improved enemy AI idle state

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

public class patrolState : StateMachineBehaviour
{
    float timer;
    List<Transform> wayPoints = new List<Transform>();
    NavMeshAgent agent;
    float chaseRange = 5;
    Transform playerTransform;
    PlayerController pc;

    float patrollSpeed = 1f;
    float PatrollAccelaration = 0.25f;

    // OnStateEnter is called when a transition starts and the state machine starts to
evaluate this state
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo,
int layerIndex)
    {
        GameObject player = GameObject.FindGameObjectWithTag("Player");
        playerTransform = player.transform;
        pc = player.GetComponent<PlayerController>();
        agent = animator.GetComponent<NavMeshAgent>();

        timer = 0;

        agent.speed = patrollSpeed;
        agent.acceleration = PatrollAccelaration;

        GameObject go = GameObject.FindGameObjectWithTag("WayPoints");
        foreach(Transform t in go.transform)
        {
            wayPoints.Add(t);
        }

        Vector3 wayPoint = wayPoints[Random.Range(0, wayPoints.Count)].position;
        animator.transform.LookAt(wayPoint);
        agent.SetDestination(wayPoint);
    }

    // OnStateUpdate is called on each Update frame between OnStateEnter and
OnStateExit callbacks
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo,
int layerIndex)
    {
        if(agent.remainingDistance <= agent.stoppingDistance)
        {
            Vector3 wayPoint = wayPoints[Random.Range(0, wayPoints.Count)].position;
            animator.transform.LookAt(wayPoint);
            agent.SetDestination(wayPoint);
        }

        timer += Time.deltaTime;
        if (timer > 10)
        {
            animator.SetBool("IsPatrolling", false);
        }

        float cR;

        // New feature
        if (pc.isSneaking)
        {
            cR = chaseRange * 0.3f;
        }
        else cR = chaseRange;
        //

        float distance = Vector3.Distance(playerTransform.position,
animator.transform.position);
        if (distance < cR)
        {
            animator.SetBool("IsChasing", true);
        }

