Silent Nut: razvoj roguelike igre u Unity okruženju

Nejedly, Vedran

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

VEDRAN NEJEDLY

SILENT NUT: RAZVOJ ROGUELIKE IGRE U UNITY OKRUŽENJU

Diplomski rad

Pula, srpanj, 2023.

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike

VEDRAN NEJEDLY

SILENT NUT: RAZVOJ ROGUELIKE IGRE U UNITY OKRUŽENJU

Diplomski rad

JMBAG: 0303075754, redoviti student

Studijski smjer: Informatika

Kolegij: 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, srpanj, 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Vedran Nejedly, 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

Vedran Nejedly

U Puli, 04.07.2023. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Vedran Nejedly dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Silent Nut: razvoj roguelike igre 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, 04.07.2023. godine

Potpis

Vedran Nejedly

Sadržaj

1. Uvod

U ovom diplomskom radu će biti opisan proces izrade roguelike igrice u Unity razvojnom okruženju pod nazivom Silent Nut. Programski jezik koji se koristi u Unity razvojnom okruženju je C# te je isti korišten za pisanje koda. Inačica Unity programa koja je korištena pri izradi ovog diplomskog rada je 2021.3.10f1.

Ključni dijelovi roguelike igrice su trajna smrt igrača, odnosno kada igrač umre igra prestaje te proceduralno stvaranje igraćeg terena. Nakon što igrač umre, može ponovno pokrenuti igru ili izaći iz nje. Pri pokretanju igre igrač mora odabrati jednog od tri pratilaca koji će ići s njim kroz igru. Svaki od pratilaca pomaže igraču na dva načina, aktivni i pasivni. Aktivni način je da se pratilac kreće prema neprijateljima te im svojim magičnim krugom nanosi štetu, a pasivni način ovisi o tome koji je pratilac odabran. Svaki pratilac ima jedinstven učinak na igru poput povećanja brzine kretanja igrača. Za nove igrače je preporučljivo koristiti pratioca koji liječi igrača. Priča (eng. story) igre je da se igrač nalazi u svom umu koji u svakoj novoj prostoriji stvara nove neprijatelje s kojima se igrač mora boriti i pobijediti ih kako bi mogao napredovati dalje kroz igru. Igrač može neprijatelje pobijediti na nekoliko načina.

Prvi način je da ih udara sjekirom, drugi način je korištenjem magije i treći način je da igrač namami neprijatelje u zamke koje ih mogu ozlijediti, ali mora pripaziti kod korištenja ove metode zbog toga što zamke mogu ozlijediti i igrača. Također igrač se može ojačati tako da pokupi razne predmete (eng. item) koji igraču pomažu tako da mu povećaju štetu koju nanosi neprijateljima, životne bodove, brzinu kretanja i slično. U nastavku rada će biti navedene inspiracije za izradu igrice, povijest razvojnog okruženja te prikazani i objašnjeni svi elementi koji su potrebni za izradu roguelike igrice te način na koji rade.

2. Inspiracije

Igrica Silent Nut povlači inspiraciju iz nekoliko igrica od koje je glavna inspiracija The Binding of Isaac od developera Edmunda McMillena i Floriana Himsla. Uz The Binding of Isaac se nalazi i Risk Of Rain 2 od developera Hopoo Games te Nobody Saves The World od DrinkBox Studios-a. U nastavku će spomenute igrice biti detaljnije opisane.

2.1 The Binding of Isaac

The Binding Of Isaac je video igra koju su razvili Edmund McMillen i Florian Himsl 2011. Godine te je ona najveća inspiracija pri izradi igrice „Silent Nut”. Igrica je kombinacija elemenata iz puzača kroz tamnicu (eng. dungeon crawler) igrica, roguelike igrica i upravljanje s dva kontrolera (eng. twin stick) pucačina. Jednim kontrolerom se upravlja kretanjem igrača, a drugim kontrolerom se upravlja smjerom pucanja. Tema igre je mračna i uznemirujuća te istražuje pitanja religije, zlostavljanja te osobne borbe. Prikaz snimke ekrana iz igrice The Binding of Isaac je vidljiv na slici 1.

Slika 1. Snimka ekrana iz igrice The Binding of Isaac

Priča igrice jest da je Isaac zatočen u podrumu od strane svoje duboko religiozne majke koja ga želi žrtvovati ne bi li dokazala svoju odanost. Isaac bježi kroz skrivena vrata koja vode u tamnicu. Igra se sastoji od nasumično generirane tamnice u kojoj se nalaze neprijatelji, prepreke i blago. Cilj igre je pobijediti šefove kroz svaku razinu te tako napredovati igrom, na zadnjem nivou Isaac bori se protiv svoje majke. The Binding of Isaac je igrica koja igraču daje mnoštvo zabave nakon što je igrač pobjedi, jer svaka je igra drugačija, od generacije mape do blaga koje igrač može pokupiti.

2.2 Risk of Rain 2

Risk of Rain 2 je kooperativna roguelike pucačina u trećem licu koju je razvio Hopoo Games. objavljena 2019. godine , preuzima intenzivno i brzo igranje originalnog „Risk of Rain" i pretvara ga u potpuno 3D iskustvo. U igrici se igrači nađu na stranom planetu i moraju se boriti s hordama neprijateljskih stvorenja dok traže način da pobjegnu. Igra se vrti oko istraživanja proceduralno stvorenih razina i borbe sa sve izazovnijim neprijateljima. Kako vrijeme odmiče, poteškoće se povećavaju, potičući igrače da prilagode svoje strategije i iskoriste različite predmete i pojačanja koja otkriju na putu. Ovi predmeti pružaju jedinstvene sposobnosti i poboljšavaju atribute igrača. Slika 2 prikazuje naslovnu sliku igrice Risk of Rain 2.

Slika 2. Naslovna slika igrice Risk of Rain 2

2.3 Nobody Saves the World

Jedina igrica koja nije roguelike, a da igrica „Silent Nut” uzima inspiraciju od je Nobody Saves The World. Nobody Saves the World je akcijska RPG video igrica koja se igra po tamnicama, a razvila ju je i objavila tvrtka DrinkBox Studios. Iz ove igrice se izvlače neke mehanike poput bombi koje naganjaju neprijatelje, iako su u Nobody Saves The World to životinje poput zečeva i tigrova te zombiji. Snimka ekrana igrice Nobody Saves the World je prikazana na slici 3.

