Razvoj 3D igre preživljavanja u Unity okruženju

Patafta, Nikola

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

NIKOLA PATAFTA

RAZVOJ 3D IGRE PREŽIVLJAVANJA U UNITY OKRUŽENJU

Diplomski rad

Pula, prosinac, 2023.

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

NIKOLA PATAFTA

RAZVOJ 3D IGRE PREŽIVLJAVANJA U UNITY OKRUŽENJU

Diplomski rad

JMBAG: 0313021521, redoviti student

Studijski smjer: Informatika

Predmet: Dizajn i programiranje računalnih igara

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, prosinac, 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Nikola Patafta, kandidat za magistra Informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

U Puli, prosinac, 2023.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Nikola Patafta dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Razvoj 3D igre preživljavanja u Unity okruženju

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, prosinac, 2023.

Potpis

Sadržaj

1. Uvod

U ovom diplomskom radu će se opisati proces izrade 3D računalne igre preživljavanja u Unity razvojnom okruženju. Unity je popularna i snažna razvojna platforma za izradu video igara koja omogućava stvaranje visokokvalitetnih i interaktivnih igra za različite platforme uključujući računala, konzole i mobilne uređaje [1]. Unity okruženje je razvijeno od strane Unity Technologies koje je prvi put objavljeno u lipnju 2005. godine, a verzija korištena u ovom radu je 2022.3.4f1.

Motivacija za pisanje ovog diplomskog rada proizlazi iz želje za dubljim razumijevanjem procesa razvoja igara unutar Unity okruženja, s posebnim fokusom na FPS žanr. Među različitim žanrovima, pucačine u prvom licu (eng. First-Person Shooter) igre ističu se kao jedne od najpopularnijih i najutjecajnijih, pružajući igračima intenzivno iskustvo. FPS igre su zbog svoje popularnosti i tehničke zahtjevnosti idealan kandidat za istraživanje mogućnosti i ograničenja koje Unity pruža.

Cilj ovog rada je razviti cijelu igru u Unity okruženju, koristeći različite tehnike i alate koje pruža ova platforma. Rad će se usredotočiti na komponente kao što su upravljanje igračem, kontrola oružja, upravljanje inventarom, umjetna inteligencija neprijatelja, vizualna i zvučna obrada i optimizacija same igre.

Igra daje najveći naglasak na preživljavanje i snalaženje unutar svijeta te sadrži neke elemente horora kao što su stvaranje zombija u najgorim trenucima kako bi igrač dobio puni doživljaj same igre. Ne znajući što se dogodilo u svijetu u kojem se igrač stvorio, potičemo radoznalost igrača na istraživanje u kojem je cilj prikupiti što više korisnih oružja, specijalnih predmeta i informacija kako bi mogli napredovati kroz igru i na kraju saznati istinu što se zapravo dogodilo.

Slične igre na tržištu koje su poslužile kao inspiracija i motivacija ovoj igri su 7 Days to Die koja je razvijena u maloj kompaniji od samo 25 ljudi te je vrlo uspješna i igra The Forest koja je objavljena od studija Endnight Games u kojoj radi oko 20 zaposlenih.

Ovaj rad opisuje gotovo sve elemente izrade 3D računalne igre iz perspektive prvog lica, od same mehanike igre, animacija, kratkih scena, izrade priče, pozadinske glazbe, elemenata iznenađenja i na kraju optimizaciju igre.

2. Razvojno okruženje Unity

Unity je popularan i snažan alat za razvoj računalnih igara. Prvi puta je najavljen i objavljen u lipnju 2005. godine. Postepeno je proširen kako bi podržao razne platforme za stolna računala, mobilne uređaje, konzole i virtualnu stvarnost. Osobito je popularan za razvoj mobilnih igara za iOS i Android i također se smatra lakim za korištenje programerima početnicima i popularan je za razvoj nezavisnih igara.

2.1. Značajke Unity-a

Unity korisnicima nudi mogućnost stvaranja 2D i 3D igara, a alat nudi skup definiranih metoda, funkcija, protokola (eng. Application Programming Interface) koji omogućuju komunikaciju i interakciju između softverskih komponenti za programiranje u C# programskom jeziku. Koristi se Mono, kao uređivač Unity-a u obliku dodataka, za samu igru i za povlačenje i ispuštanje funkcionalnosti (eng. Drag & Drop) [1].

Unutar 3D igre Unity dopušta se specifikacija kompresije teksture, mipmapa i postavke razlučivosti za svaku platformu koju pokretač igre podržava. Pruža i mapiranje neravnina, refleksije, paralakse, ambijentalno okruženje prostora zaslona, dinamičke sjene, efekte prikaza u teksturi i naknadna obrada preko cijelog zaslona.

2.2. Trgovina Unity resursima

Trgovina Unity resursima (eng. Unity Asset Store) pruža velik i širok izbor gotovih resursa poput skripti, tekstura i dodataka za poboljšanje vlastitih ili postojećih projekta. Neki od resursa su besplatni za korištenje, a s druge strane kvalitetniji i jači resursi se naknadno plaćaju. Ujedno olakšava suradnju između programera i pruža tržište za kupoprodaju resursa. Unity ima veliku i aktivnu zajednicu programera koji dijele znanje, videozapise i resurse putem blogova i sličnih platformi. Opsežna i detaljna službena dokumentacija se nalazi na njihovim stranicama kako bi početnicima u programiranju pomogli da započnu i unaprijede svoje vještine.

2.3. Unity Render Pipelines

Unity podržava mogućnosti biranja između različitih cjevovoda prikaza (eng. render pipeline), a to su konceptualni okviri koji definiraju način na koji se scene, objekti i efekti prikazuju u Unity-u. Spomenuti okviri određuju kako se obrađuje grafički sadržaj i kako se prikazuje na ekranu. Postoje dva glavna cjevovoda prikaza a to su unaprijed postavljeni prikaz (eng. built-in render pipeline) i skriptni cjevovod prikaza (eng. scriptable render pipeline).

1. Unaprijed postavljeni prikaz (eng. Built-in) je unaprijed postavljena implementacija prikaza cjevovoda koja je dostupna u Unity-u prije verzije 2019.3., a koristi pripremljene i integrirane metode za obradu i prikazivanje grafike. Ovaj prikaz pruža određenu fleksibilnost, ali ima ograničenja u prilagodbi i optimizaciji za specifične potrebe projekta.
2. Skriptni prikaz cjevovoda (eng. scriptable render pipeline) je konceptualni okvir koji omogućuje programerima da stvore prilagođene i visoko optimizirane cjevovode prikaza prema svojim specifičnim potrebama. SRP pruža kontrolu svakog koraka prikaza, uključujući geometrijsku obradu, osvjetljenje, sjene, efekte i prikaz na zaslonu. Ovaj prilagodljiv pristup omogućuje programerima da postignu bolje performanse i vizualnu kvalitetu za svoje projekte, te pruža dvije glavne implementacije:

   URP (eng. Universal Render Pipeline) je unaprijed postavljena skriptabilna implementacija cjevovoda prikaza a optimizirana je za mobilne uređaje, računalne igre srednje klase i druge platforme. Samim tim pruža puno veću fleksibilnost od prije spomenutog unaprijed postavljenog prikaza, a cilja na optimalnu ravnotežu između vizualne kvalitete i performansi.

   HDRP (eng. High Definition Render Pipeline) je unaprijed postavljena skriptabilna implementacija cjevovoda prikaza koja je optimizirana za visokokvalitetne vizualne efekte i realistične grafike. Pruža napredne mogućnosti osvjetljenja, više odabira sjena, ljepše efekte i kvalitetnije materijale.

Ovaj prikaz omogućuje programerima stvaranje vlastitih implementacija koje im najbolje odgovaraju, pružajući im maksimalnu fleksibilnost i kontrolu nad alatima u Unity-u.

3. Slične igre

3.1. 7 Days to Die

7 Days to Die je igra otvorenog svijeta koja je jedinstvena kombinacija pucačine iz prvog lica, preživljavanja, obrane tornjeva i igranja uloga. Igrač u igri kontrolira sam sebe pokušavajući preživjeti u okolini koja jr okružena zombijima i zgradama. Događaji igre odvijaju se tijekom nuklearnog napada u Trećem svjetskom ratu, koji je uništio izuzetno velik dio svijeta, osim nekih područja kao što je izmišljena županija Navezgane, Arizona. Sadržava elemente iskustva (eng. experience) koji se dobivaju ubijanjem zombija, kreiranjem novih stvari, izradom i nadogradnjom kuće i tako dalje. Slika 1 prikazuje igru 7 Days to Die [2].

Slika 1. Igra - 7 days to die

Kroz igru se može se pronaći puno elemenata iznenađenja, kao na primjer iza ulaznih vrata počiva neprijatelj koji se budi nakon što igrač dolazi u bliski kontakt s njime. Također postoje i zamke u objektima, stoga je potrebno obratiti pozornost na kretanje igrača. Igra je izašla 2013. godine, zbog čega nije u najnovijim standardima optimizacije i grafike, ali svojom elementima iznenađenjima i zadacima (eng. quests) nam može na duže vrijeme zaokupirati um i povećati kreativnost kod igranja.

3.2. The Forest

The Forest je horor igra preživljavanja koju je razvio i objavio Endnight Games. Odvija se na udaljenom poluotoku prekrivenom šumom, gdje se lik igrača Eric LeBlanc mora boriti protiv kanibalskih čudovišta, dok traži svog sina Timmyja nakon pada aviona. Igra ima nelinearno igranje u okruženju otvorenog svijeta koja se igra iz perspektive prvog lica, nema postavljene misije ili zadatake, što jeća igrača da donosi vlastite odluke za preživljavanje. Na slici 2 je prikazana igra The Forest [3].

Slika 2. Igra - The Forest

Kako igrač napreduje kroz igru istražujući špilje ispod površine šume, nailazi na sve bizarnije mutacije, uključujući deformirane bebe i mutante s nekoliko dodataka. Igra također ima dnevni i noćni ciklus, pri čemu igrač može izgraditi sklonište i zamke, loviti životinje i skupljati zalihe tijekom dana, te se braniti od mutanata noću.

4. A Week To Survive

A Week To Survive je računalna igra preživljavanja s elementima horora. Igra započinje stvaranjem lika na zemlji, koji se budi prvog dana u 6 sati ujutro, a vrijeme buđenja je naglašeno jer postaje važan element kasnije u igri. Na vrhu ekrana korisničkog sučelja nalazi se 24-satni sat i pokazivač koji označava trenutni dan. Dan u igri završava u 22 sata navečer, a nakon ponoći, kao i u stvarnom svijetu, dolazi sljedeći dan. Igra se odvija u imaginarnoj maloj državi okruženoj planinama zvanoj Resnik.

Nakon stvaranja lika, glavni zadatak igrača je prikupiti što više korisnih stvari, alata, oružja i prije noći pronaći sigurno sklonište u obliku napuštene kuće ili građevine. Većina napuštenih kuća sastoji se od golih zidova, starog, napuštenog namještaja, te je moguće pronaći neke korisne stvari koje igrač može pokupiti. S druge strane, postoje i kuće koje mogu poslužiti kao sklonište preko noći, a koje imaju i vrata, prozore.

Glavne karakteristike zombija se znatno razlikuju tijekom dana i noći. Tijekom dana su bezopasni jer im je kretanje i vid ograničen, te ne mogu napasti igrača ako mu se ovaj ne približi. Međutim, tijekom noći se karakteristike mijenjaju: lik postaje ograničen u kretanju i vidu, a zombiji postaju brži i imaju bolji vid, te su puno agresivniji.

Aspekt preživljavanja igre bazira se na tome da igrač tijekom tjedna mora nabaviti dovoljno alata i oružja za preživljavanje. Sedmog dana igre, lika okružuje grupa zombija protiv koje se mora boriti kako bi preživio. Igrač ima ljestvicu života, a hrana koju može pronaći u svijetu mu pomaže povratiti energiju i osnažuje ga. Ako igrač dođe u situaciju u kojoj ga zombiji pobjede, igra se zaustavlja na trenutnom satu i danu kako bi se vidjeli bodovi.

5. Izrada računalne igre

U ovom poglavlju bit će opisani svi elementi razvoja igre A Week To Survive u Unity okruženju. Obradit će se sve skripte koje se tiču igrača, njegovog kretanja, moment i okret kamere, inventar i popratne skripte koje su potrebne za logiku i potpunu funkcionalnost. Nakon toga objasnit će se neprijatelji i njihove skripte, vremenski ciklus od 24 sata, pozadinska muzika, upravitelj igre, scene i okruženje, naknadna obrada (eng. post processing) i neki detalji.

5.1. Igrač

Spomenuta igra se odvija iz perspektive prvog lica što znači da je kamera i osoban pogled smješten u očima igrača. Iz te perspektive vide se ruke i sve što se drži u njima. Ova perspektiva također omogućuje igračima da se osjećaju povezanima sa svojim likom, jer sve što se događa u igri percipiraju kao da se događa njima osobno. Kada lik krene ili skrene, igrač to doživljava kroz svoje vlastite pokrete miša ili kontrolera. Ova interaktivnost i sinkronizacija između igrača i lika dodatno povećava osjećaj uključenosti i potiče emocionalnu vezu s likom.

5.1.1. Kretanje igrača

Kretanje igrača je jedna od osnovnih funkcija svake igre. U nastavku bit će objašnjeno kretanje igrača pomoću skripte. Programski kod 1 prikazuje klasu PlayerMovement.

```
public class PlayerMovement : MonoBehaviour
{
    private CharacterController character_Controller;

    private Vector3 move_Direction;
    public float speed = 3f;
    public float gravity = 40f;
    public float jumpForce = 8f;
    private float verticalVelocity;

    Unity Message | 0 references
    void Awake()
    {
        character_Controller = GetComponent<CharacterController>();
    }
```

Programski kod 1. PlayerMovement - 1.dio

Na početku same skripte definirane su početne varijable kao što su Upravljač lika (eng. CharacterController) što je ujedno i klasa u Unity-u koja omogućava izvršavanje osnovnih kretnji. Definiran je Vektor 3 koji u sebi sadržava 3 smjera kretanja, koji su definirani kao x, y i z. Pomoću vektora se kontrolira kretnja igrača po horizontalnoj liniji odnosno gore, dolje, lijevo i desno, dok se po vertikalnoj liniji kontrolira kretnja gore i dolje, što je u slučaju ove igre skok pritiskom razmaknice (eng. spacebar). Također su postavljene neke varijable kao što su brzina (eng. speed) koja određuje brzinu kretanja igrača kroz svijet, silu gravitacije (eng. gravity), snagu skoka (eng. jumpForce) koja predstavlja koliko visoko možemo preskočiti neki objekt i vertikalnu brzinu koja je promjenjiva varijabla i koja će biti kasnije objašnjena kroz kod. Programski kod 2 prikazuje funkciju kretanje igrača.

```
void Update()
{
    MoveThePlayer();
}

1 reference
void MoveThePlayer()
{
    move_Direction = new Vector3(Input.GetAxis(Axis.HORIZONTAL), 0f, Input.GetAxis(Axis.VERTICAL));
    move_Direction = transform.TransformDirection(move_Direction);
    move_Direction *= speed * Time.deltaTime;

    ApplyGravity();
    character_Controller.Move(move_Direction);

}

1 reference
void ApplyGravity()
{
    verticalVelocity -= gravity * Time.deltaTime;
    PlayerJump();

    move_Direction.y = verticalVelocity * Time.deltaTime;
}

1 reference
void PlayerJump()
{
    if(character_Controller.isGrounded && Input.GetKeyDown(KeyCode.Space))
    {
        verticalVelocity = jumpForce;
    }

}
```

Programski kod 2. PlayerMovement - 2.dio

Funkcija Update() izvršava se po modelu slika po sekundi (eng. frames per second) te sve funkcije i varijable unutar iste izvršavaju se kronološkim redoslijedom. U funkciji MoveThePlayer() novi vektor definira se unosom s tipkovnice koji vraća vrijednost između -1 i 1 pritiskom tipki W ili S, što određuje kretanje naprijed ili nazad, odnosno pritiskom tipki A ili D za kretanje lijevo ili desno.

U funkciji je definirana nova funkcija ApplyGravity(), koja pritišće igrača prema zemlji u svakom trenutku. Naredba Time.deltaTime predstavlja interval u sekundama između posljednjeg i sljedećeg kadra (eng. frame) u igri i služi kako bi igrač glatko padao u slučaju skoka, sprječavajući tako efekt teleportiranja. Posljednje u ovoj funkciji je PlayerJump(), koja provjerava je li igrač na tlu i je li korisnik pritisnuo tipku razmaknice kako bi igrač mogao skočiti. Programski kod 3 prikazuje klasu trčanja i čučanja igrača.

```
public class PlayerSprintAndCrouch : MonoBehaviour
{
    [SerializeField] CharacterController characterController;
    private PlayerMovement playerMovement;

    public float sprint_Speed = 10f;
    public float move_Speed = 5f;
    public float crouch_Speed = 2f;

    private Transform look_Root;
    private float stand_Height = 1.8f;
    private float crouch_Height = 1.6f;

    private bool isCrouching;

    private PlayerFootSteps player_FootSteps;

    //Zvuk za korake
    private float sprint_Volume = 1f;
    private float crouch_Volume = 0.1f;
    private float walk_Volume_min = 0.2f;
    private float walk_Volume_max = 0.6f;
    //***

    //Razmak izmedju koraka (u sekundama)
    private float walk_Step_Distance = 0.4f;
    private float sprint_Step_Distance = 0.25f;
    private float crouch_Step_Distance = 0.5f;
    //***

    private PlayerStats player_Stats;

    private float sprint_Value = 100f;
    private float sprint_Treshold = 10f;
```

Programski kod 3. PlayerSprintAndCrouch - 1.dio

Kao i u prethodnoj skripti, na početku su definirani upravljač lika, brzina trčanja, brzina kretanja, brzina kretanja kada igrač čuči, veličina lika kada stoji i kada čuči, pomoćne varijable koje omogućuju ispravno funkcioniranje koda, te zvukovi koraka. Ispred definicije metode CharacterController nalazi se Unity atribut zvan [SerializeField], koji se koristi za prikaz privatnih metoda ili polja unutar Unity okruženja.