    }
}
```

Table 15. Improved enmey AI patrol state

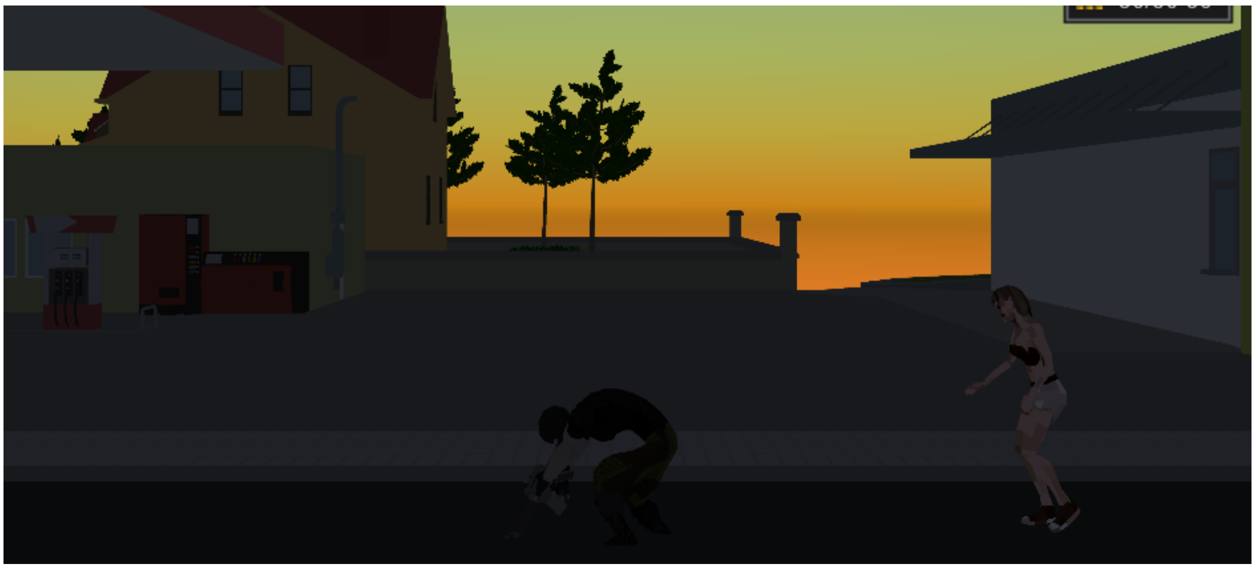


Figure 18. New sneaking animation

Laboratory work #3

List of tasks (main functionality of your project)

1. Building interior
2. Stamina mechanic
3. Melee attack
4. Different enemies
5. Home base
6. Save/Load system
7. Dynamic music
8. Sound effects
9. Upgrades
10. Looting system
11. Resources

Solution

Task #1. *Building interior*

Using assets, we found on the internet we built home interiors that you can enter an explore.

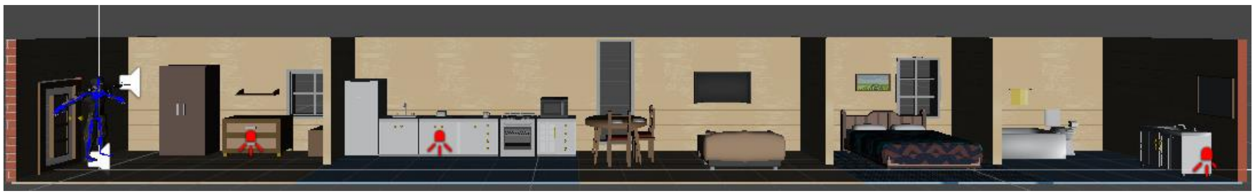


Figure 19. First house interior

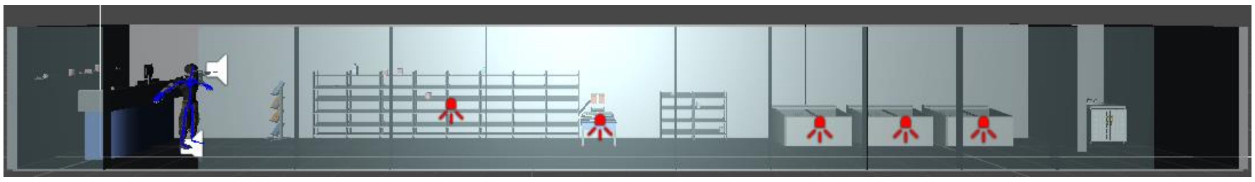


Figure 20. Gas station interior

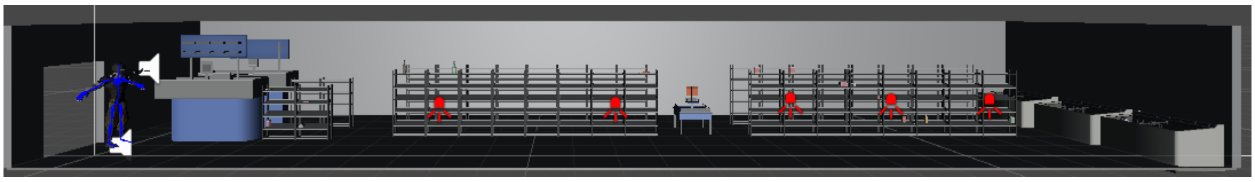


Figure 21. Supermarket interior

Task #2. *Stamina mechanic*

We added a stamina mechanic that effects how long you can run and if you can perform a melee attack.

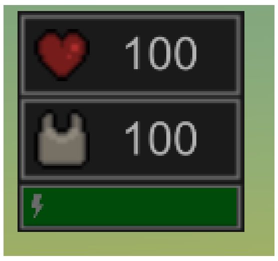


Figure 22. Stamina bar added to UI

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

public class Stamina : MonoBehaviour
{
    [SerializeField]
    private float stamina;

    [SerializeField]
    private float maxStamina = 100;

    [SerializeField]
    private Image staminaBar;

    private float calmTime = 0;

    private float runTime = 0;

    private PlayerController pc;

    private float fillBarRatio;

    private Backpack backpack;

    // Start is called before the first frame update
    void Start()
    {
        backpack =
GameObject.FindGameObjectWithTag("Backpack").GetComponent<Backpack>();
        pc = GetComponent<PlayerController>();

        stamina = backpack.stamina;
        maxStamina = 100 + backpack.EnduranceUp * 10;

        fillBarRatio = 1 / maxStamina;
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        calmTime += Time.deltaTime;

        if (pc.GetIsRunning())
        {
            runTime += Time.deltaTime;
        }
        else runTime = 0;

        if(runTime > 1)
        {
            UseStamina(5);
            runTime = 0;
        }

        if (calmTime >= 5 && stamina != maxStamina)
        {
            StaminaRegen();
            calmTime = 0;
        }
    }

    private void StaminaRegen()
    {
        if ((stamina + 10) >= maxStamina) stamina = maxStamina;
        else stamina += 10;

        staminaBar.fillAmount = stamina * fillBarRatio;
    }

    public void UseStamina(float used)
    {
        if ((stamina - used) <= 0) stamina = 0;
        else stamina -= used;

        calmTime = 0;
        staminaBar.fillAmount = stamina * fillBarRatio;
    }

    public float GetStamina()
    {
        return stamina;
    }
}
```

Table 16. Stamina script

Task #3. *Melee attack*

We added a melee attack, that player can perform if he has enough stamina. The attack is short range but is good for handling crowds.

Figure 23. Player kicking animation

```
public class MeleeAttack : MonoBehaviour
{
    [SerializeField]
    private float minDmg = 5;

    [SerializeField]
    private float maxDmg = 10;
    [SerializeField]
    private float meleeRange = 1f;

    [SerializeField]
    private Transform target;

    private LayerMask enemyLayer;

    private Animator playerAnimator;

    private Stamina stamina;

    float timer = 0f;

    private Backpack backpack;

    // Start is called before the first frame update
    void Start()
    {
        backpack =
GameObject.FindGameObjectWithTag("Backpack").GetComponent<Backpack>();
        minDmg = 6 + backpack.StrengthUp *2;
        maxDmg = 10 + backpack.StrengthUp * 2;

        playerAnimator = GetComponent<Animator>();
        enemyLayer = 1 << 8;
        stamina = GetComponent<Stamina>();
    }

    // Update is called once per frame
    void Update()
    {
        if (timer != 0)
        {
            timer += Time.deltaTime;
        }

        if(timer >= 0.7f)
        {
            Attack();
            timer = 0f;
        }

        if (Input.GetKeyDown(KeyCode.F) && stamina.GetStamina() >= 10f && timer ==
0f)
        {
            playerAnimator.SetTrigger("isMeleeing");
            timer += Time.deltaTime;
        }
    }

    private void Attack()
    {
        stamina.UseStamina(10f);

        Collider[] hitEnemies = Physics.OverlapSphere(target.position, meleeRange,
enemyLayer);

        foreach(Collider enemy in hitEnemies)
        {
            MobHealth enem = enemy.GetComponent<MobHealth>();
            if(enem)
            {
                enem.TakeDamage(Random.Range(minDmg,maxDmg));
            }
            }

        }
}
```
Table 17. Melee attack code

Task #4. *Different enemies*

9 new models were added to make enemies more diverse, all with their own specific ragdolls and colliders.