Slika 3. Snimka ekrana iz igrice Nobody Saves the World

3. Razvojno okruženje Unity

Tvrtku Over the Edge Entertainment (OTEE) osnovali su David Helgason, Nicholas Francis i Joachim Ante 2004. godine u Kopenhagenu, glavnom gradu Danske. Njihova prva izdana igra je GooBall koja nije doživjela komercijalni uspjeh, ali osnivači OTEE-a su uvidjeli potencijal u razvoju video igara te su se usredotočili na stvaranje game enginea za druge razvojne programere te 2005. godine je napravljen Unity. Oni su željeli olakšati pristup razvoju 2D i 3D igara te su to i učinili. Naredne godine su osvojili drugo mjesto na Apple-ovom natjecanju za najbolju upotrebu Mac OS X grafike. Tvrtka je 2007. godine promijenila naziv u Unity Technologies i značajno porasla kada je izdan iPhone, zato što su oni proizveli jedan od prvih enginea koji su u potpunosti podržavali platformu. U razdoblju između 2008. i 2009. godine njihovi su kupci kupovali Macintosh računala izdana od strane tvrtke Apple Computer koja su bila puštena u prodaju u siječnju 1984. godine. Shvatili su da se moraju prilagoditi tržištu te je kasnije dodana podrška za Microsoft Windows (Hass, 2014).

Unity 2.0 je izdan 2007 godine s velikim brojem novih značajki kao što su:

- Dinamične sjene u stvarnom vremenu
- Usmjerena svijetla
- Video playback
- Terrain Rendering Engine
- DirectX 9 Renderer za Windows
- Ugrađena mrežna podrška za više igrača

Unity 3.0 je izdan 2010 godine i glavne značajke su:

- Integracija alata Beast Lightmap tvrtke Illuminate Labs
- Renderiranje s odgodom
- Ugrađeni uređivač stabala

- Izvorno renderiranje fontova
- Automatsko UV mapiranje
- Audio filtere

Unity 4.0 je izdan 2012. godine kojem su glavne značajke:

- Dodana je podrška za DirectX 11 i Adobe Flash
- Dodani su novi alati za animaciju po nazivom Mecanim
- Pristup pretpregledu za Linux (Haas, 2014).

Unity 5.0 je izdan 2015. godine i glavne značajke su:

- Poboljšanje rasvjete i zvuka
- Korištenjem WebGL-a programeri su mogli dodati svoje igrice kompatibilnim web preglednicima bez dodataka potrebnih za igrače
- Globalno osvjetljenje u stvarnom vremenu
- Pregled mapiranja svijeta
- Unity cloud
- Nvidia PhysX 3.3 pokretač za fiziku (eng. Physics Engine)
- Dodani su efekti osvjetljenja i čestica
- 4K video player

Nakon Unity 5.0 verzije Unity Technologies je prešla na način prikaza Unity inačicama prema godinama izdanja. Glavne značajke izdanja Unity 2018. su Scriptable Render Pipeline za developere sa zahtjevnim grafikama i alate za strojno učenje, kao što je Imitation Learning, pri čemu igre uče od stvarnih navika igrača, podršku za Magic Leap i predloške za nove programere. Unity 2020. je predstavio Mixed and Augmented Reality

Studio koji programerima pruža dodatnu funkcionalnost za generiranje aplikacija proširene stvarnosti (AR) na temelju pravila. Promjene u Unity 2022. bile su namijenjene poboljšanju produktivnosti smanjenjem vremena potrebnog za ulazak u način rada za reprodukciju i uvoz datoteka, te implementacijom vizualnih upita za pretraživanje i višestrukog odabira u upravitelju paketa. Inačica Unity programa koja je korištena pri izradi ovog diplomskog rada je 2021.3.10f1.

4. Elementi roguelike igrica

Roguelike igrice su podžanr RPG-a (eng. Role playing games) te je žanr nazvan po igrici Rogue izdane 1980. godine. Glavne karakteristike roguelike igrica su vječna smrt igrača (eng. permadeath), proceduralno stvaranje terena i kompleksnost igre. Vječna smrt igrača znači da kada igrač umre u igrici njegov napredak je izgubljen i mora početi iz početka. Proceduralno stvaranje terena znači da će se teren graditi prema algoritmu, a ne ručno. U slučaju igrice Silent Nut proceduralno stvaranje terena započinje u ulaznoj prostoriji te se dalje stvaraju prostorije iz nje, što će biti objašnjeno u nastavku ovog rada. Kompleksnost igre predstavlja velik broj radnji koje igrač može učiniti ili zahtjev da igrač donese veliki broj taktičkih odluka kako bi ostao živ i nastavio dalje s prolaženjem video igrice (Parker, 2017).

U roguelike igricama vrlo je važno upravljanje resursima i poznavanje resursa koji su igraču dostupni. Primjer toga je ako postoji trgovina u igrici, igrač želi sakupljati novce i pametno ih trošiti, što znači da igrač neće kupiti nešto od čega nema koristi. Također u određenim situacijama igrač može biti prisiljen kupiti lošije stvari kako bi se njegova igra nastavila. Glavno pravilo roguelike igrica je ako igrač poznaje sve elemente video igrice, te ih je moguće izbjeći ako igra dovoljno oprezno, igrač bi uvijek trebao preživjeti (Harris, 2020).

U igrici „Silent Nut“ igrač će dobiti mogućnost sakupljanja stvari koje će mu pomoći u njegovoj igri, ali neke stvari igrač možda neće željeti pokupiti ovisno o svojem stilu igranja ili ovisno o samom učinku stvari koje pokupi. Također igrač se može približiti kristalima kako bi ih osvijetlio i ovisno o tipu kristala igraču će se dodijeliti ili oduzeti odgovarajuća vrijednost (npr. brzina kretanja).

Nasumična generacija će stvoriti nasumične rezultate na fiksne elemente igrice, odnosno pri stvaranju prostorije će se morati stvoriti odgovarajuća prostorija, ali ta prostorija neće uvijek biti ista već će se odabrati kao jedna prostorija od svih odgovarajućih prostorija. Nasumična generacija brojeva će osigurati da ishod nije isti svaki puta. Samo korištenje nasumične generacije brojeva može varirati od toga da nije važno u igri do toga da igra ne uspije zbog nasumične generacije i to sve ovisi o načinu korištenja nasumičnih brojeva (Warnock, 1999).

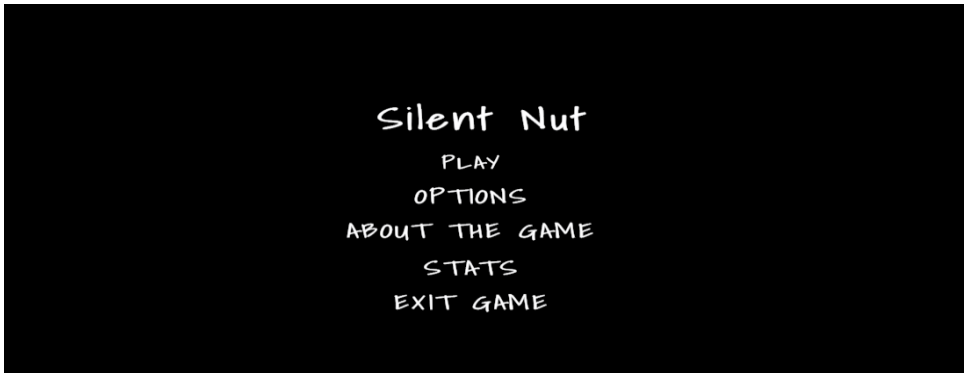
Generiranje nasumičnih brojeva služi isključivo za generiranje ishoda u rasponu brojeva. Pretpostavlja se da ako razvojni programer koristi generator nasumičnih brojeva da tako stvara pošteno iskustvo za igrača, ali to nije u potpunosti točno. Svaki igrač igrica koje sadrže nasumičnu generaciju brojeva zna da nije u potpunoj kontroli u toj situaciji već je nužno uzeti u obzir da na njegovu igricu djelomice utječe sreća, odnosno šansa da nasumična generacija da igraču povoljan broj. Dizajneri video igara imaju zadaću igračima prividno dati osjećaj da su u kontroli nad igrom iako to u potpunosti nije istina, odnosno potrebno je igraču dati osjećaj da igrač odlučuje može li pobijediti igricu. U teoriji je moguće napraviti generator nasumičnih brojeva poštenim, no to nije poanta roguelike igrica zbog toga što je u njima je bitno učiniti najviše s onim što igrač ima, odnosno kojim resursima raspolaže (Bycer, 2021).