Privatna polja nisu vidljiva unutar Unity okruženja, ali u slučaju uređivanja ili dodjeljivanja atributa potrebno ih je učiniti vidljivima. Ova metoda dohvaćanja je bolja i efikasnija od prethodno spomenute metode GetComponent jer Unity ne mora tražiti i dodjeljivati komponente nekoj skripti, stoga se ubrzava rad i učitavanje same igre. Premda je velik broj skripti u ovom projektu, potrebno je napraviti igru boljom i optimiziranijom kako bi je korisnici na slabijim računalima mogli pokrenuti. Funkcija Update() provjerava dva slučaja, odnosno dvije funkcije: Sprint() i Crouch(). U prvoj funkciji postoji nekoliko slučajeva. Prvi slučaj je ako igrač ima izdržljivost (eng. stamina), te ako pritisne lijevu tipku shift i ako se kreće, onda se postavlja brzina kretanja igrača na brzinu trčanja, čime igrač trči. Drugi slučaj je ako igrač otpusti lijevu tipku shift i ako ne čuči, stoga se brzina postavlja na brzinu normalnog hoda. Programski kod 4 prikazuje funkciju Sprint().

```
void Update()
{
    Sprint();
    Crouch();
}

1 reference
void Sprint()
{
    //ako imamo staminu onda trci
    if(sprint_Value > 0f)
    {
        if (Input.GetKeyDown(KeyCode.LeftShift) && !isCrouching && Input.GetKey(KeyCode.W))
        {
            playerMovement.speed = sprint_Speed;

            //footsteps audio settings
            player_FootSteps.step_Distance = sprint_Step_Distance;
            player_FootSteps.volume_Min = sprint_Volume;
            player_FootSteps.volume_Max = sprint_Volume;
            //***
        }

    }

    if (Input.GetKeyUp(KeyCode.LeftShift) && !isCrouching)
    {
        playerMovement.speed = move_Speed;

        //footstep audio settings
        player_FootSteps.step_Distance = walk_Step_Distance;
        player_FootSteps.volume_Min = walk_Volume_min;
        player_FootSteps.volume_Max = walk_Volume_max;
        //***

    }
```

Programski kod 4. PlayerSprintAndCrouch - 2.dio

U nastavku funkcije Sprint() provjerava se još nekoliko slučajeva kako bi igra ispravno funkcioniral. Pritiskom tipke lijevi shift i ako igrač ne čuči, ukupna izdržljivost se smanjuje množenjem faktora sprinta i funkcije Time.deltaTime, koja omogućuje glatku tranziciju ukupne količine izdržljivosti. Programski kod 5 prikazuje nastavak funkcije Sprint().

```
if (Input.GetKey(KeyCode.LeftShift) && !isCrouching && Input.GetKey(KeyCode.W))
{
    sprint_Value -= sprint_Treshold * Time.deltaTime;

    if (sprint_Value <= 0f)
    {
        sprint_Value = 0f;

        playerMovement.speed = move_Speed;
        player_FootSteps.step_Distance = walk_Step_Distance;
        player_FootSteps.volume_Min = walk_Volume_min;
        player_FootSteps.volume_Max = walk_Volume_max;
        WaitBeforeSprint();
    }
}
else
{
    if (sprint_Value != 100f)
    {
        if (isCrouching)
        {
            sprint_Value += (sprint_Treshold / 2f) * Time.deltaTime * 1.5f;
        }
        else
        {
            sprint_Value += (sprint_Treshold / 2f) * Time.deltaTime;
        }

        if (sprint_Value > 100f)
        {
            sprint_Value = 100f;
        }
    }
}
player_Stats.Display_StaminaStats(sprint_Value);
```

Programski kod 5. PlayerSprintAndCrouch - 3.dio

U sljedećem slučaju provjerava se ima li igrač još izdržljivosti i u slučaju da nema, razina se postavlja na 0 i aktivira se funkcija WaitBeforeSprint(). Ako navedeni slučaj nije točan i ako je izdržljivost ispod maksimuma, koji je postavljen na 100, izdržljivost se počinje puniti kroz vrijeme, odnosno ako igrač čuči, puni se 1,5 puta brže. Na kraju funkcije Update() ažurira se traka izdržljivosti koja je vidljiva u korisničkom sučelju (eng. User Interface – UI) u gornjem lijevom uglu ekrana. Programski kod 6 prikazuje spomenutu funkciju.

```
IEnumerator WaitBeforeSprint()
{
    yield return new WaitForSeconds(2f);
}
```

Programski kod 6. PlayerSprintAndCrouch - 4.dio

Navedena funkcija se zove korutina (eng. coroutine). Korutina je koncept koji omogućava asinkrono izvršavanje koda u Unity okruženju. Korutine omogućuju kontroliranje vremena izvršavanja određenih dijelova koda, čime se može postići usporeno ili paralelno izvršavanje. Korutina se izvršava kroz više kadrova u igri, stoga je kompliciranija za izvođenje, ali s druge strane osigurava da svi korisnici igre imaju isto iskustvo prilikom igranja. Na primjer, da se navedena funkcija izvodi u tipu funkcije Update(), igrač s jačim računalom bi imao prednost u odnosu na nekoga s slabijim računalom jer bi vrlo vjerojatno imao veći broj kadrova u sekundi (eng. framerate).

Da bi se korutine koristile, potrebno ih je definirati kao posebnu vrstu metode koja ima specifične karakteristike. Korutina se definira kao metoda s povratnim tipom koja se naziva IEnumerator. Ova metoda se ponaša kao ponavljač (eng. iterator) koji se može pauzirati i nastaviti tijekom vremena. U korutinama se može koristiti naredba yield, što je izraz za pauziranje izvođenja korutine na određeno vrijeme. U ovom slučaju je korištena naredba yield return new WaitForSeconds(2f), što znači da se nakon pozivanja korutine čeka dvije sekunde u realnom vremenu, a zatim se nastavlja izvršavanje preostalog koda. U korutinama također možemo koristiti naredbu yield return null ili yield break kako bi dali kontrolu Unity-u da nastavi s izvršavanjem sljedećeg koda.

5.1.2. Ljestvica života

Skripta HealthScript() je univerzalna skripta za određivanje razine života igraču i svim neprijateljima. Skripta je univerzalna za obje strane jer sadrži velik broj istih elemenata, pa se zbog toga može primijeniti u više slučajeva. Na početku su definirane mnoge varijable, pošto tu skriptu koristi mnogo pomoćnih skripti kako bi se omogućila dobra funkcionalnost igre. Programski kod 7 prikazuje klasu HealthScript.

```
public class HealthScript : MonoBehaviour
{
    private EnemyAnimatior enemy_Anim;
    private NavMeshAgent navAgent;
    private EnemyController enemy_Controller;
    private PlayPlayerSound playPlayerSound;

    public float health = 100f;
    public bool is_Player, is_Boar, is_Zombie;
    public CameraShake cameraShake;

    private bool is_Dead;
    private bool playerDied;

    private EnemyAudio enemyAudio;

    private PlayerStats player_Stats;
    private EnemyStats enemy_Stats;

    //kontrole za deathscreen
    [SerializeField] private UIManager uiManager;

    [SerializeField]
    private BloodScreenEffect bloodScreenEffect;
```

Programski kod 7. HealthScript - 1.dio

Funkcija Awake() se poziva kada se objekt prvi puta kreira u sceni ili kada se scena učita. Ova metoda koristi se za inicijalizaciju objekta prije nego što počne izvršavanje ostatka koda. Vrlo je slična metodi Start(), koja se poziva odmah nakon nje, kada se objekt kreira, i odmah nakon što se igra pokrene. Na početku se provjerava je li kreirani objekt tipa divlje svinje (eng. Boar) ili zombija, te igrača. Nakon toga se dohvaćaju njihove potrebne skripte kako bi ostatak kreiranog objekta ispravno funkcionirao. Programski kod 8 prikazuje funkciju Awake().

```
void Awake()
{
    is_Dead = false;

    if (is_Boar || is_Zombie)
    {
        enemy_Anim = GetComponent<EnemyAnimatior>();
        enemy_Controller = GetComponent<EnemyController>();
        navAgent = GetComponent<NavMeshAgent>();
        enemyAudio = GetComponentInChildren<EnemyAudio>();
        enemy_Stats = GetComponent<EnemyStats>();

    }
    if (is_Player)
    {
        playPlayerSound = GetComponent<PlayPlayerSound>();
        player_Stats = GetComponent<PlayerStats>();
```

Programski kod 8. HealthScript - 2.dio

Na sljedećoj slici bit će prikazana javna metoda ApplyDamage(float damage), koja prima vrijednost tipa decimalnog broja, koja ujedno reprezentira količinu života koja se oduzima igraču ili neprijatelju svaki put kada je pozvana. Varijabla tipa bool jedna je od najjednostavnijih varijabli za implementaciju. Sadrži dva stanja - točno (eng. true) i netočno (eng. false). Tako je klasificirana javna varijabla is_Dead tog tipa, koja se postavlja na true u trenutku kada razina života padne ispod 0. Stoga se prije početka izvršavanja skripte provjerava je li objekt mrtav ili nije, te se sukladno tome izvršava operacija koja oduzima razinu života i količinu napravljene štete.

Ako je objekt tipa igrač, poziva se funkcija koja u igri pušta kratak isječak zvuka, kao da je igrač stvarno napadnut, kako bi se dodao efekt. Nakon toga se poziva kratka korutina koja stvara efekt podrhtavanja kamere na nekoliko milisekundi, što također doprinosi realnosti efekta. Na kraju se vidi efekt krvi na zaslonu, koji je u obliku slike po cijelom ekranu, i mijenja se količina prozirnosti slike. Ako razina života dođe ispod 0, poziva se nova metoda koja provjerava tko je preminuo. Programski kod 9 prikazuje funkciju ApplyDamage().

```
public void ApplyDamage(float damage)
{
    if (is_Dead)
    {
        return;
    }
    else
    {
        health -= damage;
    }
    if (is_Player)
    {
        //display Health UI
        playPlayerSound.PlayPlayerHurtSound();
        player_Stats.Display_HealthStats(health);
        StartCoroutine(cameraShake.Shake());
        bloodScreenEffect.ChangeAlpha();
    }
    if (is_Boar || is_Zombie)
    {
        enemy_Stats.Display_EnemyHealth(health);
        if (enemy_Controller.Enemy_State == EnemyState.PATROL)
        {
            enemy_Controller.chase_Distance = 50f;
        }
    }
    if (health <= 0f)
    {
        CheckWhoDied();
        is_Dead = true;
```

Programski kod 9. HealthScript - 3.dio

U metodi CheckWhoDied() prvo se provjerava je li igrač preminuo, te se zatim poziva prikladna animacija na aktivnom objektu, odnosno neprijatelju. Ujedno se gasi njihov upravljač, kao i metoda navAgent. Unity ima ugrađenu komponentu NavMeshAgent, koja se koristi za inteligentno kretanje objekata u 3D svijetu. Ova komponenta je dio Unity sustava za navigaciju (NavMesh), koji omogućava objektima da izbjegavaju prepreke, prate zadane putanje i automatiziraju svoje kretanje preko kompleksnih terena. Programski kod 10 prikazuje spomenutu funkciju.

```
void CheckWhoDied()
{
    if (!is_Player)
    {
        enemy_Anim.Dead();
        enemy_Controller.enabled = false;
        navAgent.enabled = false;
        StartCoroutine(DeadSound());

        if (is_Zombie)
        {
            EnemyManager.instance.EnemyDied(true);
        }
        else
        {
            EnemyManager.instance.EnemyDied(false);
        }
        Invoke("TurnOffGameObject", 5f);
    }

    if (is_Player)
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
        for (int i = 0; i < enemies.Length; i++)
        {
            enemies[i].GetComponent<EnemyController>().enabled = false;
        }
        EnemyManager.instance.StopSpawning();
        playPlayerSound.PlayPlayerDeathSound();
        playerDied = true;

        GetComponent<PlayerMovement>().enabled = false;
        GetComponent<PlayerAttack>().enabled = false;
        GetComponent<WeaponShooting>().enabled = false;

        //UiManager ako je player dead
        uiManager.isPaused = true;
        uiManager.SetActiveHud(false);
```

Programski kod 10. HealthScript - 4.dio

Započinje se kratka korutina DeadSound(), koja reproducira zvuk neprijatelja. Provjerava se je li preminuli neprijatelj tipa zombija ili divlje svinje s dodatnom varijablom tipa bool, te se poziva metoda u skripti EnemyManager koja će biti obrađena kasnije. Ako je u metodi CheckWhoDied() preminuo igrač, tada se pokreće metoda koja pretražuje sve aktivne objekte u sceni putem Unity funkcije tag, te im nakon toga gasi kontrole, čime se pauziraju u sceni.

Nakon toga se zove skripta EnemyManager i metoda StopSpawning(), koja prekida stvaranje svih novih neprijatelja u sceni, gase se pomoćne skripte na igraču, te se igra pauzira.

U kratkoj klasi PlayerStats koriste se elementi korisničkog sučelja tipa slike i teksta kako bi njihove varijable mogle promijeniti kroz kod. U metodi Display_HealthStats uzima se količina života na igraču, dijeli se sa 100 jer je trenutna maksimalna razina života postavljena na 100. Količina popunjenosti slike ima nekoliko svojstava kao što su tip slike, metoda popunjenosti, način popunjenosti i jedno od bitnih svojstava je količina ispunjenosti (eng. fill amount). Količina popunjenosti je fiksna od 0, što znači potpuno prozirno, do 1, što bi u ovom kontekstu značilo da se količina života dijeli sa 100 kako bi se ljestvica popunjenosti postupno smanjivala kako igrač gubi više života. Programski kod 11 prikazuje klasu PlayerStats.

```
public class PlayerStats : MonoBehaviour
{
    [SerializeField]
    private Image health_Stats, stamina_Stats;
    [SerializeField]
    private Text Healthtext;

    [SerializeField]
    private WeaponUI weaponUI;


    2 references
    public void Display_HealthStats (float healthValue)
    {
        healthValue /= 100f;
        health_Stats.fillAmount= healthValue;
        Healthtext.text = (healthValue*100).ToString();
    }

    1 reference
    public void Display_StaminaStats(float staminaValue)
    {
        staminaValue /= 100f;
        stamina_Stats.fillAmount = staminaValue;
```

Programski kod 11. PlayerStats - 1.dio

5.1.3. Prikupljanje stvari

Vrlo bitan aspekt igre je pronalaženje korisnih stvari u svijetu kako bi igrač mogao preživjeti. Jedan od tih aspekata je međusobna interakcija svijeta i igrača. U klasi PlayerPickup bit će

objašnjeno kako igrač može pokupiti korisne stvari u svijetu. Na početku je definirana udaljenost odnosno razmak od kojeg igrač može prikupiti stvari, maska slojeva (eng. LayerMask), kamera, tekst, inventar i pomoćna varijabla koja određuje domet prikupljanja.

LayerMask je poseban tip podataka koji se koristi za definiranje skupova slojeva u Unity sceni. LayerMask omogućuje efikasno upravljanje kolizijama i provjerom kolizija samo između određenih slojeva, čime se ubrzava obrada i smanjuje nepotrebno pretraživanje svih objekata u sceni. Također je definirana nova metoda iz skripte InventoryManager koja će biti spomenuta kasnije. Programski kod 12 prikazuje klasu PlayerPickup.

```
public class PlayerPickup : MonoBehaviour
{
    [SerializeField] private float pickupRange;
    [SerializeField] private LayerMask pickupLayer;
    [SerializeField] private Camera cam;
    [SerializeField] private TextMeshProUGUI textMeshPro;
    [SerializeField] private InventoryManager inventoryManager;

    private float picturealpha;
```

Programski kod 12. PlayerPickup - 1.dio

Funkcija RayCastPickupWeapons() omogućuje igraču da pokupi iz svijeta igre stvari ili oružja koje kasnije može koristiti kroz svoj inventar. Unity zrak (eng. ray) je linija u 3D prostoru koja se koristi za provjeru sudara i interakciju s objektima u sceni. Zrak je jednostavno definiran svojom početnom točkom i smjerom u kojem se kreće. Početna točka je uzeta iz kamere i popraćena je koordinatama, odnosno širinom i visinom zaslona koje se dijele na dva kako bi se dobila sredina. Naredba RaycastHit je struktura koja se koristi za spremanje informacija dobivenih iz Ray naredbe [5].

Potom se koristi naredba Physics.Raycast, koja prima argumente kao što su točka od koje kreće zrak, smjer, duljina i na kraju se koristi LayerMask. Nakon što Unity detektira koliziju između zrake i objekta koji se nalazi u određenoj maski sloja, koristi se naredba hit, u kojoj se spremaju informacije o koliziji s objektom, čije ime se uzima i prikazuje u određenom tekstualnom okviru kako bi igrač stekao dojam o čemu se radi. Velik broj elemenata u igri zahtijeva takvu interakciju igrača i svijeta, stoga će biti objašnjena samo jedna skripta. Identičnu metodu koristi interakcija otvaranja i zatvaranja vrata, samo su korišteni drugi parametri i animacija,

ako je pritisnuta tipka E na tipkovnici. Programski kod 13 prikazuje funkciju RayCastPickupWeapons().

```
private void RayCastPickupWeapons()
{
    Ray ray = cam.ScreenPointToRay(new Vector3(Screen.width / 2, Screen.height / 2));
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit, pickupRange, pickupLayer))
    {
        string currentWeapon = hit.transform.GetComponent<ItemObject>().item.name;
        textMeshPro.text = "Pickup: " + currentWeapon;
        if (picturealpha <= 0)
        {
            picturealpha -= (Time.deltaTime / 10);
            textMeshPro.color -= new Color(0, 0, 0, picturealpha);
        }
        if (Input.GetKeyDown(KeyCode.E))
        {
            PickupWeapons(hit.transform);
        }
    }
    else
    {
        textMeshPro.color = new Color(255, 255, 255, 0);
        picturealpha = 0;
```

Programski kod 13. PlayerPickup - 2.dio

U funkciji PickupWeapons(), prva funkcija provjerava je li objekt u svijetu oružje, te se stvara u inventaru kao jedna stavka, odnosno jedan predmet. Ako je navedena stavka tipa potrošnog materijala (eng. Consumable), kao što su metci i prva pomoć, tada se takvi predmeti klasificiraju kao potrošni. Nakon što igrač pokupi objekt se briše odnosno uništava i njegova se ikona stvara u inventaru.