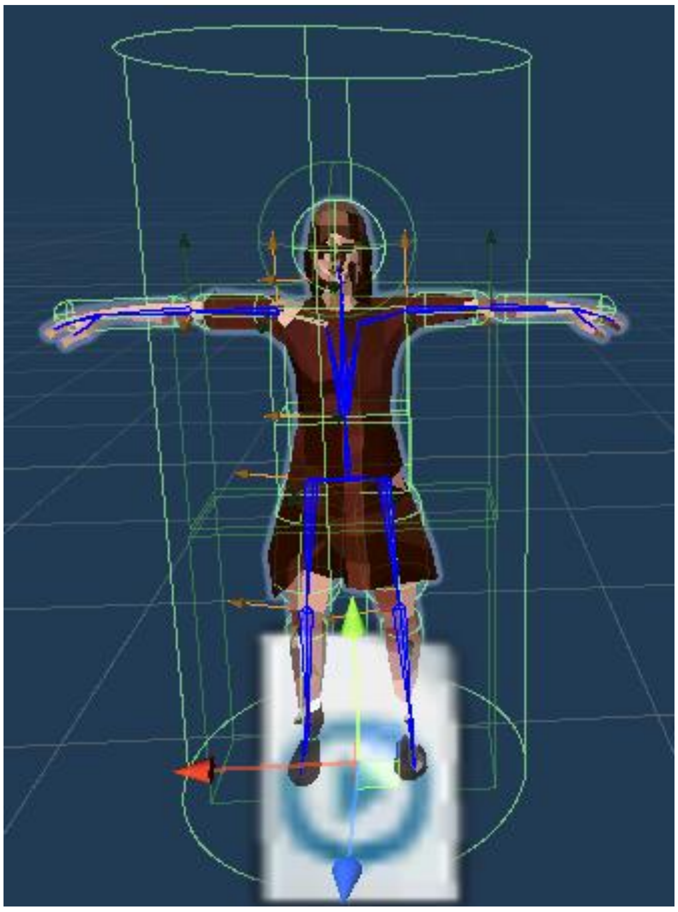


Figure 24. New enemy model 1

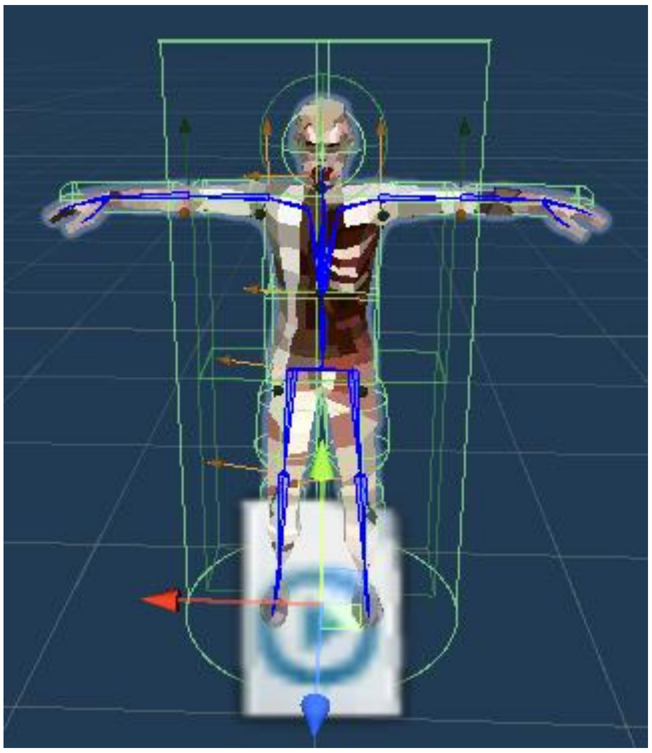
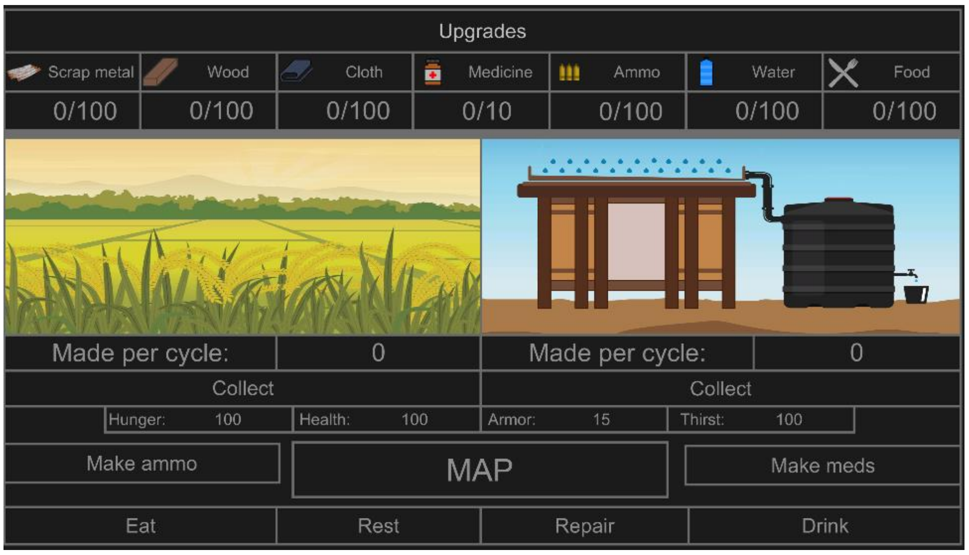