5. Proces razvoja roguelike igrica

O ovom dijelu diplomskog rada će biti opisano kako je razvijena video igrica Silent Nut te svi potrebni elementi. Prvo će biti detaljno opisan glavni izbornik uz sve popratne skripte za funkcionalnost istog. Zatim će biti opisana izrada uvodne scene i svih njenih elemenata te će biti opisan svijet u kojem se igrač nalazi, njegovi neprijatelji, zamke, napredak kroz igru, stvari koje igrač može pokupiti te svijetleći kristali.

5.1 Glavni izbornik

U prikazu glavnog izbornika igrice igrač ima pet opcija, a to su: igraj (eng. play), postavke (eng. options), statistika (eng. stats), o igri (eng. about the game) i izlaz iz igre (eng. exit game). Pritiskom na gumb „igraj“ igraču se otvara novi UI prozor u kojem bira svog pratioca te pritiskom na pokreni igru (eng. start game) može pokrenuti igru. Pritiskom na gumb „postavke“ otvaraju se postavke igre gdje igrač može mijenjati rezoluciju, postavke zvuka, kvalitetu grafike i odabrati želi li igrati igru preko cijelog zaslona (eng. fullscreen) ili ne. Pritiskom na gumb „statistike“ prikazuju se podatci o igračevim prijašnjim igrama, odnosno broju igri, porazima i pobjedama te koliko igrač pobjeda ima za redom, a pritiskom na gumb „o igri“ prikazuju se podatci o studentu, mentoru, fakultetu i sveučilištu. Pritiskom na izlaz iz igre igrač prekida rad aplikacije. Na slici 4 je prikazan glavni izbornik igrice.



Slika 4. Snimka ekrana glavnog izbornika igrice Silent Nut

Isječak koda 1 prikazuje skriptu MainMenu.cs koja će biti opisana u nastavku.

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


public class MainMenu : MonoBehaviour
{
    public void StartGame(){
        //Load next scene in the build manager
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1 );
    }

    public void ExitGame(){
        //Close the application
        Application.Quit();
    }
}
```

Isječak koda 1. Skripta MainMenu.cs

MainMenu.cs skripta sadrži dvije funkcije. Prva funkcija je StartGame() tipa void koja učitava iduću zadanu scenu. Kako bi se mogla učitati sljedeća scena potrebno je u postavkama izgradnje (eng. build settings) postaviti scene redom kojim će se učitavati. Druga metoda je ExitGame() koja prekida izvršavanje aplikacije.

5.1.1 Odabir pratilaca

Nakon što igrač pritisne gumb igraj otvara se izbornik u kojem igrač treba odabrati pratioca koji će s njim ići kroz igru. Pri odabiru igrač će imati opcije pokrenuti igru, vratiti se na glavni izbornik, te pritiskom na gumbove prethodni (eng. previous) i idući (eng. next) moći će promijeniti trenutnog pratioca, što je prikazano na slici 5. Isječci koda 2, 3 i 4 će prikazati skripte potrebne za odabir pratioca.

Slika 5. Snimka ekrana odabira pratioca Burrow

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


public class CompanionSelector : MonoBehaviour
{

    public ValueHolder val;
    public RunStats runStats;

    int currentCompanion = 0;

    private void SelectCompanion(int index){
        if(index > transform.childCount-1) {
            currentCompanion = 0;
            index = 0;
            }
        if(index<0){
            index = transform.childCount-1;
            currentCompanion = transform.childCount-1;
        }
        for(int i = 0;i<transform.childCount;i++){
            this.transform.GetChild(i).gameObject.SetActive(i == index);
            if(i==index){
                val.storeValue(i);
            }
        }
    }
```

Isječak koda 2. CompanionSelector.cs 1.dio

Funkcija ChangeCompanion(int index) prima cjelobrojni parametar indeks i ako je taj parametar veći od broja djece objekta umanjen za jedan, indeks se postavlja na 0 i vrijednost varijable currentCompanion se postavlja na vrijednost 0, odnosno odabire se pratilac pod indeksom 0. Ako je vrijednost indeksa manja od 0, vrijednost se postavlja na vrijednost broja djece objekta umanjenu za jedan, odnosno prikazuje se zadnji pratilac.

Potom se prolazi kroz svu djecu objekta te ako je vrijednost varijable i jednaka indeksu, onda se taj objekt postavlja na aktivno stanje, odnosno prikazuje se na ekranu i njegova vrijednost se sprema u varijablu val skripte ValueHolder.cs.

```
    public void ChangeCompanion(int change){
        currentCcompanion += change;
        SelectCompanion(currentCompanion);
    }

      public void StartGame(){
        //Load next scene in the build manager
        runStats.StartARun();
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1 );
    }

}
```

Isječak koda 3. Skripta CompanionSelector.cs 2.dio

Funkcija tipa void ChangeCompanion(int change) prima cjelobrojnu vrijednost kao parametar. Cjelobrojna vrijednost može biti 1 ili –1 odnosno vrijednost 1 je vrijednost za pritisak na gumb sljedeći, a vrijednost –1 je vrijednost za pritisak na gumb prethodni. Metoda StartGame() poziva metodu StartARun() iz skripte runStats.cs te učitava sljedeću scenu.

Isječak koda 4 prikazuje skriptu ValueHolder.cs.