Predmeti su stvoreni metodom objekta koji se može skriptirati (eng. scriptable object). Svaki predmet u sebi ima klasu Items, koja je tipa skriptirajućeg objekta i sadržava nekoliko varijabli, poput imena, opisa, ikone, objekta u igri, informacije je li potrošni materijal i tipa predmeta. Tip predmeta je javna metoda tipa enum, koja se definira kao poseban tip podataka korišten za definiranje skupa imenovanih konstanti. Enumeracija olakšava korištenje smislenih naziva za određene vrijednosti umjesto standardnih metoda kao što su int i float. Programski kod 14 prikazuje funkciju PickupWeapons().

```
private void PickupWeapons(Transform hit)
{
    if (hit.transform.GetComponent<ItemObject>().item as Weapons)
    {
        Items newItem = hit.transform.GetComponent<ItemObject>().item as Items;
        inventoryManager.AddItem(newItem);
    }
    else
    {
        Consumable newItem = hit.transform.GetComponent<ItemObject>().item as Consumable;
        if (newItem.types == ConsumableType.Ammo)
        {
            inventoryManager.AddItem(newItem);
        }
        else if (newItem.types == ConsumableType.Medkit)
        {
            inventoryManager.AddItem(newItem);
        }
    }
    Destroy(hit.transform.gameObject);
```

Programski kod 14. PlayerPickup - 3.dio

Unity skriptirani objekti su specijalna vrsta objekata u Unity okruženju koji omogućuju kreiranje prilagođenih resursa koji se mogu koristiti za pohranu i dijeljenje podatak između različitih skripti i scena. Oni su posebni zato što omogućuju skladištenje podataka što olakšava njihovo upravljanje i korištenje. Programski kod 15 prikazuje klasu Items.

```
public class Items : ScriptableObject
{
    public string ItemsName;
    public string description;
    public Sprite icon;
    [Header("Gameobject prefab")]
    public GameObject prefab;

    [Header("UI ONLY")]
    public bool stackable = true;

    [Header("Gameplay ONLY")]
    public ItemType type;

}

5 references
public enum ItemType
{
    Weapon,
    Ammo,
    Consumable
```

Programski kod 15. Skriptirani objekt

Klasa Weapons nasljeđuje sve atribute iz klase Items te dodaje svoje, kao što su količina štete koju oružje daje, veličina spremnika za oružje, brzina paljbe i slično. S druge strane, klasa Consumables opisuje sve predmete u svijetu koji su potrošni materijal. Programski kod 16 prikazuje klasu Weapons.

```
public class Weapons : Items
{
    [Header("Weapons only")]
    public int damage;
    public int magazineSize;
    public int storedAmmo;
    public float fireRate;
    public float range;
    public WeaponType weaponType;

    [Header("ZoomInOut")]
    public bool zoomInOut = false;
```

Programski kod 16. Klasa Weapons

5.1.4. Inventar

Inventar je vrlo bitan dio igre i korisničkog sučelja igrača jer pruža mogućnost spremanja, pomicanja i organiziranja svih prikupljenih predmeta u jednostavnom prikazu. Ključne karakteristike inventara su utorci (eng. slots), kako bi se predmeti organizirali i prikazali koji igrač posjeduje. Igrač može dodavati predmete iz svijeta tako da ih prikuplja tijekom igre i uklanjati ih isto tako. Svaki utor može držati jedan predmet ili pet potrošnih materijala kao što su metci ili tablete za obnovu životne ljestvice.

U ovoj igri, inventar je ograničen na 24 utora, stoga je potrebno organizirati i razmisliti o tome što će igrač pokupiti, odnosno zadržati, a što će baciti ili potrošiti. Inventar se prikazuje igraču putem grafičkog korisničkog sučelja, gdje se mogu pregledavati, premještati i iskoristiti neki od predmeta.

Inventar je podijeljen na dva dijela. Donji dio inventara je uvijek vidljiv i u njemu su oružja koja se mogu koristiti, to je inventar aktivna traka (eng. hotbar) koja se sastoji od 6 utora. Hotbar je namijenjen brzom korištenju oružja ili drugih potrošnih materijala kada se igrač nalazi u nevolji. Gornji dio inventara je takozvani glavni dio, u kojem se nalazi preostalih 18 utora. Pritiskom tipke TAB na tipkovnici prikazuje se glavni dio inventara, te predmeti koji su

prikupljeni u igri se spremaju kronološkim redom, prvo u hotbar, a zatim u glavni inventar. Na slici 3 je prikaz inventara, a na slici 4 prikaz hotbar-a.

Slika 3. Prikaz Inventara sa nekoliko predmeta

Slika 4. Prikaz hotbar-a sa nekoliko predmeta

Na slici je prikazan glavni dio inventara i hotbar-a s nekoliko prikupljenih stvari iz svijeta. Za implementaciju inventara korištena je povuci i ispusti (eng. drag & drop) funkcionalnost koja omogućuje igračima interaktivno manipuliranje predmetima u inventaru pomoću miša, tako što mogu povući predmete iz jednog mjesta u inventaru i ispustiti ih na drugo mjesto. U sljedećem dijelu opisat će se povuci i ispusti funkcionalnost kroz kod.

Svaki predmet sadrži skriptu koja ima sljedeće funkcionalnosti. Korištenjem sustava događaja (eng. UnityEngine.EventSystems) prostora imena (eng. namespaces) omogućeno je rukovanje događajima (eng. events) koji se odnose na korisničko sučelje, što omogućuje interakciju igrača s grafičkim elementima kao što su gumbi, trake, padajući izbornici i drugi slični elementi.

Na programskom kodu 17 je prikazana implementacija korištenja sustava događaja, uključujući tri dodatne varijable: IBeginDragHandler, IDragHandler, IEndDragHandler. Koje omogućuju igraču početak pomicanja stvari kroz utore u inventaru klikom lijevog miša i pomicanje kroz inventar.

```
using UnityEngine.EventSystems;
using UnityEngine.UI;

Unity Script (1 asset reference) | 21 references
public class InventoryItem : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
```

Programski kod 17. InventoryItem - 1.dio

Definirane su tri javne metode koje omogućuju kretanje i pomicanje predmeta kroz inventar. Prva funkcija omogućuje, pritiskom miša na predmet, njegovo uzimanje odnosno prikupljanje. Druga funkcija omogućuje predmetu pomicanje kroz prostor u grafičkom korisničkom sučelju, a zadnja funkcija omogućuje predmetu pad, odnosno prihvaćanje u novi utor u inventaru. Programski kod 18 prikazuje 3 javne metode.

```
public void OnBeginDrag(PointerEventData eventData)
{
    image.raycastTarget = false;
    parentAfterDrag = transform.parent;
    transform.SetParent(transform.root);
}

0 references
public void OnDrag(PointerEventData eventData)
{
    transform.position = Input.mousePosition;
}

0 references
public void OnEndDrag(PointerEventData eventData)
{
    image.raycastTarget = true;
    transform.SetParent(parentAfterDrag);
}
```

Programski kod 18. InventoryItem - 2. dio (javne metode pomicanja)

Svaki utor u inventaru ima skriptu InventorySlot koji omogućuje njegovoj djeci odnosno predmetima koji se nalaze u tom utoru, da se ih može manipulirati odnosno pomicati. Programski kod 19 prikazuje funkciju OnDrop.

```
public void OnDrop(PointerEventData eventData)
{
    if(transform.childCount == 0)
    {
        InventoryItem inventoryItem = eventData.pointerDrag.GetComponent<InventoryItem>();
        inventoryItem.parentAfterDrag = transform;
    }
}
```

Programski kod 19. InventorySlot

Nakon spomenutih pomoćnih skripti objasnit će se voditelj svih navedenih operacija zvanog upravitelj inventara (eng. InventoryManager). Upravitelj ima kontrolu nad svim operacija koje se odvijaju tokom igre. Na početku je definirano polje svih utora za inventar kroz koje će se odvijati sve funkcije u inventaru. Također su definirani svi popratni elementi za ispravno funkcioniranje inventara kao što su igrač, predmeti, utori i ostalo. Programski kod 20 prikazuje upravitelja inventara.

```
public class InventoryManager : MonoBehaviour
{
    public InventorySlot[] inventorySlots;

    public GameObject Hotbar;
    public GameObject Inventory;
    public GameObject Crosshair;

    public Items currentlySelectedItem;
    public InventorySlot inventorySlot;
    public UIManager uiManager;

    public GameObject player;

    public GameObject inventoryItemPrefab;

    public int maxStackOnItems = 5;
    public bool isInventoryOn;
```

Programski kod 20. InventoryManager - 1.dio

Funkcija ChangeSelectedSlot() mijenja boju utora kako bi igrač znao na kojem se utoru trenutno nalazi i taj se odabire kao aktivan. Programski kod 21 prikazuje funkciju ChangeSelectedSlot().

```
void ChangeSelectedSlot(int newValue)
{
    if (selectedSlot >= 0)
    {
        inventorySlots[selectedSlot].Deselect();
    }

    inventorySlots[newValue].Select();
    selectedSlot = newValue;
```

Programski kod 21. InventoryManager - 2.dio

Funkcija AddItem(Items item) prima atribut predmeta kao novi koji će se dodati u inventar. Na početku se provjerava ako bilo koji utor sadrži predmet koji ćemo dodati, provjerava se u

njegovom djetetu ako sadrži komponentu tipa Inventoryitem i na kraju se dodaje predmet na predmet (eng. itemstacking). Trenutni maksimalni (eng. itemstacking) je pet predmeta u jednom utoru. Programski kod 22 prikazuje funkciju AddItem().

```
public bool AddItem(Items item)
{
    //Provjeri ako bilo koji inventory slot ima isti item
    for (int i = 0; i < inventorySlots.Length; i++)
    {
        InventorySlot slot = inventorySlots[i];
        InventoryItem itemInSlot = slot.GetComponentInChildren<InventoryItem>();
        //ako nije null i ako je isti item koji trazimo i manji od max numbera
        //onda dodaj item na item (itemstack)
        if (itemInSlot != null && itemInSlot.item == item &&
            itemInSlot.count < maxStackOnItems && itemInSlot.item.stackable == true)
        {
            if (item.type == ItemType.Ammo)
            {
                weaponShooting.InitAmmoSecondaryMagazine(item as Consumable);
            }
            itemInSlot.count++;
            itemInSlot.RefreshCount();
            return true;
```

Programski kod 22. InventoryManager - 3.dio (dodavanje predmeta)

Drugi dio te iste funkcije provjerava ako utor ne sadrži nikakav predmet te se taj isti predmet dodaje kao dijete u utoru tipa InventoryItem, koji je popraćen funkcijom SpawnNewItem(). Programski kod 23 prikazuje dio funkcije AddItem().

```
for (int i = 0; i < inventorySlots.Length; i++)
{
    selectedInventorySlot = i;
    InventorySlot slot = inventorySlots[i];
    // Debug.Log("inv slot: " + i);
    InventoryItem itemInSlot = slot.GetComponentInChildren<InventoryItem>();
    if (itemInSlot == null)
    {
        SpawnNewItem(item, slot);
        return true;
```

Programski kod 23. InventoryManager - 4.dio (dodavanje predmeta)

U funkciji SpawnNewItem() uzimaju se dvije prije spomenute varijable a to su predmet i utor. Nakon toga se inicijalizira novi predmet tipa GameObject i dodaje se taj predmet u pomoćnu varijablu kako bi se kasnije mogao koristiti u igri i kroz drugu skriptu. Programski kod 24 prikazuje spomenutu funkciju.

```
void SpawnNewItem(Items item, InventorySlot slot)
{
    GameObject newItemGameObject = Instantiate(inventoryItemPrefab, slot.transform);
    InventoryItem inventoryItem = newItemGameObject.GetComponent<InventoryItem>();
    inventoryItem.InitialiseItem(item);
    currentlySelectedItem = item;
    inventorySlot = slot;
```

Programski kod 24. InventoryManager - 5.dio (dodavanje predmeta)

U posljednjoj funkciji u upravitelju inventara je objašnjena logika korištenja predmeta. Ako je predmet tipa potrošni materijal na primjer tablete za vraćanje života, te ako se nalaze tri komada u našem inventaru, troše se sukladno prema potrebama igrača. Ako se potroši zadnja tableta, nakon toga se brišu iz našeg inventara i time oslobode jedan utor. Programski kod 25 prikazuje funkciju korištenja predmeta.

```
public Items GetSelectedItem(bool use)
{
    InventorySlot slot = inventorySlots[selectedSlot];
    InventoryItem itemInSlot = slot.GetComponentInChildren<InventoryItem>();
    if (itemInSlot != null)
    {
        Items item = itemInSlot.item;
        if(use == true)
        {
            itemInSlot.count--;
            if(itemInSlot.count <= 0)
            {
                Destroy(itemInSlot.gameObject);
            }
            else
            {
                itemInSlot.RefreshCount();
            }
        }

        return itemInSlot.item;
```

Programski kod 25. InventoryManager - 6.dio (korištenje predmeta)

5.1.5. Korištenje predmeta u igri

Kako bi se predmeti mogli koristiti u igri potrebno ih je inicijalizirati odnosno stvoriti u svijetu. U sljedećoj skripti će biti objašnjeno kako se predmeti stvaraju u 3D formi kako bi ih igrač mogao koristiti. Programski kod 26 prikazuje funkciju EquipWeapon().

```
public void EquipWeapon(Weapons weapon)
{
    if (WeaponHolder.GetComponentInChildren<Animator>() != null)
    {
        UnequipWeapon();
    }
    else if (inventoryManager.GetCurrentlySelectedWeapon() != null)
    {
        currentlyEquipedWeapon = Instantiate(weapon.prefab, WeaponHolder);
        playerstats.UpdateWeaponUI(weapon);
```

Programski kod 26. EquipmentManager

U funkciji EquipWeapon() uzima se parametar oružja kao ulazni argument. Funkcijom se oružje kreira (eng. Instantiate) u obliku Unity objekta u kojem su pohranjene sve informacije i skripte kako bi se kreirao 3D model u svijetu. Oružje se kreira u 'transform' komponenti koja sadrži informacije o trenutnim koordinatama igrača i poziciji kako bi se objekt kreirao na ispravnom mjestu.

5.1.6. Pucanje iz oružja

Kako bi kreirani objekt imao smisla, potrebno je implementirati funkcionalnost i logiku za upravljanje oružjem. Stoga će to biti prikazano u nekoliko sljedećih funkcija. Velik broj privatnih i javnih varijabli definiran je prije početka same skripte kako bi funkcionalnost sa svim elementima bila ispravna. Programski kod 27 prikazuje klasu WeaponShooting.

```
[SerializeField] private Camera mainCam;
[SerializeField] private InventoryManager inventoryManager;
[SerializeField] private EquipmentManager equipmentManager;
[SerializeField] private PlayerStats playerStats;
[SerializeField] private GameObject bloodParticles;

private HealthScript healthScript;
private WeaponHandler weaponHandler;
private float lastShootTime = 0f;
public bool canReload = true;

[SerializeField] private bool canShoot = true;
[SerializeField] public GameObject[] inventorySlots;
[SerializeField] private int CurrentAmmoStorage = 0;
[SerializeField] private bool weaponIsEmpty = false;
[SerializeField] private Animator animator;
```

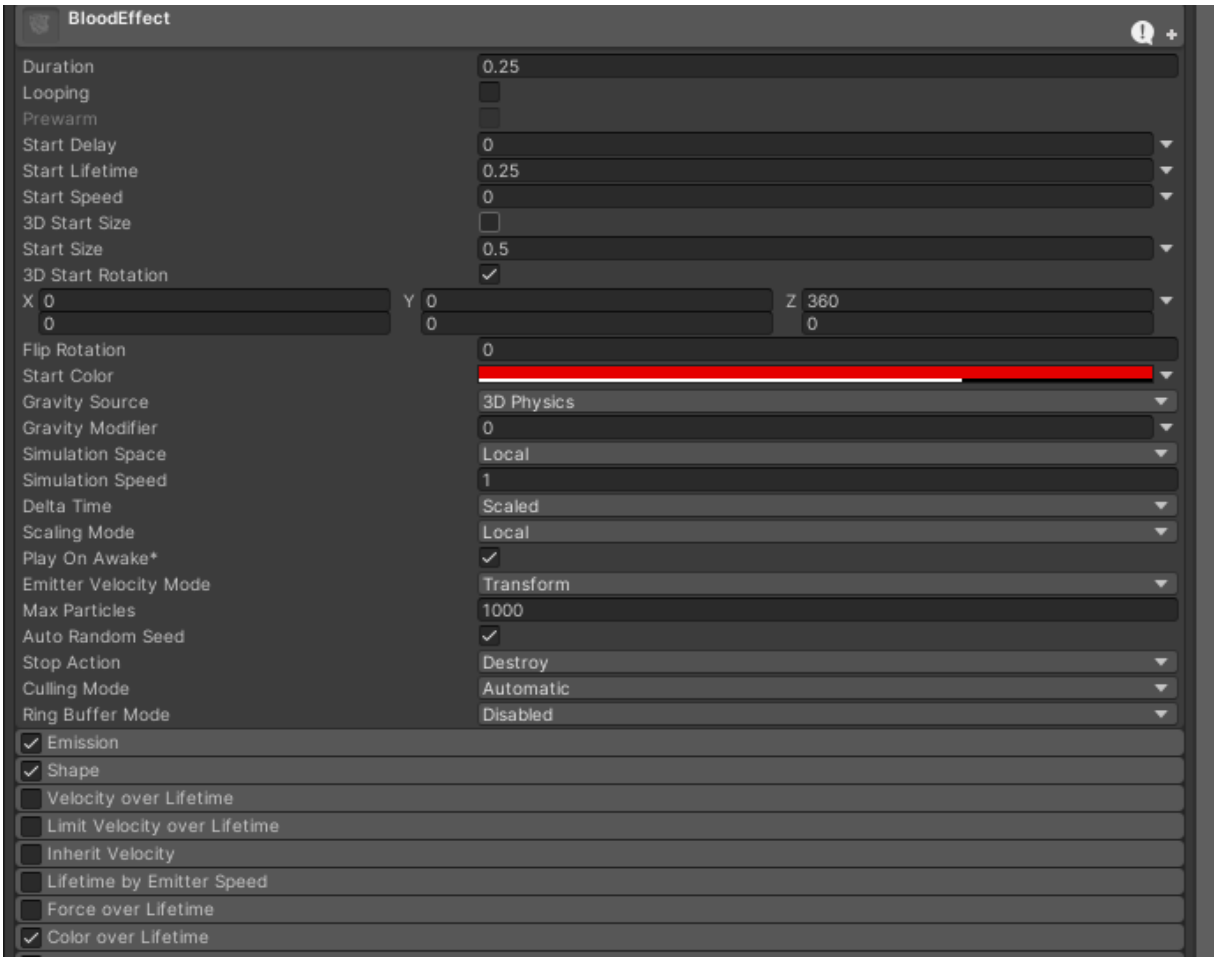
Programski kod 27. WeaponShooting - 1.dio