Figure 25. New enemy model 2

Task #5. *Home base*

We added a home base scene where player can store his collected resources, craft ammo and medicine, repair your armor, eat, drink, rest, upgrade character and base, collect generated food and water and start the next mission.

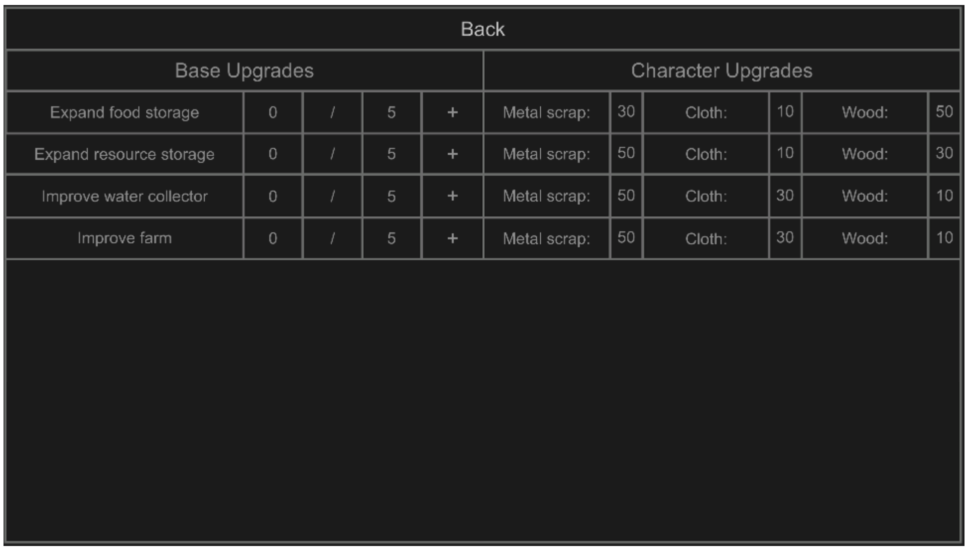


Scrap metal	Wood	Cloth	Medicine	Ammo	Water	Food
0/100	0/100	0/100	0/10	0/100	0/100	0/100

Made per cycle: 0
Made per cycle: 0

Hunger: 100
Health: 100
Armor: 15
Thirst: 100

Figure 26. Home base interface



Back										
Base Upgrades					Character Upgrades					
Expand food storage	0	/	5	+	Metal scrap:	30	Cloth:	10	Wood:	50
Expand resource storage	0	/	5	+	Metal scrap:	50	Cloth:	10	Wood:	30
Improve water collector	0	/	5	+	Metal scrap:	50	Cloth:	30	Wood:	10
Improve farm	0	/	5	+	Metal scrap:	50	Cloth:	30	Wood:	10