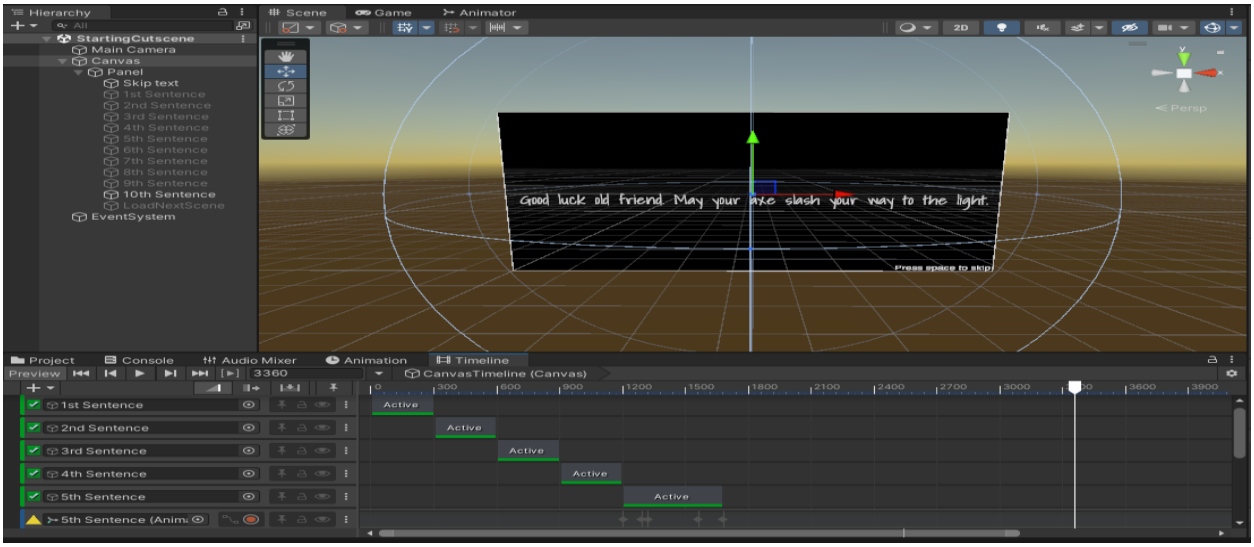
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ValueHolder : MonoBehaviour
{
    public int value;
    // Start is called before the first frame update
    public void storeValue(int val){
        value = val;
    }

}
```

Isječak koda 4. Skripta ValueHolder.cs

ValueHolder.cs skripta se sastoji od cjelobrojne varijable value i metode storeValue() koja prima cjelobrojni parametar te postavlja vrijednost varijable value na vrijednost primljenog parametra.

5.1.2 Uvodna scena u igru

Na slici 6 je prikazana vremenska crta uvodne scene koja će u nastavku biti objašnjena.



Slika 6. Vremenska crta uvodne scene

Uvodna scena u igru je napravljena koristeći vremensku crtu (eng. timeline) koja se nalazi na objektu igre platno (eng. canvas). Unutar platna je napravljen panel s jedanaest TextMeshPro objekata od kojih svaki predstavlja jednu rečenicu u vremenskoj crti, osim teksta koji se prikazuje cijelo vrijeme i nije dio vremenske crte, a to je tekst koji govori igraču da pritiskom tipke razmak on može preskočiti uvodnu scenu. Ostali TextMeshPro objekti su dodani u vremensku crtu i postavljeno im je vrijeme kada će biti postavljeni na aktivno stanje, odnosno kada će se prikazati na ekranu. Svaki od tekstova ima „fade in“ i „fade out“ animaciju. Kada se sve rečenice pokažu aktivirat će se objekt igre LoadNextScene koji je zadužen za učitavanje iduće scene. Skripta za učitavanje i preskakanje početne scene je prikazana u isječku koda 5 (Unity, 2023).

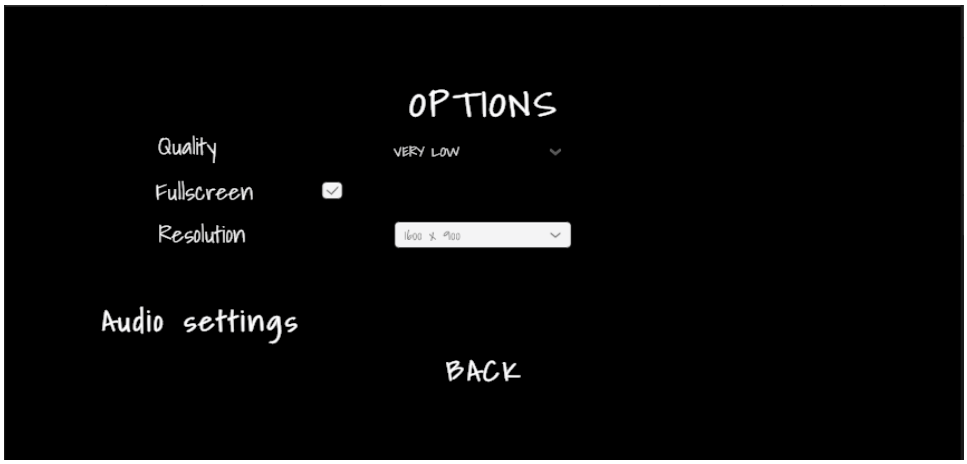
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;


public class StartingCutscene : MonoBehaviour
{
    private bool isSkipped = false;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Space)){
            if(!isSkipped){
                isSkipped = true;
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1 );
            }
        }
    }
}
```

Isječak koda 5. Skripta StartingCutscene

U Start() metodi izvodi naredbu za učitavanje sljedeće scene. U isječku koda skripte StartingCutscene.cs je definirana privatna varijabla isSkipped (hrv. preskočeno je) te ona označava ako je scena preskočena ili nije. U Update() funkciji se provjerava ako je pritisnuta tipka razmak te ako je vrijednost varijable isSkipped se postavlja na istinitu vrijednost, te se učitava nova scena.

5.1.3 Izbornik postavki

Izbornik postavki se koristi za postavljanje rezolucije ekrana, kvalitete grafike i rada preko cijelog zaslona. Kod za upravljanje postavkama će biti prikazan u isječcima koda 6 i 7, a slika 7 prikazuje izgled izbornika postavki.



Slika 7. Snimka ekrana izbornika postavki igrice Silent Nut

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


public class OptionsMenu : MonoBehaviour
{
    public TMP_Dropdown resolutionsDropdown;
    Resolution[] resolutionsArray;
    int pcResolutionIndex = 0;

    void Start(){
        resolutionsArray = Screen.resolutions;
        resolutionsDropdown.ClearOptions();
        List<string> options = new List<string>();

        for(int i=0;i<resolutionsArray.Length;i++){
            string option = resolutionsArray[i].width+ " x " +resolutionsArray[i].height;
            options.Add(option);

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

        resolutionsDropdown.AddOptions(options);
        resolutionsDropdown.value = pcResolutionIndex;
        resolutionsDropdown.RefreshShownValue();