Nakon definiranja svih početnih varijabli, u sljedećem nastavku bit će objašnjena funkcionalnost skripte. Svako oružje i predmet u igri ima definiran domet, količinu štete koju može prouzročiti i ostale korisne informacije. U početnoj funkciji Physics.Raycast uzima se taj domet, nakon čega se provjerava je li došlo do kolizije s neprijateljem. Ako je došlo do kolizije s objektom koji sadrži tag "Enemy", dohvaća se skripta HealthScript, u kojoj se uzima određeni dio ljestvice života, odnosno količina štete koju je oružje prouzrokovalo. U toj točki na neprijatelju se stvaraju takozvane čestice (eng. particles) koje izgledaju kao krv, kako bi doživljaj pucanja bio realniji. Nakon toga se poziva skripta EnemyController, koja reproducira zvuk zombija kao da je ozlijeđen, kako bi se dodao efekt realnosti. Programski kod 28 prikazuje funkciju RayCastShoot().

```
private void RayCastShoot(Weapons currentWeapon)
{
    Ray ray = mainCam.ScreenPointToRay(new Vector3(Screen.width / 2, Screen.height /2));
    RaycastHit hit;

    if(Physics.Raycast(ray, out hit , currentWeapon.range))
    {
        if(hit.transform.tag == "Enemy")
        {
            HealthScript healthScript = hit.transform.GetComponent<HealthScript>();
            healthScript.ApplyDamage(currentWeapon.damage, hit.transform);
            //Spawn blood particles
            SpawnBloodParticles(hit.point, hit.normal);
            EnemyController enemyController = hit.transform.GetComponent<EnemyController>();
            enemyController.CallPlayZombieHurtSound();
```

Programski kod 28. WeaponShooting - 2.dio

Unity sustav čestica je snažan i fleksibilan alat koji omogućuje developerima stvaranje i kontroliranje raznih vizualnih efekata. Ove čestice mogu simulirati različite fenomene poput vatre, dima, iskri, kiše i još mnogo toga. Sustav čestica ima jako širok opseg mogućnosti, zbog čega ih je na slici prikazano malo. Svaka od njih ima svoj položaj, brzinu, trajanje i druga svojstva. Sustav čestica upravlja velikim brojem čestica istovremeno, stoga je bitno znati koristiti takav sustav da se ne naruše performanse igre. Emiter je jedna od komponenata koja određuje gdje i kako će se čestice generirati. Postavljanjem pozicije, brzine, kutova i sličnih parametara može se kontrolirati raspored i ponašanje čestica. Preinake su komponente koje utječu na čestice tijekom njihovog života, pa preinake mogu mijenjati brzinu, boju, veličinu, rotaciju i druga svojstva čestica. Collideri su komponente koje omogućuju česticama da reagiraju na okolinu, a najbolji primjer za takvo korištenje je, na primjer, kiša ili snijeg. Na slici 5 je prikazan sustav čestica.



Slika 5. Prikaz sustava čestica

U funkciji Shoot() provjerava se može li igrač pucati iz oružja tako što se analizira vremenski razmak između zadnjeg ispaljenog metka i sljedećeg metka. Ako je dovoljno vremena prošlo, tada se poziva animacija za pucanj i nanosi se šteta neprijatelju. Također, nakon ispaljenog metka, troši se municija putem skripte UseAmmo(). Programski kod 29 prikazuje funkciju Shoot().

```
private void Shoot(int slot)
{
    CheckIfCanShoot(equipmentManager.selectedSlot);
    if (canShoot && canReload)
    {
        Weapons currentWeapon = inventoryManager.GetCurrentlySelectedWeapon();

        if (Time.time > lastShootTime + currentWeapon.fireRate)
        {
            lastShootTime = Time.time;

            RayCastShoot(currentWeapon);
            weaponHandler = GetComponentInChildren<WeaponHandler>();
            weaponHandler.ShootAnimation();
            UseAmmo(slot, 1, 0);
```

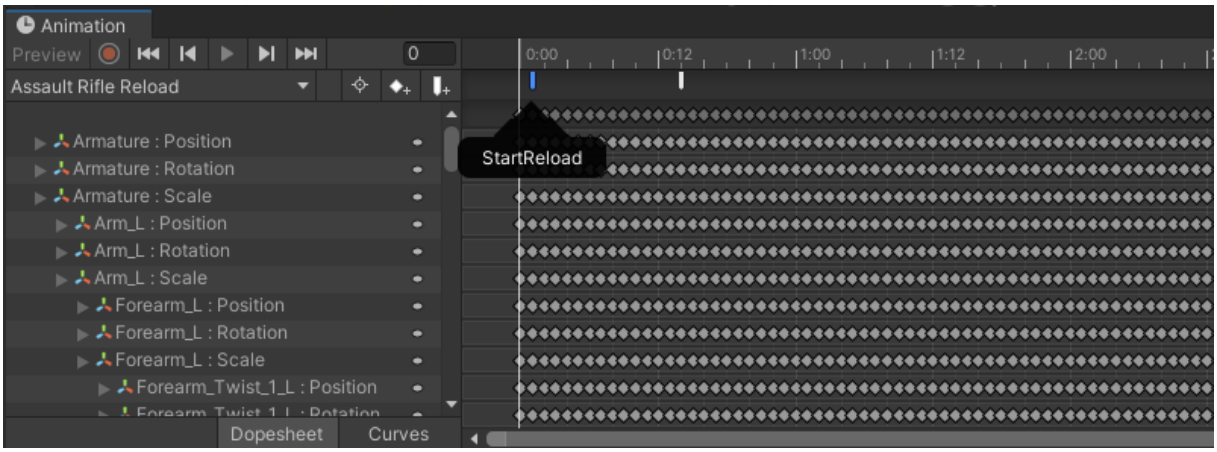
Programski kod 29. WeaponShooting - 3.dio

Funkcija CheckIfCanShoot() provjerava ako postoji dovoljno municije za paljbu. Nakon toga poziva se upravljač animacija koji ne dozvoljava pucanje ako se oružje trenutno puni ili izvlači. Programski kod 30 prikazuju spomenutu funkciju.

```
private void CheckIfCanShoot(int slot)
{
    InventoryItem inventoryItem = inventorySlots[slot].GetComponentInChildren<InventoryItem>();

    if (inventoryItem.ammoCount <= 0)
    {
        canShoot = false;
        weaponIsEmpty = true;
    }
    else
    {
        canShoot = true;
        weaponIsEmpty = false;
```

Programski kod 30. WeaponShooting - 4.dio

U dijelu prozora za animaciju dodaje se animacijski događaj (eng. animation event) koji može služiti kao okidač (eng. trigger) bilo u kodu ili u drugoj animaciji. Takvi događaji su jako korisni kod slučajeva da se zaustavi nepotrebno i neželjeno pucanje ili korištenje oružja u vrijeme animacija. U ovom slučaju okidač se koristi u funkciji StartReload koja u skripti mijenja varijablu CanShoot koja će biti opisana kasnije. Na slici 6 je prikazan prozor za animaciju.



Slika 6. Dio prozora za animaciju ponovnog punjenja oružja

Funkcija Reload() provjerava ima li igrač dovoljno municije za ponovno punjenje oružja te pokreće animaciju i puni oružje. Ako je spomenuta varijabla CanShoot postavljena na 'false', funkcija se ne može pokrenuti jer igrač trenutno ponovno puni oružje.

Ako igrač trenutačno ne puni oružje, zatim se provjerava koje oružje je trenutno aktivno i dohvaća se njegova količina metaka i količina preostale municije u inventaru. Nakon toga se provjeravaju nekoliko slučajeva. Prvi slučaj provjerava ima li igrač dovoljno municije u inventaru za ponovno punjenje oružja i uspoređuje je s trenutnom količinom metaka u oružju. Ako je količina metaka u oružju jednaka količini maksimalnog broja metaka u oružju, odnosno ako je oružje puno, funkcija se preskače. Ako to nije slučaj, pokreće se okidač u animatoru kako bi se pokrenula animacija za ponovno punjenje oružja, a zatim se oduzima količina metaka kojih nedostaje u spremniku, i ta količina oduzima se od ukupne količine municije u inventaru.

U drugom slučaju provjerava se ima li količina municije u inventaru manje od količine metaka koje nedostaju u spremniku oružja, i istovremeno količina nije 0. Nakon toga se pokreće animacija za ponovno punjenje oružja, oduzima se sva dostupna količina municije, stavlja se u spremnik oružja, a zatim se postavlja ukupna količina metaka na 0. Programski kod 31 prikazuje cijelu Reload() funkciju.

```
private void Reload(int slot)
{
    if (canReload)
    {
        InventoryItem inventoryItem = inventorySlots[slot].GetComponentInChildren<InventoryItem>();
        int ammoToReload = inventoryManager.GetCurrentlySelectedWeapon().magazineSize - inventoryItem.ammoCount;
        //ako imamo dovoljno municije za reload
        if (CurrentAmmoStorage >= ammoToReload && CurrentAmmoStorage !>= 0)
        {
            //ako je magazine full
            if (inventoryItem.ammoCount == inventoryManager.GetCurrentlySelectedWeapon().magazineSize)
            {
                Debug.Log("Magazine is already full!");
                return;
            }
            animator.SetTrigger("Reload");
            AddAmmo(slot, ammoToReload, 0);
            UseAmmo(slot, 0, ammoToReload);
            weaponIsEmpty = false;
            CheckIfCanShoot(slot);

        }//reload
        if (CurrentAmmoStorage < ammoToReload && CurrentAmmoStorage != 0)
        {
            int finalReload = CurrentAmmoStorage;
            animator.SetTrigger("Reload");
            AddAmmo(slot, finalReload, 0);
            UseAmmo(slot, 0, finalReload);
            CurrentAmmoStorage = 0;
            weaponIsEmpty = false;
            CheckIfCanShoot(slot);
```

Programski kod 31. WeaponShooting - 5.dio

5.2. Funkcionalnost neprijatelja

Neprijatelji odnosno zombiji su jedna od glavnih značajka ove igre. Prisutni su u svakom trenutku oko igrača čak i kada nije spreman za interakciju. Koriste se 3D modeli i animacije kako bi imali realističan izgled i ponašanje. Zombiji koriste jednostavnu umjetnu inteligenciju kako bi pratili igrača, izbjegavali prepreke i napadali igrača. Način razvijanja logike i sposobnosti kretanja neprijatelja odnosno zombija biti će opisana u nastavku. Na slici 7 su prikazana 2 od nekoliko modela zombija u igri.

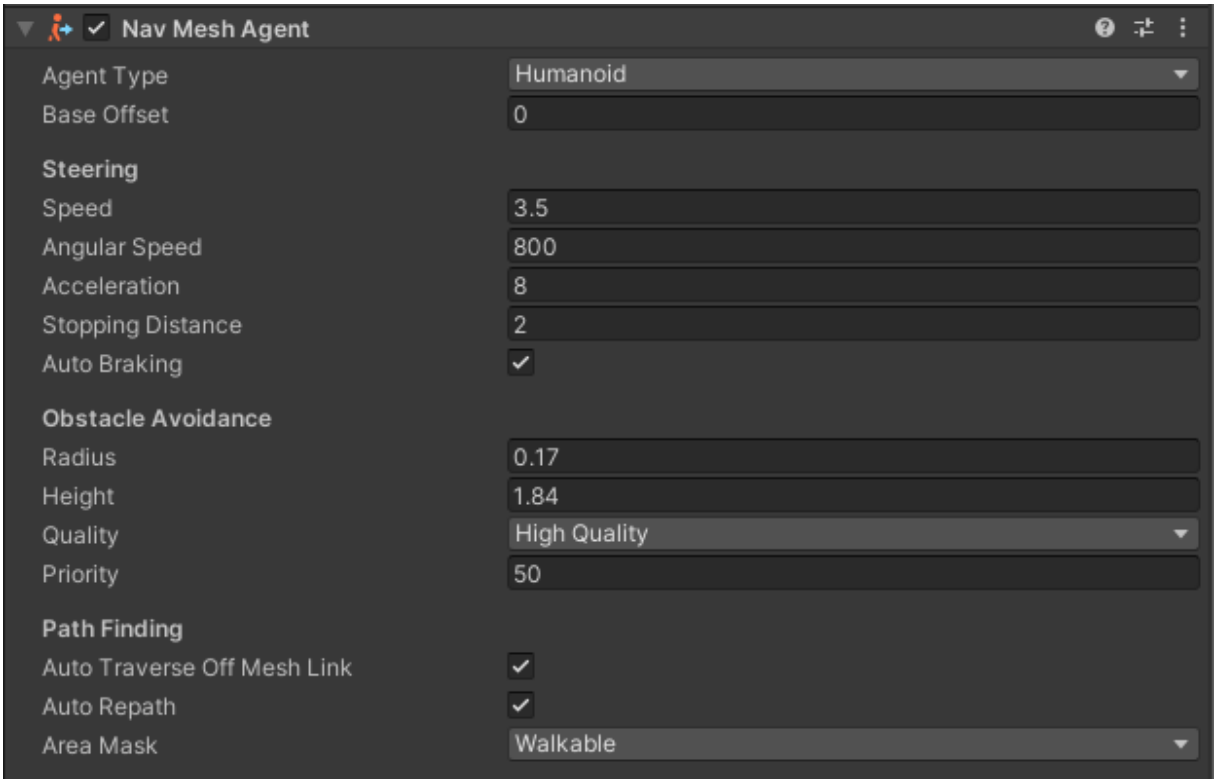
Slika 7. Prikaz dva modela zombija

Svaki od njih ima određene karakteristike poput brzine hoda, veličine ili se pojavljuje u određenoj sceni. U nastavku će biti opisan rad umjetne inteligencije zombija (eng. AI). Svaki objekt koji se kreće i ima neku vrstu umjetne inteligencije sadrži Unity ugrađenu komponentu NavMeshAgent. To je komponenta koja se koristi za stvaranje autonomnih objekata koji se mogu kretati koristeći navigacijsku mrežu (eng. NavMesh). Navigacijsku mrežu u većini slučajeva postavlja se na komponentu 'terrain' i stvara mrežu na tlu po kojem se objekti mogu kretati. Svaki trokut ili poligon u NavMesh-u predstavlja područje kretanja. Neprohodna područja, kao što su zidovi, stijene ili voda, neće biti uključena u NavMesh kako bi ih zombiji izbjegavali.

NavMeshAgent izračunava i slijedi put na NavMesh-u kako bi dosegao svoje odredište. Može rješavati složene navigacijske scenarije, poput kretanja oko prepreka, penjanja po

stepenicama i kretanja po neujednačenom terenu. Ako se prepreka nalazi na putu, on dinamički ponovno računa put do odredišta.

NavMeshAgent komponenta sadrži mnoga svojstva, kao što su brzina kretanja, brzina kretanja uzbrdo i ubrzanje, a koristi se većinom kod vozila. U potkategoriji se nalaze svojstva poput radijusa objekta kako bi mogao izračunati može li objekt proći kroz uske prostore, te visina. Na kraju, komponenta sadrži funkcionalnost pronalaska puta, gdje se mogu konfigurirati različiti slojevi kroz koje se NavMeshAgent može kretati. Na slici 8 je prikazana NavMeshAgent komponenta.



Slika 8. Prikaz NavMeshAgent komponente

Prije svega potrebno je definirati puno varijabli kako bi inteligencija neprijatelja bila što preciznija. Također ovu skriptu potrebno je spojiti sa nekom od postojećih, kao što su dan/noć sistem koji će biti objašnjen kasnije, enemy_Audio koji je zadužen za zvukove neprijatelja i slično. U prilagođenom tipu podataka 'enum' EnemyState definirana su 3 stanja zombija, a to su slobodno lutaj, prati igrača i napadni igrača. Programski kod 32 prikazuje klasu EnemyController.

```
public class EnemyController : MonoBehaviour
{
    [SerializeField] private NavMeshAgent navAgent;
    [SerializeField] private EnemyDestructible enemyDestructible;
    [SerializeField] private UIManager uIManager;
    [SerializeField] private Transform target;
    public DayAndNightSystem dayAndNightSystem;
    public GameObject attack_Point;
    private EnemyAudio enemy_Audio;
    private EnemyAnimatior enemy_Anim;

    private bool isitDay = false;
    private EnemyState enemy_State;

    public float walk_Speed = 0.5f;
    public float run_Speed = 4f;

    public float chase_Distance = 7f;
    private float current_Chase_Distance;
    public float attack_Distance = 1f;
    public float chase_After_Attack_Distance = 2f;
    public bool currentState;

    //radijus kretanja od mjesta spawnanja zombija
    [HideInInspector] public float patrol_Radius_Min = 20f, patrol_Radius_Max = 60f;

    //kolko dugo ce se kretati u jednom smjeru, prije nego dodjelimo drugi smjer
    [HideInInspector] public float patrol_For_This_Time = 15f;
    private float patrol_Timer;
    public float wait_Before_Attack = 2f;
    private float attack_Timer;
```

Programski kod 32. EnemyController - 1.dio

U funkciji Update() provjerava se da li je jedno od stanja trenutno aktivno. Ukoliko je stanje aktivno pokreću se istoimene funkcije. Programski kod 33 prikazuje funkciju Update().

```
void Update()
{
    if(!uIManager.isPaused)
    {
        if (enemy_State == EnemyState.PATROL)
            Patrol();
        if (enemy_State == EnemyState.CHASE)
            Chase();
        if (enemy_State == EnemyState.ATTACK)
            Attack();
```

Programski kod 33. EnemyController - 2.dio

U prvoj funkciji Patrol() provjerava se vremenski razmak, koliko dugo će se zombi kretati do jedne određene točke prije nego što krene prema drugoj. Nakon čega se postavlja nova nasumična destinacija u svijetu. Funkcija SetNewRandomDestionation() provjerava unutar

sebe je li nova destinacija ispravna, odnosno nalazi li se unutar svijeta i nije li previsoka da joj zombi pristupi. Ako je destinacija ispravna, odabire se kao sljedeća, a zombi mijenja smjer kretanja. U slučaju da zombi stoji animacija hoda se ugasila.