Figure 27. Home base upgrades menu

```
public void UpgradeFoodStorage()
    {
        if (foodStorageUpgrade < 5)
        {
            int[] Cost = FoodStorageUpCost();
            if (Cost[0] <= Scrap && Cost[1] <= Wood && Cost[2] <= Cloth)
            {
                foodStorageUpgrade++;
                Scrap -= Cost[0];
                Wood -= Cost[1];
                Cloth -= Cost[2];

                int[] Values = FoodStorageUpValue();
                MaxWater = Values[0];
                MaxFood = Values[1];

                Text FoodNoofUpgrades =
GameObject.Find("/Upgrades/BASE/VALUES/Expandfood/NumberofUpgrades").GetComponent<
Text>();
                Text MaxWaterText =
GameObject.Find("/Home/VALUES/Water/Max").GetComponent<Text>();
                Text MaxFoodText =
GameObject.Find("/Home/VALUES/Food/Max").GetComponent<Text>();

                MaxWaterText.text = $"{(int)MaxWater}";
                MaxFoodText.text = $"{(int)MaxFood}";
                FoodNoofUpgrades.text = $"{(int)foodStorageUpgrade}";
                UpdateCostFoodStorage();
            }
        }
    }

    public void UpgradeResourceStorage()
    {
        if (resourceStorageUpgrade < 5)
        {
            int[] Cost = ResourceStorageUpCost();
            if (Cost[0] <= Scrap && Cost[1] <= Wood && Cost[2] <= Cloth)
            {
                resourceStorageUpgrade++;
                Scrap -= Cost[0];
                Wood -= Cost[1];
                Cloth -= Cost[2];

                int[] Values = ResourceStorageUpValue();
                MaxScrap = Values[0];
                MaxWood = Values[1];
                MaxCloth = Values[2];
                MaxMedicine = Values[3];
                MaxAmmo = Values[4];

                Text ResourcesNoofUpgrades =
GameObject.Find("/Upgrades/BASE/VALUES/Expandresources/NumberofUpgrades").GetCompo
nent<Text>();
                Text MaxScrapText =
GameObject.Find("/Home/VALUES/ScrapMetal/Max").GetComponent<Text>();
                Text MaxWoodText =
GameObject.Find("/Home/VALUES/Wood/Max").GetComponent<Text>();
                Text MaxClothText =
GameObject.Find("/Home/VALUES/Cloth/Max").GetComponent<Text>();
                Text MaxAmmoText =
GameObject.Find("/Home/VALUES/Ammo/Max").GetComponent<Text>();
                Text MaxMedicineText =
GameObject.Find("/Home/VALUES/Meds/Max").GetComponent<Text>();

                MaxScrapText.text = $"{(int)MaxScrap}";
                MaxWoodText.text = $"{(int)MaxWood}";
                MaxClothText.text = $"{(int)MaxCloth}";
                MaxAmmoText.text = $"{(int)MaxAmmo}";
                MaxMedicineText.text = $"{(int)MaxMedicine}";
                ResourcesNoofUpgrades.text = $"{(int)resourceStorageUpgrade}";
                UpdateCostResourceStorage();
            }
        }
    }

    public void UpgradeWaterCollector()
    {
        if (waterCollectorUpgrade < 5)
        {
            int[] Cost = WaterCollectorUpCost();
            if (Cost[0] <= Scrap && Cost[1] <= Wood && Cost[2] <= Cloth)
            {
                waterCollectorUpgrade++;
                Scrap -= Cost[0];
                Wood -= Cost[1];
                Cloth -= Cost[2];
                waterPerCycle = WaterCollectorUpValue();

                Text WaterCollectorNoofUpgrades =
GameObject.Find("/Upgrades/BASE/VALUES/Improvewater/NumberofUpgrades").GetComponen
t<Text>();
                Text waterPerCycleText =
GameObject.Find("/Home/VALUES/WaterMade").GetComponent<Text>();

                waterPerCycleText.text = $"{(int)waterPerCycle}";
                WaterCollectorNoofUpgrades.text = $"{(int)waterCollectorUpgrade}";

                UpdateCostWaterCollector();
            }
        }
    }

    public void UpgradeFarm()
    {
        if (farmUpgrade < 5)
        {
            int[] Cost = FarmUpCost();
            if (Cost[0] <= Scrap && Cost[1] <= Wood && Cost[2] <= Cloth)
            {
                farmUpgrade++;
                Scrap -= Cost[0];
                Wood -= Cost[1];
                Cloth -= Cost[2];
                foodPerCycle = FarmUpValue();

                Text FarmNoofUpgrades =
GameObject.Find("/Upgrades/BASE/VALUES/Improvefarm/NumberofUpgrades").GetComponent
<Text>();
                Text foodPerCycleText =
GameObject.Find("/Home/VALUES/FarmMade").GetComponent<Text>();

                foodPerCycleText.text = $"{(int)foodPerCycle}";
                FarmNoofUpgrades.text = $"{(int)farmUpgrade}";

                UpdateCostFarm();
            }
        }
    }
```
Table 18. Fragments from Stats script

Task #6. *Save/Load system*

We added Save/Load system, that creates a binary data file when a new game is started that contains player stats, upgrades, and resources. The data is saved every time that player comes back to the home base. We have also implemented hardcore mechanics as the save file is deleted if the player dies and the game must be started from scratch.

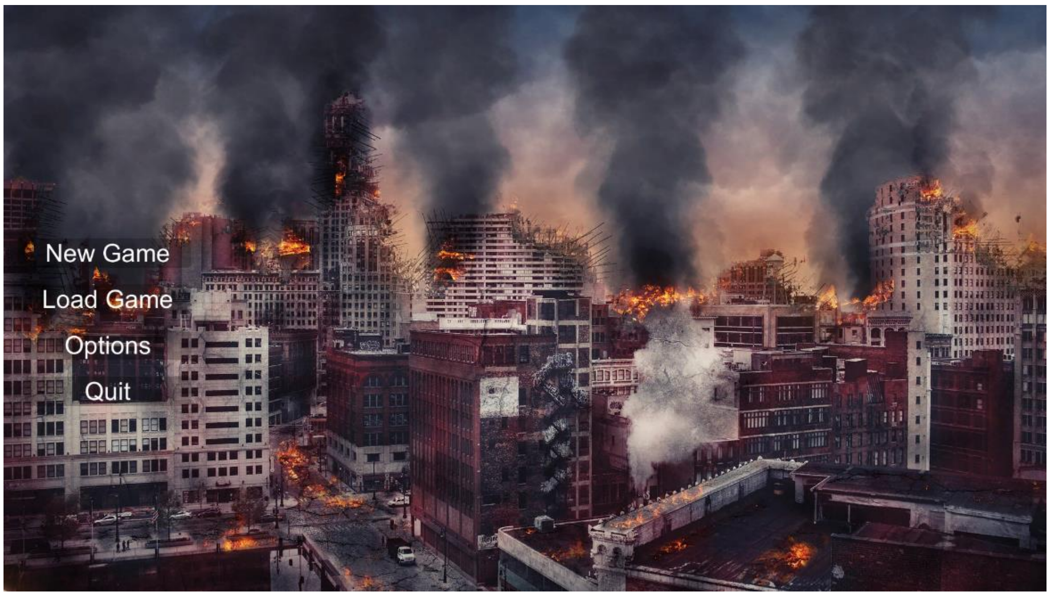


Figure 28. MainMenu with New Game and Load Game buttons

```
using UnityEngine;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public static class SaveSystem
{
    public static void Save(Stats stats)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/stats.data";
        FileStream steram = new FileStream(path, FileMode.Create);

        StatsData data = new StatsData(stats);
        formatter.Serialize(steram, data);
        steram.Close();
    }

    public static StatsData Load()
    {
        string path = Application.persistentDataPath + "/stats.data";
        if (File.Exists(path))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream steram = new FileStream(path, FileMode.Open);

            StatsData data = formatter.Deserialize(steram) as StatsData;
            steram.Close();
            return data;
        }
        else return null;
    }
}
```