    }

    public void updateResolution(int resolutionIndex){
        Resolution resolution = resolutionsArray[resolutionIndex];
        Screen.SetResolution(resolution.width,resolution.height, Screen.fullScreen);
    }
```

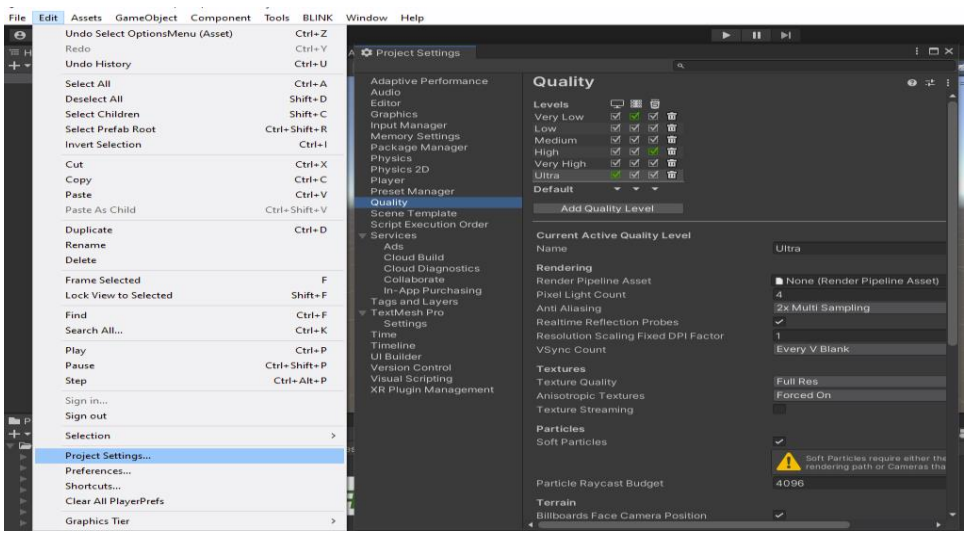
Isječak koda 6. Skripta OptionsMenu.cs 1.dio

U skripti OptionsMenu.cs se dohvaćaju sve dostupne rezolucije zaslona te se dodaju u padajući izbornik s rezolucijama. Funkcija updateResolution() koja prima cjelobrojnu vrijednost kao parametar se koristi za postavljanja određene rezolucije kao odabrane rezolucije.

```
    public void setGraphicQuality(int qualityIndex){
        QualitySettings.SetQualityLevel(qualityIndex);
    }

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

Isječak koda 7. Skripta OptionsMenu.cs 2.dio

Metoda setGraphicQuality(int qualityIndex) tipa void prima cjelobrojni parametar te postavlja novu razinu kvalitete grafike. Potrebno je postaviti postavke kvalitete koje se postavljaju tako da se slijedi sljedeća putanja: Uredi> Postavke projekta > kvaliteta. Tamo je moguće dodati, ukloniti ili urediti postavke kvalitete što je vidljivo na slici 8. Metoda fullscreenToggle() prima parametar tipa bool te ako je vrijednost varijable isFullscreen istinita omogućuje se način rada aplikacije preko cijelog zaslona, a ako je vrijednost parametra lažna onemogućuje se prikaz preko cijelog zaslona.



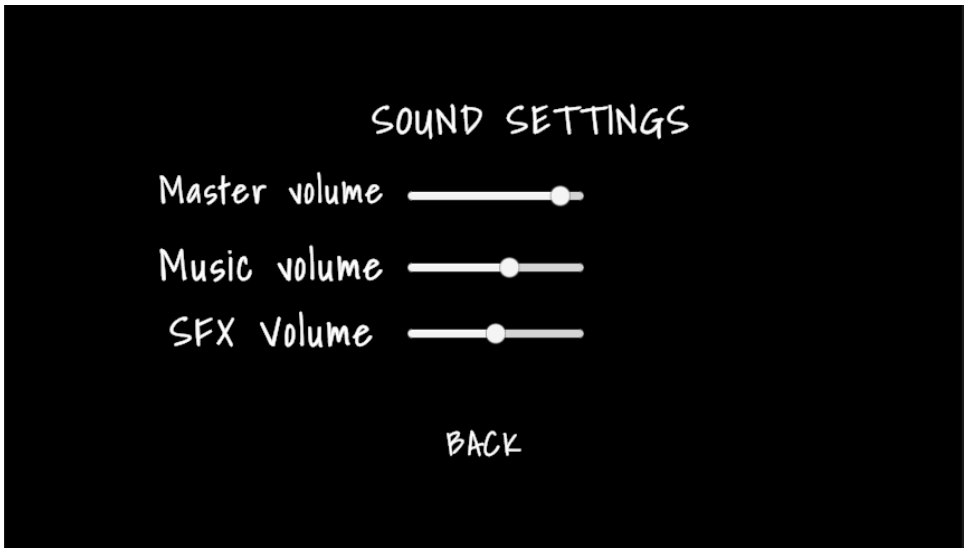
Slika 8. Snimka ekrana postavki kvalitete igrice Silent Nut u razvojnom okruženju

Pritiskom na padajući izbornik moguće je promijeniti kvalitetu igre. Dostupne kvalitete su very low (hrv. jako nisko), low (hrv. nisko), medium (hrv, srednje), high (hrv. visoko) i ultra. Igrač također može označiti želi li igricu igrati u punom zaslonu ili u prozoru (eng. window). Ako je u okviru stavljena kvačica to znači da će igra biti igrana u punom zaslonu. Rezoluciju je moguće odabrati pritiskom na padajući izbornik pokraj teksta Resolution

(hrv. rezolucija) te kada se padajući izbornik otvori moguće je odabrati željenu rezoluciju. Pritiskom na Audio settings (hrv. postavke zvuka) otvara se dodatni izbornik opcija u kojem se može podesiti zvuk.

5.1.2.1 Postavke zvuka

Izbornik postavki zvuka je moguće vidjeti na slici 9.



Slika 9. Snimka ekrana izbornika postavki jačine zvuka igrice Silent Nut

Postavke zvuka se postavljaju putem klizača (eng. slider). Master volume (hrv. glavni volumen) je odgovoran za sav zvuk u igrici te ako je taj zvuk postavljen na 0, sav zvuk će biti postavljen na vrijednost 0 odnosno neće se čuti. Music volume (hrv. volumen glazbe) i SFX volume (hrv. volumen zvučnih efekata) su prikvačeni na glavni volumen, ali mijenjanje njihove vrijednosti utječe na zvukove koji se nalaze na Music i SFX mikserima. Mijenjanje njihovih vrijednosti će biti prikazanu u isječku koda 8.

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

public class SoundOptionsMenu : MonoBehaviour
{

    public AudioMixer audioMixer;

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

    public void SetMusicVolume(float volume){
        audioMixer.SetFloat("MusicVolume",volume);
    }
    public void SetSFXVolume(float volume){
        audioMixer.SetFloat("SFXVolume",volume);
    }

}
```

Isječak koda 8. Skripta SoundOptionsMenu.cs

Skripta SoundOptionsMenu.cs se koristi za mijenjanje jačine zvuka te se sastoji od javne varijable AudioMixer i tri funkcije tipa void koje su zadužene za postavljanje jačine zvuka za određeni mikser. Funkcija SetMasterVolume(float volume) prima float vrijednost putem klizača te postavlja vrijednost miksera čiji je izdvojeni parametar naziva MasterVolume. Funkcije SetMusicVolume(float volume) i SetSFXVolume(float volume) čine istu stvar samo za različiti izdvojeni parametar miksera.

5.1.2.2 Upravitelj zvuka

Upravitelj zvuka koristi dvije skripte za upravljanje zvukom, a to su Sound.cs i AudioManager.cs. Kod koji je potreban za upravljanje zvukom bit će prikazan u isječcima koda 9, 10, 11 i 12 od kojih se isječak koda 9 odnosi na Sound.cs skriptu, a ostali na AudioManager.cs skriptu.