Nakon toga provjerava se je li razmak između igrača i zombija dovoljan da bi se on počeo kretati prema igraču, istovremeno postavljajući EnemyState na (eng. chase), odnosno prati igrača. Programski kod 34 prikazuje funkciju Patrol().

```
void Patrol()
{
    ChangeZombieSpeed();
    navAgent.isStopped= false;
    navAgent.speed = walk_Speed;
    patrol_Timer += Time.deltaTime;

    if(patrol_Timer > patrol_For_This_Time)
    {
        SetNewRandomDestination();
        patrol_Timer = 0f;
    }
    //ako se enemy krece
    if(navAgent.velocity.sqrMagnitude > 0)
    {
        enemy_Anim.Walk(true);
    }
    else
    {
        enemy_Anim.Walk(false);
    }
    //testiraj razmak izmedju playera i neprijatelja za chase distance
    if(Vector3.Distance(transform.position, target.position) <= chase_Distance)
    {
        enemy_Anim.Walk(false);
        enemy_State = EnemyState.CHASE;
        enemy_Audio.Play_ScreamSound();

    }
}//Patrol
```

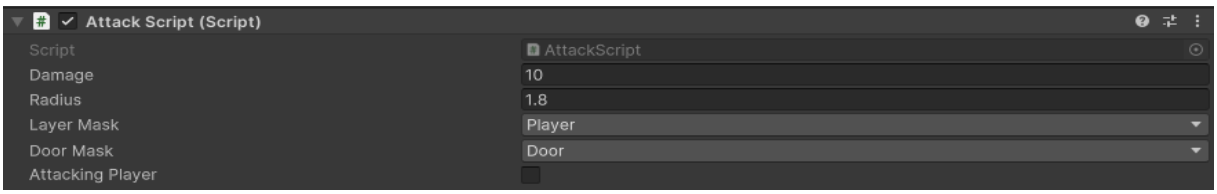
Programski kod 34. EnemyController - 3.dio

U funkciji Chase() postavlja se brzina kretanja zombija na brzinu trčanja kako bi brže stigao do igrača. U funkcijama se uvijek provjerava razmak između igrača i neprijatelja kako bi logika uvijek bila precizna. Ako je razmak između igrača i zombija manji od 1 metra nakon toga se stanje mijenja u napad na igrača. S druge strane ako igrač uspije izaći iz vidnog polja neprijatelja stanje se postavlja na početno odnosno na slobodno kretanje. Programski kod 35 prikazuje funkciju Chase().

```
void Chase()
{
    navAgent.isStopped = false;
    navAgent.speed = run_Speed;

    //postavi poziciju playera kao destinaciju jer lovimo playera
    navAgent.SetDestination(target.position);

    //testiraj razmak izmedju playera i neprijatelja za attack razmak
    if (Vector3.Distance(transform.position, target.position) <= attack_Distance)
    {
        //zaustavi prijasnje animacije
        enemy_Anim.Run(false);
        enemy_Anim.Walk(false);
        enemy_State = EnemyState.ATTACK;

        //ako je igrac van attack ili chase razmaka i upuca neprijatelja, onda ga neprijatelj vidi
        if(chase_Distance != current_Chase_Distance)
        {
            chase_Distance = current_Chase_Distance;
        }

    }
    //igrac izadje van vidika od neprijatelja
    else if (Vector3.Distance(transform.position, target.position) > chase_Distance)
    {
        enemy_Anim.Run(false);
        enemy_State = EnemyState.PATROL;

        //resetiraj patrol timer da funkcija moze kalkulirati novu patrol destinaciju
        patrol_Timer = patrol_For_This_Time;

        if (chase_Distance != current_Chase_Distance)
        {
            chase_Distance = current_Chase_Distance;
```

Programski kod 35. EnemyController - 4.dio

U zadnjoj funkciji logike neprijatelja provjerava se je li zombi u dovoljno malom razmaku od igrača, nakon čega se uključuje animacija napada. U Animaciji napada postavljena je točka nazvana AttackPoint(), koja je zapravo prazni objekt. Sadrži istoimenu kratku skriptu koja se aktivira tijekom sredine animacije i nanosi štetu igraču. Moguće je podesiti količinu štete i domet napada unutar ove skripte. Slika 9 prikazuje postavke skripte AttackScript().



Slika 9. Prikaz postavka skripte AttackScript()

Funkcija Attack() sadrži nekoliko dodatnih svojstava koja osiguravaju jasnu logiku neprijatelja. Nakon pokretanja spomenute funkcije, bilo kakvo kretanje neprijatelja zaustavlja se. To pruža igraču malu prednost i omogućuje mu pokušaj izbjegavanja napada neprijatelja. Nakon toga provjerava se vremenski razmak između prvog napada i vremena do sljedećeg napada kako

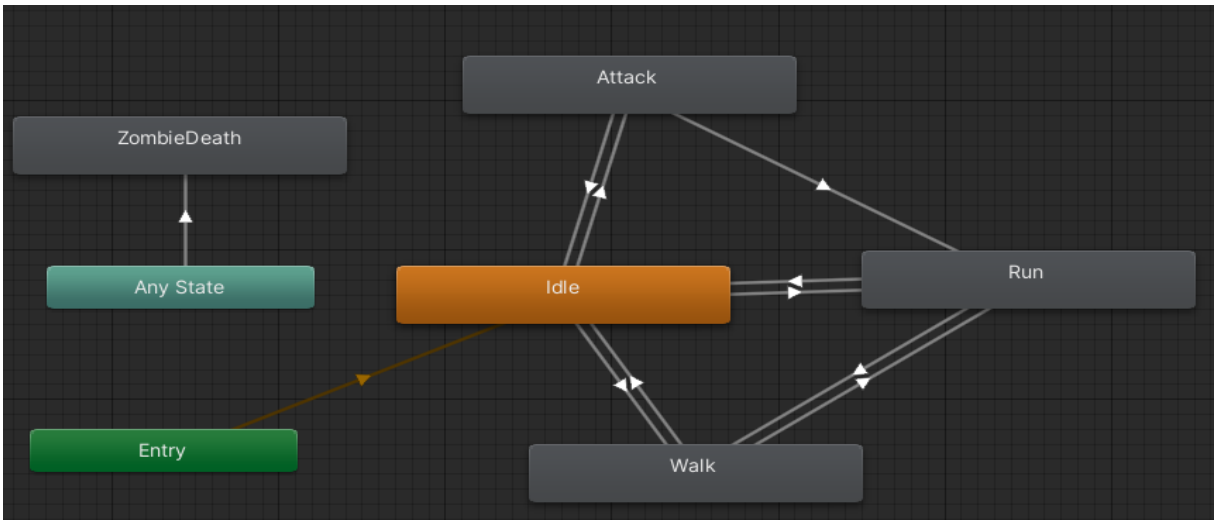
ne bi došlo do neprestanog napadanja igrača. U slučaju da je moguće napasti, pokreće se animacija za napad i reproducira zvuk zombija za dodatan efekt. Ako se igrač uspije udaljiti iz zone napada, status zombija mijenja se u Chase(). Programski kod 36 prikazuje Attack() funkciju.

```
void Attack()
{
    navAgent.velocity = Vector3.zero;
    navAgent.isStopped = true;
    attack_Timer += Time.deltaTime;
    if(attack_Timer > wait_Before_Attack)
    {
        enemy_Anim.Attack();
        attack_Timer = 0;
        enemy_Audio.Play_AttackSound();
    }
    //dajemo malu prednost igracu kako bi lakse pobjegao sa ovim + chase_After_Attack_Distance
    //kako neprijatelj nebi odma poceo trcati
    if (Vector3.Distance(transform.position, target.position) > attack_Distance + chase_After_Attack_Distance)
    {
        enemy_State = EnemyState.CHASE;
    }
```

Programski kod 36. EnemyController - 5.dio

Sve već spomenute animacije upravljaju se u komponenti 'animator controller'. Animator kontroler omogućuje definiranje i organiziranje različitih animacijskih prijelaza i stanja objekta kako bi se postigla glatka i fluidna animacija tijekom igre. Animator kontroler sastavljen je od različitih stanja (eng. states) koja predstavljaju specifične animacije ili animacijske sekvence koje lik može izvoditi. Kao na primjer, u ovom slučaju na početku igre pokreće se animacija 'idle', odnosno stanje mirovanja. Nakon toga putem okidača u kodu pokreću se ostale animacije poput 'attack', 'run' ili 'walk'. Animacija za smrt zombija može se aktivirati tijekom prikazivanja bilo koje druge animacije, zbog toga je posebno postavljena.

Sve animacije mogu se prekinuti i pokrenuti u određenom trenutku, a njima upravlja skripta EnemyController. Također, animator kontroler može se primijeniti na bilo kojem neprijatelju ili bilo kojem objektu, te se svaki od njih može dodatno konfigurirati. Slika 10 prikazuje animator kontroler.



Slika 10. Prikaz upravljača animacija

5.2.1. Upravljanje stvaranjem neprijatelja

Zombiji i divlje svinje se u svijetu stvaraju putem skripte. Kako ne bi došlo do prekomjernog stvaranja objekata u svijetu, a time narušila funkcionalnost i optimiziranost igre, potrebno je uključiti upravljač stvaranjem neprijatelja. Na početku same skripte definira se javna statička varijabla imena instance tipa EnemyManager. Ključna riječ 'static' znači da je varijabla zajednička za sve instance klase EnemyManager. Postoji samo jedna varijabla instance za cijelu igru, bez obzira na to koliko objekata EnemyManager je stvoreno. Svrha ove varijable je implementacija obrasca dizajna poznatog kao Singleton. Obrascu Singleton osigurava da klasa ima samo jednu instancu i pruža globalnu točku pristupa toj instanci. Nakon toga je definirano polje objekata koji se mogu stvoriti, odnosno zombija, te teren i UI upravitelj. Programski kod 37 prikazuje klasu EnemyManager.

```
public class EnemyManager : MonoBehaviour
{
    public static EnemyManager instance;
    [SerializeField] private GameObject[] spawnableEnemy;
    [SerializeField] private GameObject spawnableBoar;
    [SerializeField] private Terrain terrain;
    [SerializeField] private NavMeshSurface navMeshSurface;
    [SerializeField] private UIManager uiManager;
    public Transform player;
    private int spawnableEnemyCount = 15;
```

Programski kod 37. EnemyManager - 1.dio

Funkcija SpawnEnemy() na početku provjerava je li trenutni broj neprijatelja manji od maksimalnog broja neprijatelja koji se mogu stvoriti. Nakon toga se poziva funkcija GetRandomPosition() u kojoj se provjerava nasumična lokacija u određenom radijusu od igrača koja vraća spomenutu lokaciju tipa Vector3. Radijus stvaranja je bitan kako ne dolazi do stvaranja neprijatelja u igraču ili predaleko u svijetu. Programski kod 38 prikazuje funkciju SpawnEnemy().

```
private Vector3 GetRandomPosition()
{
    Vector3 randomDirection = Random.insideUnitSphere;
    randomDirection.y = 0f;
    randomDirection.Normalize();

    float randomDistance = Random.Range(minSpawnRadius, maxSpawnRadius);
    Vector3 randomPosition = player.transform.position + randomDirection * randomDistance;

    return randomPosition;
```

Programski kod 38. EnemyManager - 2.dio

Zatim se dobije visina terena na toj poziciji kako se neprijatelj ne bi stvorio ispod svijeta. Također se izračuna kut nagiba terena koristeći funkciju Vector3.Angle i normale terena. Ukoliko je kut manji ili jednak 45 stupnjeva, teren je dovoljno ravan za stvaranje neprijatelja. Na kraju, pomoću funkcije NavMesh.SamplePosition provjerava se je li lokacija unutar NavMesh polja, te se nasumično odabire jedan od spremljenih objekata neprijatelja koji se postavi na odabranu lokaciju. Programski kod 39 prikazuje ostatak funkcije SpawnEnemy().

```
private IEnumerator SpawnEnemy()
{
    if (currentEnemyCount < spawnableEnemyCount)
    {
        yield return new WaitForSeconds(4);
        Vector3 randomPosition = GetRandomPosition();
        float terrainHeight = terrain.SampleHeight(randomPosition);
        Vector3 terrainNormal = terrain.terrainData.GetInterpolatedNormal(randomPosition.x
            / terrain.terrainData.size.x, randomPosition.z / terrain.terrainData.size.z);
        float slopeAngle = Vector3.Angle(Vector3.up, terrainNormal);
        if(slopeAngle <= 45)
        {
            randomPosition.y = terrainHeight;

            NavMeshHit hit;
            if(NavMesh.SamplePosition(randomPosition, out hit, 1.0f, NavMesh.AllAreas))
            {
                GameObject agent = Instantiate(spawnableEnemy[Random.Range(0, spawnableEnemy.Length)]
                    , randomPosition, Quaternion.identity);
                NavMeshAgent navAgent = agent.GetComponent<NavMeshAgent>();
                if (navAgent != null)
                {
                    navAgent.Warp(randomPosition);
                }
                currentEnemyCount++;
```

Programski kod 39. EnemyManager - 3.dio

5.2.2. Dan/noć ciklus

Ciklus dan/noć je jedna od važnih i ključnih značajki igre, kao što je već objašnjeno. Po danu, brzina kretanja zombija je duplo sporija u odnosu na brzinu kretanja po noći, a njihova vidljivost je malo povećana tijekom noći, dok je smanjena tijekom dana. Sljedeća skripta je povezana s objektom sunce u sceni kako bi kontrolirala njegovu rotaciju oko svijeta. U početku igre, objekt sunce je pozicioniran tako da izgleda kao jutro, a igra počinje u šest sati ujutro. Također je moguće mijenjati brzinu dnevnog/noćnog ciklusa kako bi igra bila intenzivnija, što mijenja i vrijeme. Stoga se mora primijeniti nekoliko provjera kako bi sunce i vrijeme u igri uvijek započelo u šest sati ujutro. Metoda Start() je prikazana na slici ispod. Programski kod 40 prikazuje početak klase DayAndNightSystem.

```
void Start()
{
    rotationSpeed = 360 / dayLengthMinutes / 60;
    midday = dayLengthMinutes * 60/2;
    if(dayLengthMinutes == 5)
    {
        currentTime = 75f;
    }
    else if(dayLengthMinutes == 10)
    {
        currentTime = 150f;
    }
    else if(dayLengthMinutes == 0.5)
    {
        currentTime = 7.5f;
```

Programski kod 40. DayAndNightSystem - 1.dio

U funkciji Update() izvršava se nekoliko matematičkih funkcija kako bi se vrijeme izračunalo točno. Nakon toga se provjerava je li prošao cijeli dan, odnosno ciklus od 24 sata. Također se provjerava je li dan ili noć, a na kraju se vrijeme ispisuje u obliku teksta u sredini gornjeg dijela zaslona. Nakon toga, u funkciji DayToSpawnHorde() provjerava se je li prošlo 7 dana u igri nakon početka, te se stvara horda zombija koja je brža, jača i fokusirana specifično na traženje igrača u svim mogućim trenucima. Svakim izvršavanjem metode Update() objekt sunce se pomiče za jedan stupanj kako bi promjene bile vidljive u svijetu. Programski kod 41 prikazuje ostatak klase DayAndNightSystem.

```
void Update()
{
    if (!uiManager.isPaused)
    {
        if (!healthScript.IsDead())
        {
            currentTime += Time.deltaTime;
            translateTime = (currentTime / (midday * 2));

            float t = translateTime * 24f;

            float hours = Mathf.Floor(t);
            currentHours = hours;

            string displayHours = hours.ToString();

            if (hours >= 24)
            {
                displayHours = (hours - 24).ToString();
            }

            t *= 60;
            float minutes = Mathf.Floor(t % 60);
            string displayMinutes = minutes.ToString();
            if (minutes < 10)
            {
                displayMinutes = "0" + minutes.ToString();
            }

            DaytoSpawnHorde();

            CheckDayTime();

            string displayTime = "Day: " + day + " Time: " + displayHours + ":" + displayMinutes;

            timeText.text = displayTime;

            transform.Rotate(new Vector3(1, 0, 0) * rotationSpeed * Time.deltaTime);
```