Table 19. SaveSystem script

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

[System.Serializable]
public class StatsData
{
    public int Scrap;
    public int MaxScrap;

    public int Wood;
    public int MaxWood;

    public int Cloth;
    public int MaxCloth;

    public float Medicine;
    public float MaxMedicine;

    public float Ammo;
    public float MaxAmmo;

    public int Water;
    public int MaxWater;

    public int Food;
    public int MaxFood;

    public int Hunger;
    public int Thirst;

    public float Health;
    public float Armor;

    public int waterPerCycle;
    public int wateCollected;

    public int foodPerCycle;
    public int foodCollected;

    public int BacpackUpgrade;
    public int AccuracyUpgrade;
    public int StrengthUpgrade;
    public int PerceptionUpgrade;
    public int SpeedUpgrade;
    public int EnduranceUpgrade;


    public int foodStorageUpgrade;
    public int resourceStorageUpgrade;
    public int waterCollectorUpgrade;
    public int farmUpgrade;

    public StatsData (Stats stat)
    {
        Scrap = stat.Scrap;
        MaxScrap = stat.MaxScrap;

        Wood = stat.Wood;
        MaxWood = stat.MaxWood;

        Cloth = stat.Cloth;
        MaxCloth = stat.MaxCloth;

        Medicine = stat.Medicine;
        MaxMedicine = stat.MaxMedicine;

        Ammo = stat.Ammo;
        MaxAmmo = stat.MaxAmmo;

        Water = stat.Water;
        MaxWater = stat.MaxWater;

        Food = stat.Food;
        MaxFood = stat.MaxFood;

        Hunger = stat.Hunger;
        Thirst = stat.Thirst;

        Health = stat.Health;
        Armor = stat.Armor;

        waterPerCycle = stat.waterPerCycle;
        wateCollected = stat.wateCollected;

        foodPerCycle = stat.foodPerCycle;
        foodCollected = stat.foodCollected;

        BacpackUpgrade = stat.BacpackUpgrade;
        AccuracyUpgrade = stat.AccuracyUpgrade;
        StrengthUpgrade = stat.StrengthUpgrade;
        PerceptionUpgrade = stat.PerceptionUpgrade;
        SpeedUpgrade = stat.SpeedUpgrade;
        EnduranceUpgrade = stat.EnduranceUpgrade;


        foodStorageUpgrade = stat.foodStorageUpgrade;
        resourceStorageUpgrade = stat.resourceStorageUpgrade;
        waterCollectorUpgrade = stat.waterCollectorUpgrade;
        farmUpgrade = stat.farmUpgrade;
    }
}
```

Table 20. StatsData script

```
public void NewGame()
    {
        string path = Application.persistentDataPath + "/stats.data";
        if(File.Exists(path))
        {
            File.Delete(path);
        }

        SceneManager.LoadScene(1);
    }

    public void LoadGame()
    {
        string path = Application.persistentDataPath + "/stats.data";
        if (File.Exists(path))
        {
            SceneManager.LoadScene(1);
        }
    }
```

Table 21. Fragment from SceneLoader script

Task #7. *Dynamic music*

We implemented dynamics music. The music changes depending on players health and if he is being attacked or not. The health threshold to change music is 40.

```
void Update()
    {
        spawnTimer += Time.deltaTime;
        if (spawnTimer >= spawnTime)
        {
            AddNewEnemys();
            spawnTimer = 0f;
        }

        float health = hp.GetHealth();
        isAttacked = ChcekState();

        if (health <= 40f && currentClip != 1 && !isAttacked) nextClip = 1;
        else if (health > 40f && currentClip != 0 && !isAttacked) nextClip = 0;
        else if (health > 40f && currentClip != 2 && isAttacked) nextClip = 2;
        else if (health <= 40f && currentClip != 3 && isAttacked) nextClip = 3;

        if(nextClip != currentClip && !changing)
        {
            Music[nextClip].mute = false;
            changeTimer = 0f;
            changing = true;
        }
        changeTimer += Time.deltaTime;

        if(changeTimer < 10f && changing)
        {
            Music[nextClip].volume = 0.1f * changeTimer;
            Music[currentClip].volume = 1 -0.1f * changeTimer;
        }