```
using UnityEngine.Audio;
using UnityEngine;

[System.Serializable]
public class Sound
{

    public string name;
    public AudioClip clip;

    [Range(0f,2f)]
    public float volume = 1.0f;
    [Range(0.1f,3f)]
    public float pitch;

    public bool loop;
    public float soundLength;
    public AudioSource source;
    public AudioMixerGroup mixerGroup;

}
```

Isječak koda 9. Skripta Sounds.cs

Varijabla za naziv zvuka name se odnosi na naziv audio datoteke. Varijabla clip se odnosi na audio zapis. Varijabla volume je tipa float i raspona je od 0 do 2. Varijabla pitch odnosno visina tona je također varijabla tipa float koja ima raspon od 0.1 do 3. Varijabla koja se koristi za neprestano ponavljanje zvučnog zapisa je loop koja je tipa bool i ako je istinita onda će se zapis neprestano ponavljati, a ako je lažna onda se zvučni zapis neće neprestano ponavljati.

```
using UnityEngine.Audio;
using UnityEngine;
using System;
using System.Collections;


public class AudioManager : MonoBehaviour
{
    public Sound[] sounds;
    public PauseMenu pauseMenu;
    // Start is called before the first frame update
    void Awake()
    {
        foreach(Sound s in sounds){
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
            s.soundLength = s.clip.length;
            s.source.outputAudioMixerGroup = s.mixerGroup;
        }
    }

    void Start(){
        this.playSound("BackgroundMusic");
        StartCoroutine(playWithDelay("WhatIsThisPlaceHowDoIGetOut",5.0f));
    }

    void Update()
    {
        if(pauseMenu.gameIsPaused){
            pauseAllSound();
        }else{
            resumeAllSound();
        }
    }
```

Isječak koda 10. Skripta AudioManager.cs 1.dio

Unutar Awake() funkcije se koristi foreach petlja koja je iste funkcionalnosti kao for petlja s čišćom sintaksom. Za svaki zvuk (eng. sound) u polju zvukova (eng. sounds) se postavljaju odgovarajuće vrijednosti (Amlin, 2021).

Vrijednost s.source se postavlja na komponentu izvora zvuka (eng. audio source) te se sve vrijednosti postavljaju na odgovarajuće vrijednosti izvora zvuka. U Start() metodi se pokreće zvuk s nazivom „BackgroundMusic“ te se pokreće korutina koja nakon 5 sekundi pokreće zvuk s nazivom „WhatIsThisPlaceHowDoIGetOut“. U Update() funkciji se pauzira sav zvuk ako je igra pauzirana i nastavlja se ako nije.

```
public void playSound(string soundName){
    Sound s = Array.Find(sounds,sound=> sound.name == soundName);
    if(s == null){
        return;
    }
    s.source.Play();
}
public void stopSound(string soundName){
    Sound s = Array.Find(sounds,sound=> sound.name == soundName);
    if(s == null){
        return;
    }
    s.source.Stop();
}

public float getSoundLength(string soundName){
    Sound s = Array.Find(sounds,sound=> sound.name == soundName);
    if(s == null){
        return 0;
    }
    return (float)s.soundLength;
}

public void pauseAllSound(){
   foreach(Sound s in sounds){
    s.source.Pause();
   }
}
public void resumeAllSound(){
   foreach(Sound s in sounds){
    s.source.UnPause();
   }
}
```

Isječak koda 11. Skripta AudioManager.cs 2.dio

U metodi PlaySound(string soundName) se prosljeđuje string parametar soundName pretražuje se polje te se traži objekt zvuka čija vrijednost parametra name je jednaka proslijeđenoj vrijednosti. Ako se pronađe vrijednost pokreće se zvuk, a ako nema vrijednost prekida se daljnje izvršavanje funkcije pomoću return (Pagán Dávila, 2021).

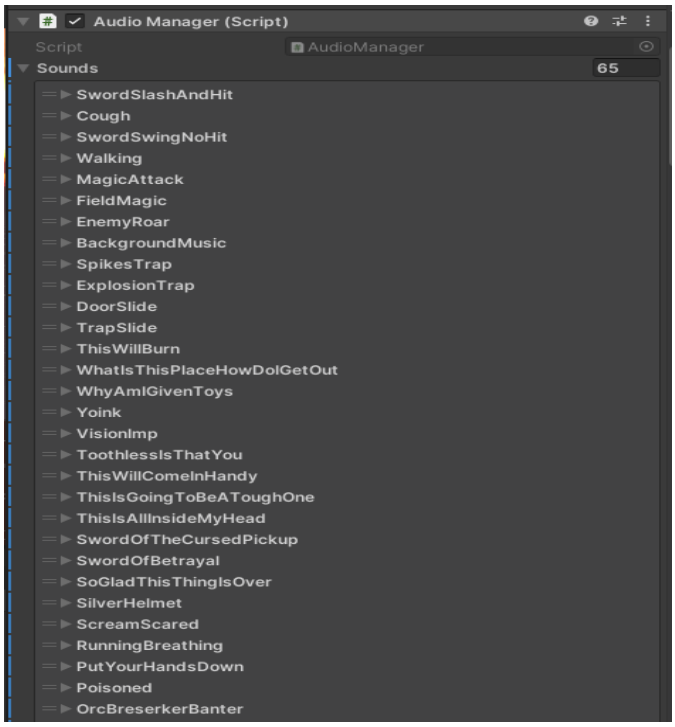
Metoda stopSound(string soundName) radi na identičan način kao i metoda PlaySound(string soundName), osim što umjesto pokretanja zvuka ona ga zaustavlja. Funkcija getSoundLength(string soundName) tipa float prima string vrijednost te se pretražuje polje i pokušava se pronaći zvuk kojem je vrijednost varijable name jednaka proslijeđenoj varijabli. Ako se pronađe, funkcija će vratiti float vrijednost, a ako ne vratit će vrijednost 0.

Metoda pauseAllSound() prolazi kroz svaki element polja zvukova te ga pauzira, dok metoda resumeAllSound() prolazi kroz svaki element polja zvukova te prekida pauzu.