Programski kod 41. DayAndNightSystem - 2.dio

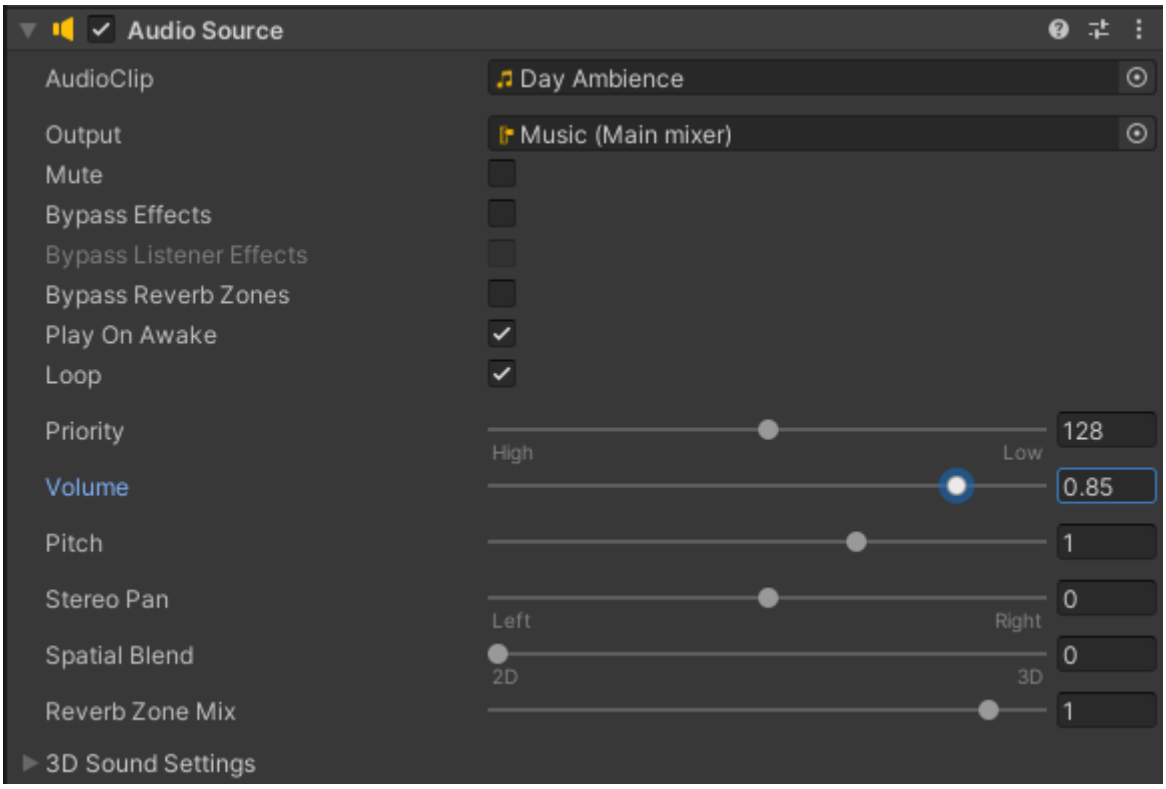
5.3. Upravitelj zvuka

Ambijentalna pozadinska glazba ima ključnu ulogu u industriji igara, kao i u filmskoj industriji. U igrama, ambijentalna glazba pridonosi posebnom ugođaju, stvara trenutke napetosti i poboljšava ukupni doživljaj igre. U nastavku će biti opisan rad skripte zadužene za sveukupni zvučni doživljaj. Skripta AudioManager počinje s definiranjem jednostavnog izvora zvuka (eng. AudioSource). U ovom slučaju, postoje dva izvora zvuka: dnevni, koji je malo tiši i opušteniji, te noćni, koji je agresivniji i jači. Nakon toga, skripta se povezuje s dan/noć sustavom kako bi se glazbeni primjerak ispravno reproducirao tijekom dana ili noći. Programski kod 42 prikazuje početak klase AudioManager.

```
public class AudioManager : MonoBehaviour
{
    public AudioSource dayAudioSource;
    public AudioSource nightAudioSource;

    [SerializeField] private DayAndNightSystem dayTime;
    [SerializeField] private UIManager uiManager;
```

Programski kod 42. AudioManager

Unity AudioSource je komponenta koja omogućuje reprodukciju zvuka. Koristi se za puštanje zvučnih efekata, glazbe ili dijaloga tijekom igre. Ova komponenta obično je povezana s objektom u sceni, poput igrača, ili može biti prazan objekt namijenjen samo reprodukciji zvuka. Struktura komponente sastoji se od AudioClip-a koji služi kao izvor zvuka koji će se reproducirati, output-a koji može biti koristan kada postoje različite vrste zvukova u igri, prioritet-a, glasnoće i drugih značajki. Stoga je za prvi zvuk odabran snimak dnevne ambijentalne glazbe, a za output je odabrana kategorija 'music' u glavnom mikseru, što će biti obrađeno kasnije. Slika 11 prikazuje komponentu AudioSource.



Slika 11. Prikaz AudioSource komponente

5.4. Korisničko sučelje

Korisničko sučelje, poznato kao User Interface (UI), predstavlja sustav unutar Unity okruženja koji omogućava izradu sučelja za igre i aplikacije. Taj sustav omogućava kreiranje različitih elemenata sučelja poput gumba, tekstualnih prozora, slika i drugih grafičkih elemenata koji pojednostavljuju interakciju između korisnika i igre.

Platno, ili (eng. Canvas), služi kao osnovni element korisničkog sučelja, pružajući praznu pozadinu koja sadrži sve druge UI elemente. Na platnu se postavljaju i organiziraju svi grafički elementi. Unity nudi razne unaprijed definirane UI elemente kao što su gumbi, tekstualni prozori, slike, ploče i polja za unos teksta, koji se mogu lako postaviti i prilagoditi putem editora.

UI također uključuje sustav događaja koji elementima dodjeljuje ponašanja kada se dogode specifični događaji poput pritiska na gumb. Platno se može prilagoditi s pomoću komponente 'Canvas Scaler', što omogućava prilagodbu sučelja za različite veličine ekrana i rezolucije, osiguravajući da sučelje održava kvalitetan izgled na različitim uređajima. Svi UI elementi se mogu transformirati u 'gameprefabs', što omogućava njihovu ponovnu upotrebu u različitim scenama ili projektima.

Na priloženoj slici ispod prikazana je igra unutar Unity okruženja s nekoliko UI elemenata: količina života i izdržljivosti u gornjem lijevom kutu zaslona, mini karta u gornjem desnom kutu, otvoreni inventar i hotbar smješteni u središnjem dijelu te dolje desno mjesto rezervirano za prikaz trenutno aktivnog oružja u igri. Osim toga, UI sustav omogućava dinamičko ažuriranje elemenata u stvarnom vremenu, pružajući igračima trenutne povratne informacije. Slika 12 prikazuje izgled zaslona u igri kada se aktivira pauza pritiskom tipke ESC na tipkovnici.

Slika 12. Prikaz igre sa nekoliko UI elemenata

Tokom pauze igra potpuno staje i može se aktivirati ili deaktivirati u bilo kojem trenutku. Prikaz zaslona je sive boje i uključuje se 5 elemenata od kojih su 4 aktivni kao što su gumb za ponovo igranje (eng. restart), polazak u glavni meni (eng. main menu), opcije (eng. options) i gašenje igre (eng. quit game). Na slici 13 prikazana je pauza.

Slika 13. Prikaz zaslona tokom aktivirane pauze u igri

Pritiskom tipke opcije otvara se novi prozor koji ima nekoliko mogućnosti. Igra se može postaviti na igranje u punom zaslonu rada (eng. fullscreen), igranje sa zaključanom brzinom kadrova (eng. limited framerate) i rezolucija zaslona.

Nakon toga su postavke glasnoće zvuka u kojem se može mijenjati glasnoća glavnog zvuka, muzike i specijalnih efekata. U prijašnjem poglavlju u audio upravitelju bilo je objašnjeno kako se mogu koristiti i reproducirati različiti zvukovi u različitim slojevima. Spomenute postavke vidljive su na slici 14 ispod.

Slika 14. Prikaz zaslona opcije

U sljedećem dijelu će biti opisana funkcionalnost i rad postavki putem koda. Na početku su definirane nekoliko varijabli koje su potrebne za funkcionalnost UI elemenata, kao što su prebacivanje (eng. toggle), lista rezolucija koja se može proširiti ako korisnik nema određenu rezoluciju, tekstualni elementi te klizač (eng. slider) kako bi se jednostavno i lijepo mogla podešavati glasnoća zvučnih efekata. Također je definirana nova komponenta Audio mixer koja je zadužena za puštanje ambijentalne glazbe i ostale glazbe u jednom sloju, te ostalih zvučnih efekata kao što su pucanj ili glasanje zombija u drugom sloju. Ovakva podjela zvukova daje korisniku igre mogućnost povećavanja odnosno smanjivanja glasnoće određenih dijelova igre, ako su navedeni elementi preglasni ili pretihi. Programski kod 43 prikazuje početak klase OptionsScreen.

```
public class OptionsScreen : MonoBehaviour
{
    public Toggle fullScreenTog, vSyncTog;
    public List<ResItem> resolutions = new List<ResItem>();
    private int selectedResolution;
    public Text resolutionLabel;
    public AudioMixer theMixer;
    public Text masterLabel, musicLabel, SFXLabel;
    public Slider masterSlider, musicSlider, SFXslider;
```

Programski kod 43. OptionsScreen - 1.dio

U početnoj funkciji Start() se provjeravaju postavke kao što su igranje u punom zaslonu ili igranje sa zaključanom brzinom kadrova. Nakon toga se pretražuje lista rezolucija koje već postoje i odabire se jedna od postojećih. U protivnom ako unesena rezolucija nije u listi dodaje se u listu i odabire se ta ista kao aktivna. Na kraju svakog odabira, tekstualni okvir sa listom rezolucija se ažurira sa trenutno odabranom rezolucijom. Programski kod 44 prikazuje izbor rezolucija.

```
void Start()
{
    fullScreenTog.isOn = Screen.fullScreen;

    if (QualitySettings.vSyncCount == 0)
    {
        vSyncTog.isOn = false;
    }
    else
    {
        vSyncTog.isOn = true;
    }

    bool foundRes = false;
    for(int i = 0; i < resolutions.Count; i++)
    {
        if(Screen.width == resolutions[i].horizontal && Screen.height == resolutions[i].vertical)
        {
            foundRes = true;
            selectedResolution = i;
            UpdateResolutionLabel();
        }
    }
    if(!foundRes)
    {
        ResItem newRes = new ResItem();
        newRes.horizontal = Screen.width;
        newRes.vertical = Screen.height;

        resolutions.Add(newRes);
        selectedResolution = resolutions.Count - 1;
        UpdateResolutionLabel();
```

Programski kod 44. OptionsScreen - 2.dio

U sljedećem dijelu funkcije Start() postavljaju se klizači za glavni dio zvukova, muzike i specijalnih efekata. Svi zvukovi se sastoje od pozitivne i negativne strane zvuka. Samim time

se zbrajaju sa brojem 80 kako bi klizaći bili postavljeni na nulu i sprječavaju korisnika da postavi zvuk manje od nule. Programski kod 45 prikazuje postavke za klizače zvukova.

```
float vol = 0f;
theMixer.GetFloat("MasterVol", out vol);
masterSlider.value = vol;
theMixer.GetFloat("MusicVol", out vol);
musicSlider.value = vol;
theMixer.GetFloat("SFXVol", out vol);
SFXslider.value = vol;

masterLabel.text = Mathf.RoundToInt(masterSlider.value + 80).ToString();
musicLabel.text = Mathf.RoundToInt(musicSlider.value + 80).ToString();
SFXLabel.text = Mathf.RoundToInt(SFXslider.value + 80).ToString();
```

Programski kod 45. OptionsScreen - 3.dio

U funkciji ApplyGraphics() se postavljaju sve promijenjene postavke prema željama korisnika, kako je prikazano na Programskom kodu 46.

```
public void ApplyGraphics()
{
    Screen.fullScreen= fullScreenTog.isOn;

    if(vSyncTog.isOn)
    {
        QualitySettings.vSyncCount = 1;
    }
    else
    {
        QualitySettings.vSyncCount = 0;
    }
    Screen.SetResolution(resolutions[selectedResolution].horizontal,
        resolutions[selectedResolution].vertical, fullScreenTog.isOn);
```

Programski kod 46. OptionsScreen - 4.dio

U funkciji SetMasterVolume() se u igri postavlja klizač na željenu glasnoću od strane korisnika i sprema su putem komponente PlayerPrefs kako bi bila spremljena za kasnije seanse igre. Programski kod 47 prikazuje navedenu funkciju.

```
public void SetMasterVolume()
{
    masterLabel.text = Mathf.RoundToInt(masterSlider.value + 80).ToString();

    theMixer.SetFloat("MasterVol", masterSlider.value);

    PlayerPrefs.SetFloat("MasterVol", masterSlider.value);
```

Programski kod 47. OptionsScreen - 5.dio

5.5. Sustav napredovanja

Sustav napredovanja u igri "A Week To Survive" sastoji se od nekoliko elemenata. Svijet je podijeljen u 4 takozvane zone. Svaka zona sadrži određeni broj tragova, predmeta, vrsta oružja i kratkih filmskih scena.

Igrač se počinje u prvoj zoni na početnoj lokaciji kod kolibe. Potrebno je u prvoj zoni prikupiti što više oružja i korisnih stvari, istražiti okolinu, pronaći tragove i ostale zanimljivosti. Za otključavanje druge zone potrebno je preživjeti prvu horde zombija nakon prvih 7 dana u igri (eng. in-game days). Ako igrač preživi, otključava mu se granica između prve i druge zone koja u igri izgleda kao ograda koju nije moguće preskočiti ili maknuti na drugi način.

Za preostale dvije zone potrebno je također preživjeti 14 dana u igri, pronaći specijalne predmete koji će kasnije biti potrebni za posljednju filmsku scenu koja vodi do finalne zone. Minimalno nakon 21. dana u igri i prikupljenih svih specijalnih predmeta, otključava se finalna zona u kojoj se susrećemo s raspletom radnje, saznajemo što se dogodilo u trenutnom području i dolazimo do konačne istine.

5.5.1. Tragovi

Tragovi (eng. clues) su komadi papira koji sadrže neki koristan ili zbunjujući tekst. U tragovima može biti skrivena destinacija, mjesto na kojem bi se mogao nalaziti specijalan predmet ili priča o nečemu što se dogodilo u igri. Na slici 15 se vidi izgled traga odnosno papira u svijetu koji se može pročitati.

Slika 15. Snimka zaslone iz igre u kojem se nalazi trag

Skripta koja je zadužena za prikupljanje tragova biti će opisana u nastavku. Na početku se definiraju već neki poznati elementi kao što su kamera, 'layermask', tekstualni okvir, objekt na kojem se nalazi trag, lista tragova u igri te neke pomoćne varijable. Programski kod 48 prikazuje početak klase CluePickup.

```
public class CluePickup : MonoBehaviour
{
    [SerializeField] Camera cam;
    [SerializeField] LayerMask clueMask;
    [SerializeField] TextMeshProUGUI interactText;
    [SerializeField] UIManager uiManager;
    [SerializeField] GameObject ClueGameObject;
    [SerializeField] private string[] clueList;

    private float pickupRange = 5f;
    private float picturealpha;
    private int clueListCounter = 0;
```

Programski kod 48. CluePickup - 1.dio

Skriptu započinjemo s korutinom. U korutini se obavlja funkcija Ray koja provjerava ako je igrač u koliziji sa slojem u kojem se nalazi lista tragova, te potom prikazuje tekstualni okvir na sredini zaslona kako bi igrač znao da se radi o tragu s kojim može pročitati. Ako se pritisne tipka E na tipkovnici uključuje se objekt na kojem se nalazi trag i prikazuje se na zaslonu dok

se u međuvremenu igra pauzira od svih ostalih trenutnih aktivnosti. Na kraju svega korutina poziva samu sebe ako nismo pronašli nikakav trag. Programski kod 49 prikazuje funkciju CheckForClues().

```
private IEnumerator CheckForClues()
{
    Ray ray = cam.ScreenPointToRay(new Vector3(Screen.width / 2, Screen.height / 2));
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit, pickupRange, clueMask))
    {
        if (hit.transform.CompareTag(clueList[clueListCounter]))
        {
            if (Input.GetKeyDown(KeyCode.E))
            {
                clueListCounter++;
                uiManager.isPaused = true;
                ClueGameObject.gameObject.SetActive(true);
                uiManager.gameObject.SetActive(false);
            }

            if (picturealpha <= 0)
            {
                picturealpha -= (Time.deltaTime / 10);
                interactText.color -= new Color(0, 0, 0, picturealpha);
            }
        }
    }
    else
    {
        interactText.color = new Color(255, 255, 255, 0);
        picturealpha = 0;
    }
    yield return new WaitForSeconds(0.05f);
    StartCoroutine(CheckForClues());
```

Programski kod 49. CluePickup - 2.dio

Nakon što se aktivirao objekt na kojem se nalazi trag, pokreće se skripta ClueController. Ova skripta je zadužena za lagano pokazivanje prvo samog papira i nakon toga teksta koji se nalazi na tom papiru. Takvo svojstvo se dobiva manipuliranjem odnosno mijenjanjem prozirnosti slike koja je spremljena kao svojstvo alfa (eng. picuture alpha). To svojstvo oduzima putem funkcije Time.deltaTime i dijeli se s brojem 30 kako bi na izgled bilo glatko i nesmetano.