        if(changeTimer > 10f && changing)
        {
            changing = false;
            Music[currentClip].mute = true;
            Music[currentClip].volume = 0f;
            Music[nextClip].volume = 1f;
            currentClip = nextClip;
        }
    }
```

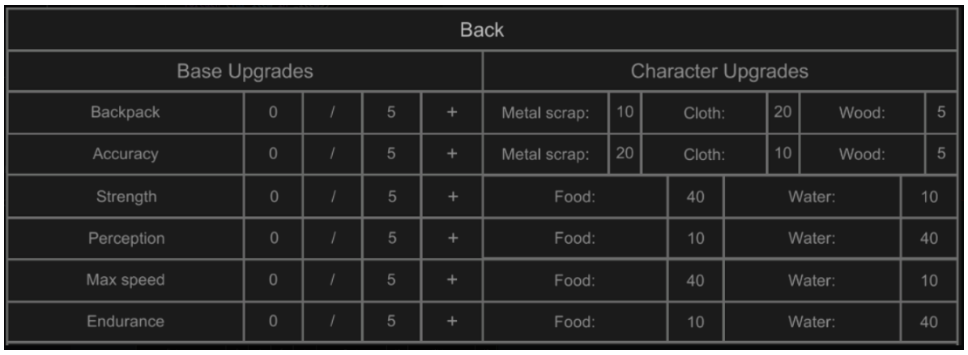
Table 22. Fragment from DynamicMusic script

Task #8. Sound effects

New sound effects were added to shooting and walking actions.

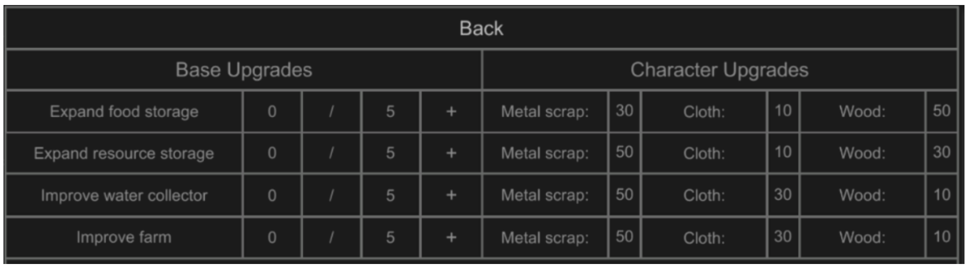
Task #9. *Upgrades*

Upgrades is a feature that allows player to use gathered resources and this way gain progress. There are two types of upgrades – home base upgrades and character upgrades. Home base upgrades can increase food storage, resource storage, improve water collectors or farms. Character upgrades can increase backpack size, accuracy, strength, perception, max speed and endurance. The higher the level of upgrade, the higher the costs to get that upgrade.



Back										
Base Upgrades					Character Upgrades					
Backpack	0	/	5	+	Metal scrap:	10	Cloth:	20	Wood:	5
Accuracy	0	/	5	+	Metal scrap:	20	Cloth:	10	Wood:	5
Strength	0	/	5	+	Food:	40	Water:	10		
Perception	0	/	5	+	Food:	10	Water:	40		
Max speed	0	/	5	+	Food:	40	Water:	10		
Endurance	0	/	5	+	Food:	10	Water:	40		

Figure 29. Character upgrades menu



Back										
Base Upgrades					Character Upgrades					
Expand food storage	0	/	5	+	Metal scrap:	30	Cloth:	10	Wood:	50
Expand resource storage	0	/	5	+	Metal scrap:	50	Cloth:	10	Wood:	30
Improve water collector	0	/	5	+	Metal scrap:	50	Cloth:	30	Wood:	10
Improve farm	0	/	5	+	Metal scrap:	50	Cloth:	30	Wood:	10

Figure 30. Home base upgrades menu

Task #10. *Looting system*

We needed a way to receive materials for upgrading, that’s why we created looting mechanic. Certain objects in buildings have colliders that activate lights and loot tooltip. When you press E, you get a random amount of 5 different resources. Every loot spot has weights for different materials and a fixed number of items you can receive in total. This number can be increased if player upgrades his perception level.

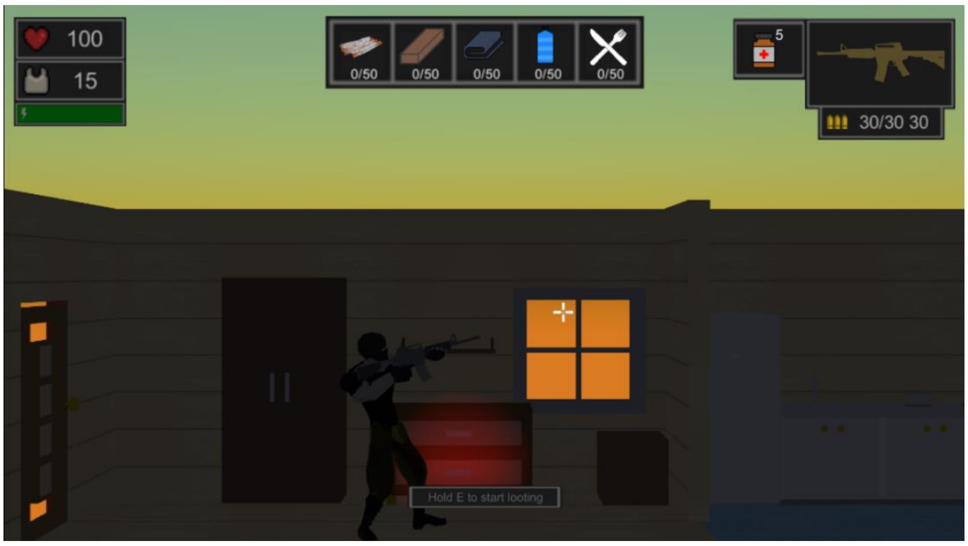


Figure 31. Picture showing Looting script in action