```
    IEnumerator playWithDelay(string soundName,float delay){
        yield return new WaitForSeconds(delay);
        Sound s = Array.Find(sounds,sound=> sound.name == soundName);
        if(s != null){
            s.source.Play();
        }
    }
}
```

Isječak koda 12. Skripta AudioManager.cs 3.dio

Korutina playWithDelay(string soundName, float delay) prima dva argumenta, prvi argument je tipa string i predstavlja naziv zvuka, a drugi argument je iznos odgode (eng. delay) tipa float koja predstavlja broj sekundi koje će proći prije nego što se izvrši korutina. Slika 10 prikazuje upravitelj zvukova u inspektoru.



Slika 10. Prikaz upravitelja zvuka u inspektoru

5.1.4 Statistika o odigranim igrama

Statistika igre predstavlja koliko je igrač igri odigrao te koliko ih je pobijedio, izgubio i pobijedio za redom. Isječci koda 13 i 14 prikazuju kod skripte RunStats.cs koja će u nastavku biti objašnjena.

Skripta RunStats.cs sadrži četiri cjelobrojne varijable. Varijabla totalRuns označuje koliko je puta igrač igrao igricu, varijabla wins označuje koliko je puta igrač pobijedio igricu, varijabla losses označuje koliko je puta igrač izgubio te varijabla winsInARow označuje koliko je puta igrač za redom pobijedio igricu.

```
public class RunStats : MonoBehaviour
{

    public int totalRuns;
    public int wins;
    public int losses;
    public int winsInARow;

    void Start()
    {
        totalRuns = PlayerPrefs.GetInt("TotalRuns");
        losses = PlayerPrefs.GetInt("RunLosses");
        wins = PlayerPrefs.GetInt("RunWins");
        winsInARow = PlayerPrefs.GetInt("WinsInARow");

        if(totalRuns != losses + wins){
            losses = totalRuns - wins;
            winsInARow = 0;
            PlayerPrefs.SetInt("WinsInARow",winsInARow);
            PlayerPrefs.SetInt("RunLosses",losses);
        }
    }

    public void StartARun(){
        ++totalRuns;
        PlayerPrefs.SetInt("TotalRuns",totalRuns);
    }

    public void RestartARun(){
        ++losses;
        winsInARow = 0;
        PlayerPrefs.SetInt("WinsInARow",winsInARow);
        PlayerPrefs.SetInt("RunLosses",losses);
        ++totalRuns;
        PlayerPrefs.SetInt("TotalRuns",totalRuns);
    }
```

Isječak koda 13. Skripta RunStats.cs 1.dio

Navedene varijable su pohranjene u PlayerPrefs, odnosno klasu koja pohranjuje postavke igrača između sesija igre. U Start() metodi se varijable postavljaju na vrijednosti spremljene u klasi PlayerPrefs pod odgovarajućim nazivima te se provjerava ako je zbroj

poraza i pobjeda jednak broju igri. Ako zbroj nije jednak, potrebno je postaviti broj poraza na razliku pobjeda i ukupnih igri zato što igrač može prekinuti izvršavanje aplikacije na način na koji nije implementirano, primjerice korištenjem upravitelja zadataka ili pritiskom tipki ALT i F4. Metoda StartARun() uvećava vrijednost totalRuns te je pohranjuje u PlayerPrefs. Metoda RestartARun() uvećava vrijednost losses jer je igrač izgubio, postavlja vrijednost winsInARow na vrijednost 0 te povećava vrijednost totalRuns za jedan jer je pokrenuta nova igra. Također se ti podatci spremaju u PlayerPrefs.

```
    public void QuitARun(){
        ++losses;
        winsInARow = 0;
        PlayerPrefs.SetInt("WinsInARow",winsInARow);
        PlayerPrefs.SetInt("RunLosses",losses);
    }

    public void WinARun(){
        ++wins;
        ++winsInARow;
        PlayerPrefs.SetInt("RunWins",wins);
        PlayerPrefs.SetInt("WinsInARow",winsInARow);

    }

    public void LoseARun(){
        ++losses;
        winsInARow = 0;
        PlayerPrefs.SetInt("WinsInARow",winsInARow);
        PlayerPrefs.SetInt("RunLosses",losses);
    }

}
```

Isječak koda 14. RunStats.cs 2.dio

Metoda QuitARun() povećava vrijednost varijable losses za jedan i postavlja vrijednost varijable winsInARow na vrijednost 0 te podatke sprema u PlayerPrefs. Metoda WinARun() se poziva kada igrač pobjedi igricu te se uvećavaju vrijednosti varijabli wins i winsInARow za jedan te se potom spremaju u PlayerPrefs. Metoda LoseARun() se poziva kada igrač umre te se varijabla losses uvećava za jedan, a varijabla winsInARow se

postavlja na vrijednost nula. Te vrijednosti se spremaju u PlayerPrefs. Prikaz podataka o igrama se je prikazan na slici 11.

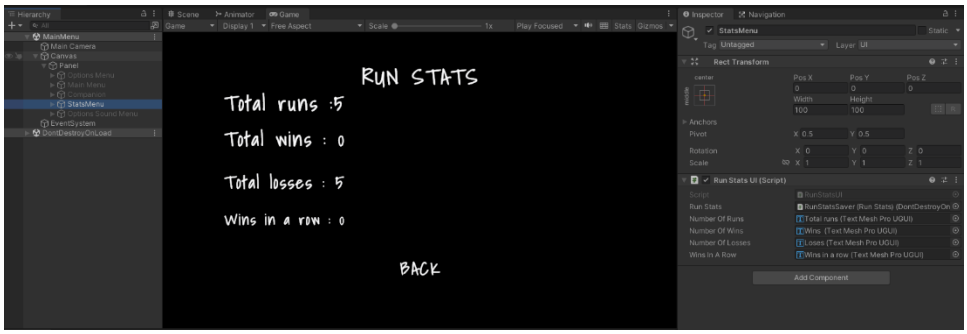
Slika 11. Snimka ekrana statistike igračevih igara

Izbornik o statistici igre prikazuje koliko je igrač puta odigrao igru te koliko ju je puta pobijedio i izgubio. Također prikazuje koliko igrač ima pobjeda za redom (eng. wins in a row). RunStatsUI.cs skripta je prikazana u isječku koda 15.

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


public class RunStatsUI : MonoBehaviour
{
    public RunStats runStats;
    public TMP_Text numberOfRuns;
    public TMP_Text numberOfWins;
    public TMP_Text numberOfLosses;
    public TMP_Text winsInARow;

    void Start()
    {
        numberOfRuns.text = "Total runs :" + runStats.totalRuns.ToString();
        numberOfWins.text = "Total wins : " + runStats.wins.ToString();
        numberOfLosses.text = "Total losses : "+runStats.losses.ToString();
        winsInARow.text = "Wins in a row : "+runStats.winsInARow.ToString();


    }
}
```

Isječak koda 15. Skripta RunStatsUI.cs

Prikaz statistike igri se izvodi tako da su UI elementi dodijele skripti RunStatsUI. Svakom od dodijeljenih elemenata se mijenja vrijednost tekst komponente. Varijable iz skripte RunStats su tipa int, odnosno cjelobrojni podatci, dok tekst je tipa string, što znači da je potrebno promijeniti tip podataka iz cjelobrojnih u string i to se vrši pomoću ToString() funkcije. Dodijeljeni UI elementi skripti RunStats.cs su vidljivi na slici 12.