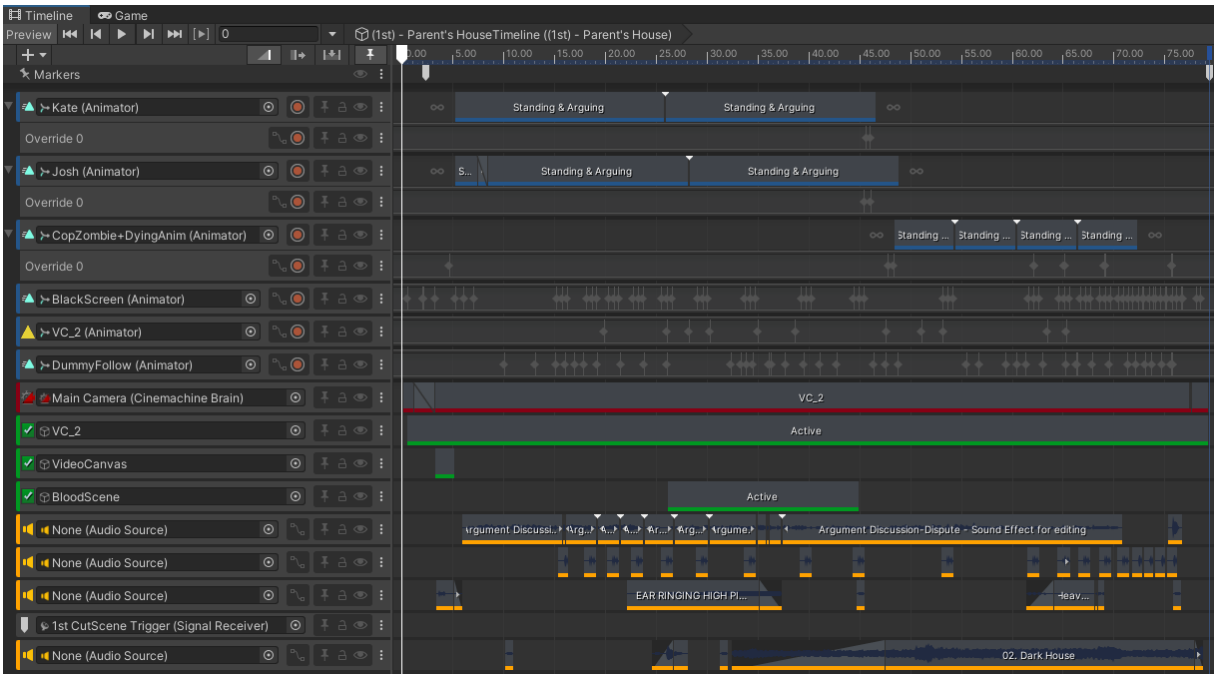
Nakon nekog vremena kad se tekst prikazao do kraja odnosno kad je prozirnost nestala, igraču se omogućuje funkcija SkipText(). U navedenoj funkciji provjerava se ako je igrač pritisnuo tipku ESC ili razmaknicu potom se objekt gasi. Programski kod 50 prikazuje klasu ClueController.

```
private void Update()
{
    if (pictureAlpha1 <= 0)
    {
        pictureAlpha1 -= (Time.deltaTime / 30);
        Background.color -= new Color(0, 0, 0, pictureAlpha1);
        if (pictureAlpha1 <= - 0.08f)
        {
            pictureAlpha2 -= (Time.deltaTime / 40);
            textMeshPro.color -= new Color(0, 0, 0, pictureAlpha2);
            if(pictureAlpha2 <= -0.15f)
            {
                uiManager.gameObject.SetActive(true);
                SkipText();
            }
        }
    }
}

1 reference
private void SkipText()
{
    skipText.color = new Color(0, 0, 0, 1);
    if (Input.GetKeyDown(KeyCode.Space) || Input.GetKeyDown(KeyCode.Escape))
    {
        CluesCanvas.gameObject.SetActive(false);
        uiManager.isPaused = false;
```

Programski kod 50. ClueController

5.5.2. Kratke filmske scene

Kratke filmske scene (eng. cutscenes) su moguće putem alata vremenske trake (eng. Timeline) [5]. To je vizualni alat koji omogućuje stvaranje kompleksnih sekvenca animacija, kamera, događaja i drugih akcija kako bi se upravljali tijekom igre ili u filmskim scenama u Unity razvojnom okruženju. Timeline koristi vizualno sučelje slično onome što se može naći u softveru za uređivanje videozapisa, gdje korisnici mogu povlačiti i ispuštati različite elemente kao što su animacije, zvučni zapisi i skripte duž vremenske crte. Slika 16 prikazuje alat za izradu filmskih scena.



Slika 16. Prikaz Timeline alata unutar Unity-a

Na slici je prikazan "timeline" alat u Unity okruženju koji prikazuje drugu kratku filmsku scenu i sve njene elemente. Vremenska traka je mjesto na kojem se izrađuju i organiziraju animacijski ključevi, događaji i akcije. Različiti slojevi postavljaju se u vremenskoj traci, a svaki sloj predstavlja određenu vrstu animacije ili događaja. Slojevi uključuju animacijske slojeve, audio slojeve, kamere i slično. Svaki sloj služi za organizaciju aspekta scene ili igre.

Ključevi (eng. Keys) su vremenske oznake na vremenskoj traci gdje se postavljaju parametri za animacije ili događaje. Na primjer, na početku vremenske trake postavljen je ključ koji pauzira igru putem skripte.

"Timeline" omogućuje kontrolu vremena unutar vremenske trake. Klizač se može pomicati kako bi se promijenila trenutna točka reprodukcije, reproducirala animacija unatrag, usporila ili zaustavila animacija.

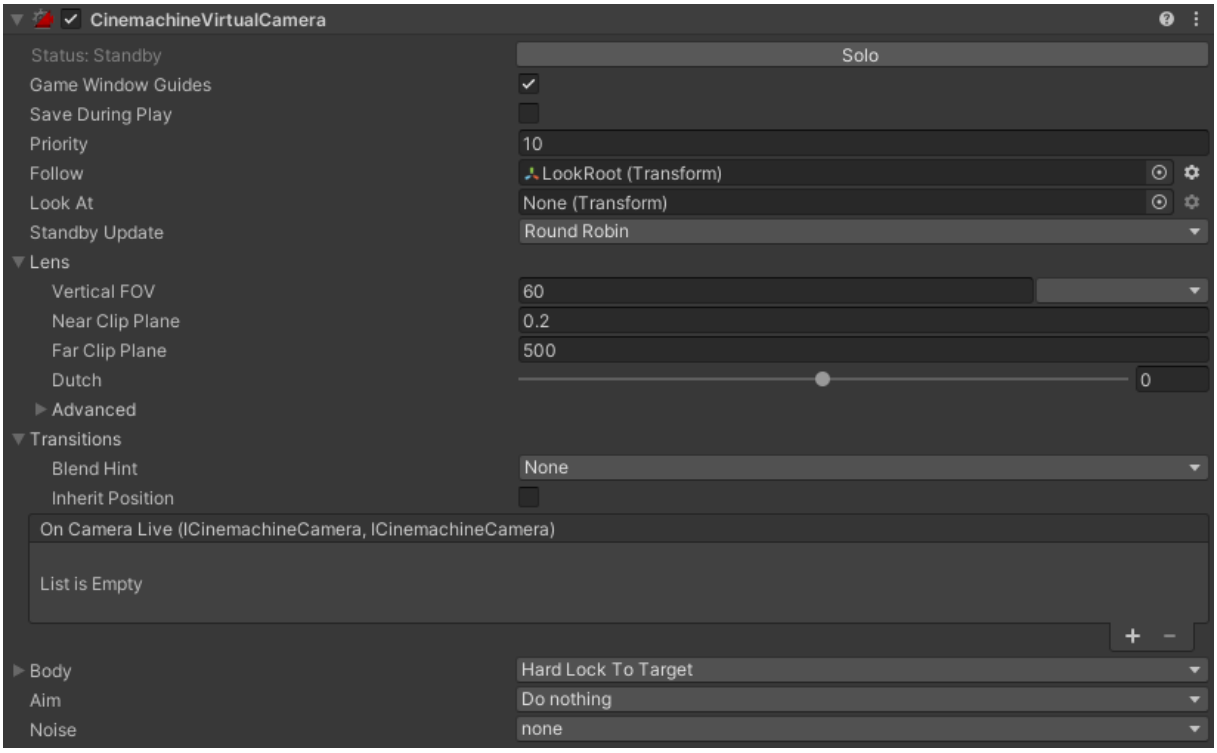
Slojevi na vremenskoj traci mogu se preklapati i miješati kako bi se postigli složeni efekti. To omogućuje sinkronizaciju animacija sa zvukovima ili kontrolu kamere tijekom ključnih trenutaka u igri. U drugoj kratkoj sceni postavljene su dvije virtualne kamere.

Virtualne kamere su dostupne u paketu "Cinemachine" [6][7]. To je napredni paket koji je razvio Unity Technologies kako bi omogućio jednostavniju i učinkovitiju kontrolu nad kamerama u igrama ili simulacijama. Cinemachine se bazira na konceptu virtualnih kamera, ali

dodaje brojne dodatne značajke i funkcionalnosti koje čine upravljanje kamerama još fleksibilnijim i snažnijim. Ovaj paket posebno cilja na stvaranje kinematografskih iskustava i glatkih tranzicija između različitih kamera.

Virtualne kamere su komponente koje se koriste za kontrolu pogleda i prikaza u igri. One simuliraju ponašanje stvarnih kamera, omogućujući dinamičko upravljanje pogledom igre kako bi se postigao željeni vizualni efekt.

Na slici je prikazana virtualna kamera jedan koja se nalazi na istom mjestu i u istoj točci kao obična Unity kamera. Neka od svojstava virtualne kamere uključuju praćenje objekta, rotiranje prema objektu, veličinu vidnog polja, daljinu prikaza, brzinu glatke tranzicije i ostalo. Na slici 17 su prikazana svojstva virtualne kamere.



Slika 17. Prikaz svojstva virtualne kamere

Prva virtualna kamera prati igrača u bilo kojem zadanom trenutku, dok je druga kamera stacionarna i nalazi se u jednoj od kuća u svijetu iza vrata. Nakon dolaska igrača do tog mjesta aktivira se okidač koji pokreće spomenutu filmsku scenu.

Pokretanjem kratke filmske scene uključuje se tranzicija od prve virtualne kamere, odnosno igrača, do druge virtualne kamere kako bi se omogućilo sigurno postojanje glatke tranzicije svaki put kada igrač dođe s drugog kuta ili mjesta do okidača. U kratkoj filmskoj sceni su

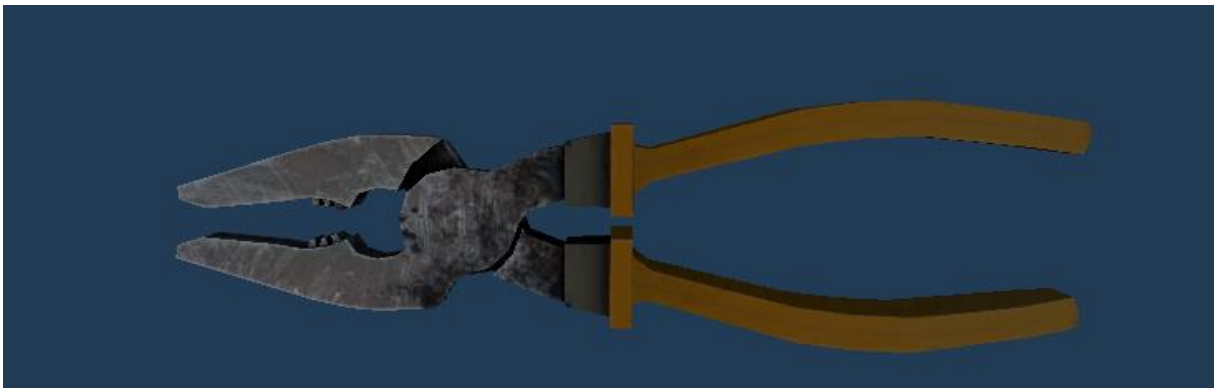
isključene sve kontrole na igraču kako ne bi došlo do neželjenog kretanja ili aktiviranja ostalih aktivnosti. Nakon toga se pušta već pripremljeni vremenski video zapis koji ima svoj tok aktivnosti spomenutih ranije, te se nakon filmske scene aktivira ključ koji izvodi tranziciju iz druge virtualne kamere natrag u prvu i vraća kontrola igraču.

Važno je spomenuti da se nakon filmske scene, na primjer, otključava garaža u kojoj se nalazi specijalan predmet, te bez njega ne možemo dovršiti igru. Stoga su filmske scene neophodne za završetak igre. Slika 18 prikazuje izradu kratke filmske scene s virtualnim kamerama.

Slika 18. Prikaz izrade kratke filmske scene sa virtualnim kamerama.

5.5.3. Specijalni predmeti

Specijalni predmeti imaju veliku ulogu kod napredovanja u igri. Potrebno ih je prikupiti sve za otključavanje finalne zone u igri kako bi igraču omogućili završetak igre. Samim time potičemo igrača na istraživanje veliku većinu svijeta, kako bi se upoznao s okolinom a ne pokušao brzinski završiti igru. Na slici 19 se vidi jedan od specijalnih predmeta odnosno kliješta koje će biti korisna za napredovanje kasnije u igri.



Slika 19. Specijalni predmet – kliješta

5.6. Spremanje napretka igre

Unity ima nekoliko metoda i tehnika za spremanje podataka, a izbor metode ovisi o složenosti igre i podacima koje je potrebno spremiti. Neke od metoda su PlayerPrefs, Serijalizacija (JSON, XML, binarna), SQL baza podataka, spremanje u oblaku i skriptirani objekti. U ovom projektu korištena je serijalizacija podataka tipa JSON. Unity nema ugrađenu metodu spremanje igre putem JSON formata, stoga je stvoren vlastiti sustav koji će biti opisan u nekoliko sljedećih koraka.

A) Struktura podataka: U posebnu skriptu definira se tip podataka koji će se spremati, tipa liste cijelih brojeva, bool ili slično. U ovom projektu korišten je Vector3 tip za spremanje koordinata igrača na mapi, cijeli broj za spremanje bodova odnosno broj ubijenih neprijatelja, trenutni dan u igri i na kraju metoda koja dozvoljava pristup tim podacima. Programski kod 51 prikazuje klasu GameData.

```
[System.Serializable]
24 references
public class GameData
{
    public long lastUpdated;
    public int scoreCounter;
    public int dayCounter;
    public Vector3 playerPosition;
    1 reference
    public GameData()
    {
        this.scoreCounter = 0;
        this.dayCounter = 0;
        playerPosition = Vector3.zero;
```

Programski kod 51. Prikaz klase GameData

B) Upravljanje spremanjem igre: Potrebno je definirati nekoliko metoda koje će spremati i učitavati podatke, odrediti putanju u kojoj će se podaci spremati i nekoliko pomoćnih metoda. Na početku se definira lista svih objekata koji koriste metodu za spremanje igre kako bi se svi potrebni podati ispravno spremili. Programski kod 52 prikazuje spomenutu listu.

```
private List<IDataPersistence> FindAllDataPersistenceObjects()
{
    IEnumerable<IDataPersistence> dataPersistenceObjects =
        FindObjectsOfType<MonoBehaviour>().OfType<IDataPersistence>();

    return new List<IDataPersistence>(dataPersistenceObjects);
```

Programski kod 52. Definiranje liste spremanja podataka

Nakon toga je implementirana funkcija LoadGame() koja provjerava ako postoje spremljeni podaci u datoteci te ako postoje učitavaju se spremljeni podaci iz svih objekata koji koriste metodu za spremanje podataka, a u suprotnom se kreira nova instanca igre. Programski kod 53 prikazuje funkciju LoadGame().

```
public void LoadGame()
{
    if (disableDataPersistence)...

    //load any saved data from file
    this.gameData = dataHandler.Load(selectedProfileId);

    if(this.gameData == null && initializeDataIfNull)
    {
        NewGame();
    }

    //if no data can be loaded, return
    if(this.gameData == null)
    {
        Debug.Log("No data was found.");
        return;
    }

    //push the loaded data to all other scripts that need it
    foreach (IDataPersistence dataPersistenceObj in dataPersistenceObjects)
    {
        dataPersistenceObj.LoadData(gameData);
    }
```

Programski kod 53. Prikaz metode LoadGame()

Sljedeće definirano je funkcija SaveGame(). Kao što samo ime funkcije govori, ona sprema trenutne podatke igre i vrijeme zadnjeg ažuriranja spremanja igre tako da provjerava sve aktivne objekte koji sadrže metodu spremanja igre. Sve podatke šalje drugoj skripti koja će biti opisana kasnije. Programski kod 54 prikazuje funkciju SaveGame().

```
public void SaveGame()
{
    if (disableDataPersistence)
    {
        return;
    }

    if (this.gameData == null)
    {
        Debug.LogWarning("No data was found. New game needs to be started.");
        return;
    }

    foreach (IDataPersistence dataPersistenceObj in dataPersistenceObjects)
    {
        dataPersistenceObj.SaveData(ref gameData);
    }
    gameData.lastUpdated = System.DateTime.Now.ToBinary();

    //save data to a file
    dataHandler.Save(gameData, selectedProfileId);
```

Programski kod 54. Prikaz metode SaveGame()

Nakon toga su definirane neke funkcije koje određuju ponašanje upravitelja spremanja igre. Upravitelj spremanja igre nalazi se kroz više scena u igri kako bi omogućio da je ispravna instanca igre pokrenuta.

Upravitelja je potrebno inicijalizirati u više scena, kako bi bio omogućen kroz kratke filmske scene ukoliko korisnik slučajno ugasi igru, u glavnom izborniku kako bi učitavanje igre bilo ispravno s ispravnim podacima te u ulaznoj sceni igre. Funkcija OnEnable() se uključuje prva odmah nakon što je objekt aktiviran u sceni, te on definira brojač kojem je zadatak za učitavanje ispravne scene. Funkcija OnDisable() radi apsolutno suprotno i pomiče brojač scena u negativnom smjeru. Nakon toga je definirana metoda OnSceneLoaded() koja se aktivira učitavanjem scene i pronalazi sve objekte s upraviteljem spremanja i nakon toga učitava igru. Također definirana je metoda OnSceneUnloaded() koja radi suprotno i nakon izlaska iz trenutne scene sprema igru i naš napredak. Programski kod 55 prikazuje nekoliko funkcija u skripti upravitelja igre.

```
private void OnEnable()
{
    SceneManager.sceneLoaded += OnSceneLoaded;
    SceneManager.sceneUnloaded += OnSceneUnload;
}

Unity Message | 0 references
private void OnDisable()
{
    SceneManager.sceneLoaded -= OnSceneLoaded;
    SceneManager.sceneUnloaded -= OnSceneUnload;
}

2 references
public void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    this.dataPersistenceObjects = FindAllDataPersistenceObjects();
    LoadGame();
}

2 references
public void OnSceneUnload(Scene scene)
{
    SaveGame();
}

1 reference
public void ChangeSelectedProfileId(string newProfileId)
{
    this.selectedProfileId = newProfileId;

    LoadGame();
}

2 references
public void NewGame()
{
    this.gameData = new GameData();
```

Programski kod 55. Prikaz nekoliko funkcija u skripti upravitelja spremanja igre

C) Klasa potrebna za komunikaciju računala i igre kako bi sve gore navedene metode i funkcije radile ispravno. U nastavku će biti opisana klasa FileDataHandler koja je odgovorna za spremanje igre na disk, učitavanje iste s diska, kreiranje direktorija u kojem se igra sprema i pomoćna metoda koja sprema zadnje spremljenu igru. Prije svega, uključuje se System.IO imenski prostor kako bi omogućio rad s datotečnim operacijama kao što su čitanje i pisanje

datoteka. System.IO je dio .NET okvira koji pruža niz klasa i metoda koje omogućuju manipulaciju datotekama i direktorijima.