```
private void startLooting()
        {
            while(lootableItems > 0)
            {
                float diceRoll = Random.Range(0f, totalWeight);

                foreach (var item in _items)
                {
                    if (item.weight >= diceRoll)
                    {
                        item.quantity++;
                        lootableItems--;
                        break;
                    }

                    diceRoll -= item.weight;
                }
            }
            isLooted = true;
            avoidWrongNumbers();
            updateValues();
        }
```

Table 23. Fragment from Looting script

Task #11. *Resources*

We implemented 5 different main resources, that act like game's currency – scrap, wood, cloth, water and food. These resources are collected during gameplay and can be used to upgrade home base or character (costs increase with each upgrade). You can also use some of these resources to do different things like *Eat (consumes food)* or *Drink (consumes water)*. Resources can also be used to make ammunition or medicine.

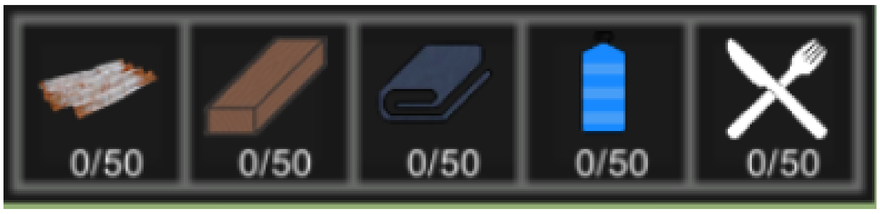


Figure 32. Picture showing collected resources bar

User's manual (for the Individual work defense)

How to play?
When you first start the game, you’re greeted in the main menu scene. From there, you can access options menu, load game if you started it before, or start a new game. Starting a new game loads the home base, from there you can press MAP button and select area in which you’re going to play. After selecting play area, you get short instructions on how to move and perform actions. Then player appears in the selected play area, from there you can walk, enter building, fight zombies. When in the buildings, you can loot marked objects to get different materials. At the top of the user interface, you can see limits of materials you can collect. When you come back to your home base, these materials can be spent to upgrade either your home base or your character. Your primary objective is to survive all the zombies lurking around the selected area, upgrade your character and home base, thus increasing your chances of your survival.

Descriptions of the rules of the game.

- Whenever players health drops to 0, you die and lose all of your progress;
- Below health and armor is a stamina bar, when player uses all of his stamina, he can no longer run or use melee attack;
- Your materials and ammunition is limited – when player runs out of ammunition, he can no longer use his gun, then the only way to get more ammunition is to gather materials and make ammunition in home base;
- Upgrading home base or character costs resources, the more upgrades are made, the more it will cost to continue upgrading.

Descriptions of the controls / keys.

- Hold A or D to walk left or right;
- Holding left shift while walking makes player run;
- Holding left ctrl while walking makes player sneak;
- Moving mouse makes player aim in the direction of the mouse;
- Pressing first mouse key makes player shoot in the following direction;
- You may also hold first mouse key to rapid fire in the following direction;
- Pressing E while at the entrance of the building makes player enter that building;
- Holding E while at the marked object makes player loot that object.

Literature list

1. https://www.turbosquid.com/3d-models/free-base-mesh-character-rigging-3d-model/582169#
2. https://www.wallpaperflare.com/architecture-buildings-city-cityscape-clouds-daytime-roads-wallpaper-aoucx
3. https://sketchfab.com/3d-models/polygonal-modern-weapons-asset-package-a35fc24f512e448ebed403beffe09c0d#download
4. https://www.aoaforums.com/forum/attachments/digital-image-photo-video-audio-editing/25132d1270227076-camoflage-seamless-texture-maps-free-use-camo_cloth_woodland_2048.png
5. https://c2.staticflickr.com/6/5170/5233243436_4818c1bcac.jpg
6. https://img.hutshopping.de/Conductive-Leather-Gloves-by-Stetson-black.48528_5rf4.jpg
7. https://d2dk0b2obiw7ur.cloudfront.net/media/catalog/product/cache/1/thumbnail/600x/17f82f742ffe127f42dca9de82fb58b1/_/1/_1244107218_3.jpg
8. https://assetstore.unity.com/packages/3d/environments/roadways/low-poly-road-pack-67288
9. https://sketchfab.com/3d-models/polygonal-zombies-with-animations-free-pack-d9bcfdd88f5348549bc947226af7c314
10. https://www.zapsplat.com/?s=7.62x39&post_type=music&sound-effect-category-id=
11. https://www.youtube.com/watch?v=dnRX_bHbYgs
12. https://vnbp.itch.io/low-poly-home-set
13. https://pavel-3d.itch.io/low-poly-city-vehicles-pack
14. https://pennyappeal.org/storage/app/media/appeals/muharram/rainwater-harvesting-systems/rain-water-harvesting-02-min-1.jpg
15. https://assetstore.unity.com/packages/3d/props/supermarket-interior-level-168412
16. https://www.kenney.nl/assets/food-kit
17. https://www.vecteezy.com/vector-art/128801-free-rice-field-illustration
18. https://www.pixelsquid.com/png/corrugated-metal-sheet-rusted-1216781312199759156?image=G07
19. https://craftpix.net/freebies/free-tds-modern-gui-pixel-art/?utm_source=twitter&utm_medium=social&utm_campaign=myselfhttps://assetstore.unity.com/packages/tools/animation/leantween-3595
20. https://www.microsoft.com/lt-lt/microsoft-365

ANNEX

All source code is contained in this part.