Prvo će biti objašnjena funkcija Save(), koja je odgovorna za spremanje igre u direktorij na računalo korisnika. Funkcija prihvaća argument tipa GameData koji je bio definiran na početku poglavlja upravitelja spremanja igre i argument imena profila. Ime profila govori računalu kako će se datoteka imenovati. U metodi Path.Combine se spaja trenutni direktorij u kojem se nalaze podaci od igre, ime profila i ime datoteke. Nakon toga se kreira direktorij u koji će se datoteka spremiti. Također je implementirana klasa JsonUtility koja omogućuje konverziju objekata između JavaScript objektnih notacija i C# objekata i čini serijalizaciju i deserijalizaciju podataka jednostavnom. Vrlo je korisna kada je potrebno spremiti podatke u datoteku ili razmjenjivati podatke između Unity okruženja i vanjskih sustava. Programski kod 56 prikazuje funkciju Save().

```
public void Save(GameData data, string profileId)
{
    if (profileId == null)
    {
        return;
    }

    string fullPath = Path.Combine(dataDirPath, profileId, dataFileName);
    try
    {
        Directory.CreateDirectory(Path.GetDirectoryName(fullPath));

        string dataToStore = JsonUtility.ToJson(data, true);

        using (FileStream stream = new FileStream(fullPath, FileMode.Create))
        {
            using (StreamWriter writer = new StreamWriter(stream))
            {
                writer.Write(dataToStore);
            }
        }
    }
    catch (Exception e)
    {
        Debug.LogError("Error occured when trying to save data to a file: " + fullPath + "\n" + e);
    }
```

Programski kod 56. Prikaz funkcije Save()

Sljedeća funkcija Load() ima zadatak učitavanje igre s diska. Ulazni argument u funkciju je ime profila gdje se provjerava ako ulazno ime postoji u datoteci na disku te ako postoji učitava se igra iz datoteke i sprema se u pomoćnu varijablu loadedData. Programski kod 57 prikazuje funkciju Load().

```
public GameData Load(string profileId)
{
    if(profileId == null)
    {
        return null;
    }

    string fullPath = Path.Combine(dataDirPath, profileId, dataFileName);
    GameData loadedData = null;
    if (File.Exists(fullPath))
    {
        try
        {
            string dataToLoad = "";
            using (FileStream stream = new FileStream(fullPath, FileMode.Open))
            {
                using (StreamReader reader = new StreamReader(stream))
                {
                    dataToLoad = reader.ReadToEnd();
                }
            }

            loadedData = JsonUtility.FromJson<GameData>(dataToLoad);
        }
        catch (Exception e)
        {
            Debug.LogError("Error occured when trying to load data from file: " + fullPath + "\n" + e);
        }

    }
    return loadedData;
```

Programski kod 57. Prikaz funkcije Load()

Funkcija LoadAllProfiles vraća povratni tip dictionary koji sadrži ime profila i učitava sve podatke tipa GameData te tako omogućuje čitanje podataka iz datoteke. Dictionary je kolekcija koja omogućuje pohranjivanje parova ključ – vrijednost. Svaki ključ u Dictionary-u mora biti jedinstven i koristi se za pristup povezanoj vrijednosti. U prijevodu to bi značilo da se vrijednosti mogu brzo i efikasno dohvatiti koristeći ključ. Programski kod 58 prikazuje funkciju LoadAllProfiles().

```
public Dictionary<string, GameData> LoadAllProfiles()
{
    Dictionary<string, GameData> profileDictionary = new Dictionary<string, GameData>();
    IEnumerable<DirectoryInfo> dirInfos = new DirectoryInfo(dataDirPath).EnumerateDirectories();
    foreach (DirectoryInfo dirInfo in dirInfos)
    {
        string profileId = dirInfo.Name;

        string fullPath = Path.Combine(dataDirPath, profileId, dataFileName);
        if (!File.Exists(fullPath))...

        GameData profileData = Load(profileId);

        if(profileData != null)
        {
            profileDictionary.Add(profileId, profileData);
        }
        else...
    }

    return profileDictionary;
```

Programski kod 58. Prikaz funkcije LoadAllProfiles()

Nakon svih funkcija bit će prikazana i objašnjena implementacija svih navedenih elemenata kroz grafičko sučelje. U sučelju je izrađen novi izbornik koji nudi mogućnost igraču spremanja četiri različite instance igre, te izbornik prikazuje dva osnovna detalja a to su količina sakupljenih bodova u određenoj instanci te prikaz trenutnog dana u igri, kako bi igrač znao o kojoj spremljenoj instanci igre se radi prije njenog samog učitavanja. Neiskorišteni utori, odnosno utori bez spremljenog napretka su prikazani kao prazni. Na slici 20 je prikazan izbornik sa spremljenim napretkom.

Slika 20. Prikaz izbornika sa spremljenim napretkom

5.7. Naknadna obrada

Naknadna obrada (eng. post-processing) se odnosi na skup vizualnih efekata i tehnika manipulacije slike koje se primjenjuju na konačni izlaz kamere u sceni. Ovi efekti se obično koriste kako bi se poboljšala ukupna vizualna kvaliteta i atmosfera igre [10]. Unity pruža stog za naknadnu obradu (eng. post-processing stack) koji olakšava primjenu ovih efekata. Koristeći naknadnu obradu može se drastično poboljšati izgled igre i igračima pružiti bolje iskustvo. Efekti poput zamućenje dubine (eng. depth of field), zamućenje pokreta (eng. motion blur), bojenje (eng. color grading), sjaj (eng. Bloom) omogućuju postignuće različitih stilova i

atmosfera koje se žele postići u igri. Na slici 21 bit će prikazan izgled igre bez naknadne obrade, a na slici 22 prikazan s naknadnom obradom.

Slika 21. Snimak igre bez naknadne obrade

Slika 22. Snimak igre sa naknadnom obradom

Bitno je također napomenuti kako je prva inačica ove igre bila rađena u Unity "built-in" prikazu cjevovoda, koji je detaljno objašnjen na početku ovog rada, a kasnije verzije igre su

nadograđene odnosno pretvorene u univerzalni skriptni cjevovod. Takav tip cjevovoda pruža puno veću optimizaciju i bolje performanse tijekom igre. Na slici 23 će se prikazati izgled igre s "built-in" cjevovodom, nakon čega će slika 24 prikazati izgled igre s takozvanim URP cjevovodom.

Slika 23. Snimak igre sa 'built-in' cjevovodom

Slika 24. Snimak igre sa URP cjevovodom

5.8. Završetak igre

U završnoj sceni igre, igrač se penje na treći kat zgrade gdje ulazi u sobu koja postaje poprište posljednjeg sukoba. Soba, obavijena mračnom atmosferom, postaje arena u kojoj demoni iskušavaju igračevu psihičku izdržljivost. Oni se igraju s percepcijom stvarnosti, izazivajući seriju iznenadnih strahova (eng. jumpscare) trenutaka koji igrača dovode do ruba straha.

Kako se igrač bori s osobnim demonima, u sobi otkriva tijelo svoje preminule majke koja leži na krevetu, što je prizor koji paralizira i izaziva duboku emocionalnu reakciju. Trauma se nastavlja kada igrač pronalazi i tijelo svog oca, također mrtvog, što dodatno pojačava osjećaj gubitka i očaja.

Napetost doseže vrhunac kada igrač ulazi u kupaonicu s ogledalom. U tom trenutku, igra se poigrava s elementima horora kada se svjetlo u kupaonici gasi i pali, stvarajući klaustro fobičnu i napetu atmosferu. U trenucima kada je svjetlo ugašeno, igrač je prepušten mašti i zvukovima koji sugeriraju da nešto ili netko može biti u prostoriji s njim.

Konačni obrat događa se kada se u ogledalu otkrije da je igrač zapravo glavni um iza svih zavrzlama koje su se odvijale tijekom igre. Otkriva se da je igrač kontrolirao demone i zombije, što igraču postavlja pitanje vlastite uloge i identiteta unutar igre. Ovaj neočekivani preokret dovodi do dubokog preispitivanja i razmišljanja o prirodi zla i slobodne volje.

Završetak igre ostavlja igrača u stanju šoka i nevjerice, prisiljavajući ga da razmisli o svakom koraku koji je prethodio ovom trenutku i kako su njegove akcije možda bile pod utjecajem nečeg mnogo mračnijeg od onoga što je prvotno pretpostavljao.

6. Zaključak

Kroz proces istraživanja i razvoja prototipa FPS igre u Unity okruženju, ovaj diplomski rad je pružio detaljan uvid u kompleksnost stvaranja interaktivnog igračkog iskustva. Dok je Unity pokazao impresivnu fleksibilnost i snagu u pogledu razvoja igara, postoje ograničenja koja su se pojavila tijekom realizacije projekta.

Jedno od ključnih ograničenja ovog projekta bilo je vrijeme. S obzirom na vremenska ograničenja akademskog okvira, neke od naprednijih funkcionalnosti i poliranja igre nisu mogle biti u potpunosti razvijene ili implementirane. To uključuje dublje mehanike umjetne inteligencije, složenije mrežne mogućnosti za više igrača i detaljnije razrađen svijet.

Drugo ograničenje odnosi se na resurse. Iako Unity pruža širok spektar alata i sredstava, stvaranje visokokvalitetnih modela, tekstura i animacija zahtijeva vrijeme i vještine koje premašuju okvire ovog projekta. Kao rezultat, neki vizualni i dizajnerski elementi nisu dostigli razinu profesionalne produkcije koja se očekuje u komercijalnim igrama.

Unatoč ovim ograničenjima, postoji niz mogućih poboljšanja i proširenja za budući razvoj projekta. Na primjer, integracija naprednijih AI sustava mogla bi pružiti igračima izazovnije i dinamičnije iskustvo, dok bi razvoj mrežnih kapaciteta omogućio podršku za igranje s više igrača, što bi značajno proširilo doseg i privlačnost igre. Također, dodatno ulaganje u razvoj priče i svijeta igre moglo bi obogatiti narativ i stvoriti dublju povezanost igrača s igrom.

U konačnici, ovaj diplomski rad je postavio temelje za razvoj FPS igre u Unity-u, ističući kako snage tako i slabosti pristupa. Budući rad na projektu trebao bi se usredotočiti na prevladavanje identificiranih ograničenja i iskorištavanje punog potencijala Unity-a kao alata za razvoj igara. S obzirom na brzi razvoj tehnologije i rastuće mogućnosti u industriji igara, potencijal za daljnje poboljšanje i inovacije ostaje velik i otvoren za istraživanje.

A Week To Survive Github: https://github.com/NikolaPatafta/A-Week-To-Survive

Literatura:

[1] Zenva (2023.), https://gamedevacademy.org/what-is-unity/, datum pristupa 23.2.2023.

[2] 7DaysToDie (4.5.2016.), https://7daystodie.com/, datum pristupa 2.2.2023.

[3] Endnight Games (2017.), https://endnightgames.com/games/the-forest, datum pristupa 2.2.2023.

[4] Unity Technologies (2023.), Unity documentation, https://docs.unity.com/ datum prvog pristupa 12.2.2023.

[5] Unity Technologies (2023.), Physics Raycast, https://docs.unity3d.com/ScriptReference/Physics.Raycast.html, datum pristupa 14.4.2023.

[6] Unity Technologies (2023.), Cinemachine Documentation, https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/index.html, datum pristupa 24.6.2023.

[7] Daniel Buckley (22.10.2023.), Understanding Unity Cinemachine, https://gamedevacademy.org/unity-cinemachine-tutorial/, datum pristupa 27.10.2023.

[8] Unity Technologies (16.5.2023.), Introduction to Timeline, https://learn.unity.com/tutorial/introduction-to-timeline-2019-3, datum pristupa 13.6.2023.

[9] Temitope Oyedele (5.9.2022.), Exploring post-processing in Unity, https://blog.logrocket.com/exploring-post-processing-unity/, datum pristupa 16.7.2023.

[10] Lukas Bobor (21.12.2016.), „Village Buildings“, https://assetstore.unity.com/packages/3d/environments/urban/village-buildings-77486

[11] Dr.Bean (20.1.2023.), „Free Open Building“, https://assetstore.unity.com/packages/3d/environments/free-open-building-112907

[12] MadMedicSoft, (5.4.2018.), Russian Buildings Pack“, https://assetstore.unity.com/packages/3d/environments/urban/russian-buildings-pack-113375

[13] Dmitriy Dryzhak, (10.8.2021.), „Ghoul-zombie“, https://assetstore.unity.com/packages/3d/characters/ghoul-zombie-114531

[14] Jan Fidler, (31.5.2018.), „Classic Interior Door Pack 1“, https://assetstore.unity.com/packages/3d/props/interior/classic-interior-door-pack-1-118744

[15] Kobra Game Studios, (18.10.2016.), „Chainlink Fences“, https://assetstore.unity.com/packages/3d/chainlink-fences-73107

[16] PolygonPi, (13.9.2021.), „Fence Layout Tool“, https://assetstore.unity.com/packages/tools/utilities/fence-layout-tool-162856

[17] Mila Shalabai, (2.1.2019.), „Modular Abandoned Slaughterhouse: Lite“, https://assetstore.unity.com/packages/3d/environments/urban/modular-abandoned-slaughterhouse-lite-58082

[18] Duane's Mind, (7.9.2017.), „Military Vehicle“, https://assetstore.unity.com/packages/3d/vehicles/land/military-vehicle-9225

[19] Dmitriy Dryzhak, (6.8.2019.), „Low Poly Vehicles Pack“, https://assetstore.unity.com/packages/3d/vehicles/land/low-poly-vehicles-pack-26707

[20] Essplashkid, (15.2.2018.), „Some Pliers“, https://assetstore.unity.com/packages/3d/props/tools/some-pliers-110393

[21] Monqo Studios, (18.1.2021.), „Free Pipe Wrench“, https://assetstore.unity.com/packages/3d/props/tools/free-pipe-wrench-187070

[22] RRFreelance, (6.12.2017.), „Crowbar“, https://assetstore.unity.com/packages/3d/props/tools/crowbar-20500

[23] AiKodex, (14.7.2021.), „Simple Garage“, https://assetstore.unity.com/packages/3d/props/interior/simple-garage-197251

[24] Realtime Models, (5.3.2023.), „Aircraft Hangars“, https://sketchfab.com/3d-models/aircraft-hangar-94ea9599bfe941598b52cdc4b7ac8147

[25] Noah, (11.7.2021.), „Plane“, https://sketchfab.com/3d-models/plane-4c3598cd0ba547aeb10a4833d6a36566

[26] Ririnhasra88, (15.9.2021.), „Aerodrome“, https://sketchfab.com/3d-models/aerodrome-ec5a7b76380448f38a39e45f2e848d16

[27] Omni Studio, (24.9.2018.), „High Quality War Plane“, https://assetstore.unity.com/packages/3d/vehicles/air/high-quality-war-plane-128092

[28] Unrealistic Arts, (22.10.2020.), „Free House 01 – PBR“, https://assetstore.unity.com/packages/3d/environments/urban/free-house-01-pbr-181353

[29] Nikolay Fedorov, (3.2.2016.), „Old Soviet Shop“, https://assetstore.unity.com/packages/3d/environments/urban/old-soviet-shop-54767

[30] Rossendy & Brito, (1.2.2017.), „Concrete Barricades“, https://assetstore.unity.com/packages/3d/concrete-barricades-80401

[31] Ribrado, (14.3.2019.), „Free Street Props“, https://assetstore.unity.com/packages/3d/props/exterior/free-street-props-141604

[32] Aleksn09, (20.8.2020.), „Old Road Signs PBR“, https://assetstore.unity.com/packages/3d/props/exterior/old-road-signs-pbr-170952

[33] WhiteXopc, (31.8.2022.), „Pillars Pack, https://assetstore.unity.com/packages/3d/environments/industrial/pillars-pack-228563

[34] Universal Assets, (19.9.2018.), „Industrial Props Kit“, https://assetstore.unity.com/packages/3d/props/industrial/industrial-props-kit-84745

[35] Zombie Studios, (3.9.2015.), „Warpig“, https://p3dm.ru/files/characters/supernatural/4159-warpig-.html

[36] Johnny Kasapi, (2.25.2022.), „Furnished Cabin“, https://assetstore.unity.com/packages/3d/environments/free-open-building-112907

Popis slika

Popis programskog koda

Sažetak

Ovaj diplomski rad bavi se razvojem pucačine iz prvog lica (FPS) igre koristeći Unity, jedan od vodećih alata za razvoj igara. Rad detaljno istražuje sve faze razvoja igre, od početne konceptualizacije, preko dizajna i implementacije ključnih mehanika, do testiranja i finalizacije. Cilj rada je pružiti duboko razumijevanje procesa razvoja igara unutar Unity okruženja, s posebnim naglaskom na izazove i rješenja specifična za FPS žanr. U radu se razmatraju osnovni elementi koji čine FPS igru, uključujući dizajn svijeta, mehanike pucanja, umjetnu inteligenciju neprijatelja, korisničko sučelje i optimizaciju performansi. Također, rad se osvrće na važnost narativa i atmosfere igračkog iskustva. Kroz proces razvoja, dokumentirane su različite faze dizajna i programiranja, kao i integracija različitih Unity alata. Unatoč uspješnom razvoju igre, rad identificira ključna ograničenja projekta, uključujući vremenska i resursna ograničenja koja su utjecala na dubinu i poliranje konačnog proizvoda. Kao rezultat, rad predlaže moguća poboljšanja i proširenja, poput razvoja naprednijih AI sustava, mrežnih mogućnosti i poboljšanja vizualnih elemenata, koja bi mogla biti istražena u budućim iteracijama projekta.

Ključne riječi: Unity okruženje, animacije, objekti, upravljači, programiranje

Abstract

This thesis deals with the development of a first-person shooter (FPS) game using Unity, one of the leading game development tools. The work thoroughly investigates all stages of game development, from initial conceptualization, through design and implementation of key mechanics, to testing and finalization. The aim of the thesis is to provide a deep understanding of the game development process within the Unity environment, with a special emphasis on the challenges and solutions specific to the FPS genre. The paper considers the basic elements that make up an FPS game, including world design, shooting mechanics, enemy artificial intelligence, user interface, and performance optimization. Additionally, the work reflects on the importance of narrative and atmosphere in the gaming experience. Throughout the development process, various stages of design and programming are documented, as well as the integration of different Unity tools. Despite the successful development of the game, the work identifies key limitations of the project, including time and resource constraints that have affected the depth and polish of the final product. As a result, the paper suggests possible improvements and expansions, such as the development of more advanced AI systems, networking capabilities, and enhancements to visual elements, which could be explored in future iterations of the project.

Key words: Unity enviroment, animations, objects, controllers, programming