Slika 12. Prikaz dodijeljenih UI elemenata skripti RunStats.cs

U inspektoru je moguće vidjeti da su UI TextMeshPro elementi proslijeđeni skripti RunStatsUI.cs s odgovarajućim vrijednostima. Skripta ButtonSounds.cs se koristi za pokretanje zvuka nakon što igrač pritisne gumb na ekranu i kod skripte je prikazan u isječku koda 16.

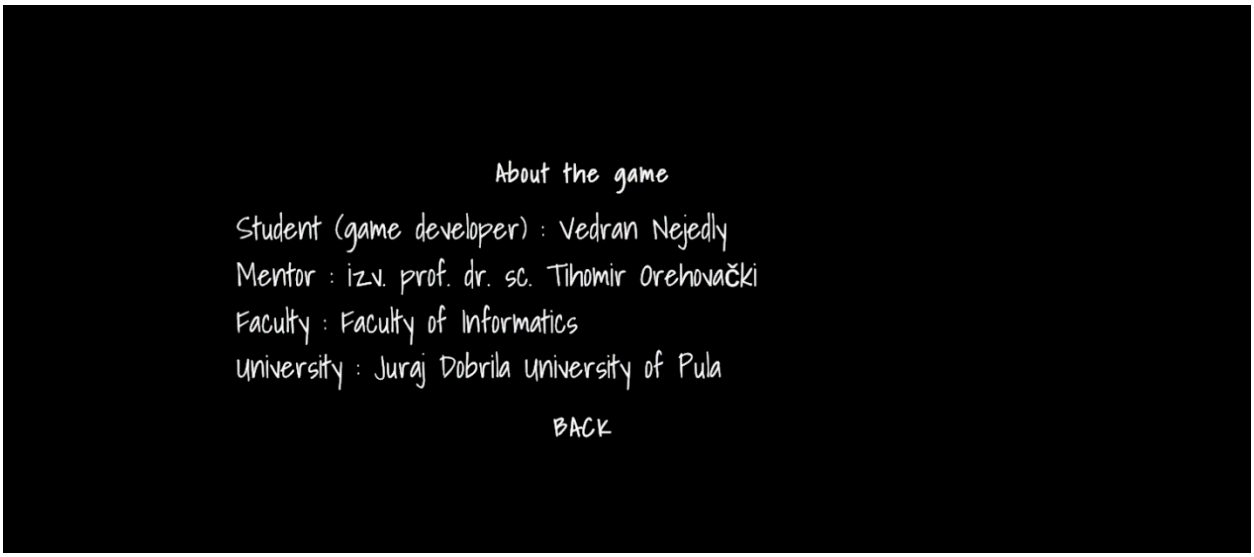
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ButtonSound : MonoBehaviour
{
    private Button button;
    private AudioSource audioSource;
    // Start is called before the first frame update
    void Start()
    {
        button = gameObject.GetComponent<Button>();
        button.onClick.AddListener(TaskOnClick);
        audioSource = GameObject.Find("Canvas").GetComponent<AudioSource>();

    }

    void TaskOnClick(){
        audioSource.Play();
    }

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

    }
}
```

Isječak koda 16 Skripta ButtonSound.cs

Zvuk se nalazi na objektu igre platno te ga se dohvaća tako da se pomoću Find(string name) funkcije koja prima argument tipa string pronalazi objekt u igri s vrijednošću proslijeđene varijable te se pomoću GetComponent () dohvaća komponenta izvor zvuka. Naredbom button.onClick.AddListener(TaskOnClick) se kliku gumba dodaje poziv funkcije TaskOnClick() tipa void u kojoj se pokreće izvor zvuka.

5.1.5 Informacije o igrici

Slika 13 prikazuje izbornik o igri (eng. about the game) odnosno podatke o studentu, mentoru, fakultetu te sveučilištu.



Slika 13. Prikaz izbornika about the game

Klikom na „o igri“ u glavnom izborniku igrice otvara se novi panel u kojem se nalaze informacije u studentu, mentoru, fakultetu te sveučilištu. Svi elementi su tipa TextMashPro.

5.2 Igrač

U ovom dijelu diplomskog rada će biti opisano kako se igrač kreće kroz svijet, na koji način koristi magije te kako one funkcioniraju. Također će biti objašnjen način na koji radi sistem životnih bodova kod igrača.

5.2.1 Kretanje igrača kroz svijet

Kako bi se igrač mogao kretati kroz svijet važno je uzeti u obzir dvije stvari, a tu su kretanje kamere te pomicanje samog objekta igrača na x, y i z osima.

Upravljanje kamerom je ostvareno tako da je kamera prikvačena na objekt igre igrač, odnosno glavna kamera (eng. main camera) je dijete objektu igre igrač. Na kameru je dodana skripta pod nazivom MouseLook.cs koja je zadužena za pomicanje kamere prikazana u isječku koda 17.

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

public class MouseLook : MonoBehaviour
{

    public float mouseSensitivity = 100f;
    //Objekt koji mozemo rotirati
    public Transform playerBody;
    float xRotation= 0f;
    // Start is called before the first frame update
    void Start()
    {
        //Kursor je zakljucan u sredini ekrana i nije vidljiv.
        Cursor.lockState = CursorLockMode.Locked;
    }

    // Update is called once per frame
    void Update()
    {
        //Uzimamo input za x i y os te ih mnozimo sa mouseSensitivity i Time.deltaTime
        float mouseXAxis = Input.GetAxis("Mouse X")*mouseSensitivity * Time.deltaTime;
        float mouseYAxis = Input.GetAxis("Mouse Y")*mouseSensitivity * Time.deltaTime;

        xRotation -= mouseYAxis;
        // Rotacija može biti maksimalno -90 stupnjeva i 90 stupnjeva kako kamera se ne bi okrenula kroz igraca.
        // Igrac moze pogledati u nebo i pod ali ne i kroz sebe.
        xRotation = Mathf.Clamp(xRotation,-90f,90f);

        transform.localRotation = Quaternion.Euler(xRotation,0f,0f);
        //Rotacija objekta na x osi
        playerBody.Rotate(Vector3.up * mouseXAxis);

    }
}
```

Isječak koda 17. Skripta MouseLook.cs

Varijabla mouseSensitivity se koristi za brzinu kursora, što je broj veći kursor je osjetljiviji. Varijabla playerBody se odnosi na objekt igrača koji će biti rotiran na x osi odnosno pomicanjem miša u lijevo će rotirati objekt u lijevo, a pomicanje miša u desnu stranu će rotirati igrača u desno. U start metodi je kursor zaključan na sredinu ekrana te je sakriven. U Update() metodi se putem Input.GetAxis() funkcije dobivaju vrijednosti putem kursora za x i y osi. Uz pomicanje kamere nužno je pomicati i objekt igrača te to se čini pomoću PlayerMovement.cs skripte koja je prikazana u isječcima koda 18 i 19.

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

public class PlayerMovement : MonoBehaviour
{
    public CharacterController controller;
    public float movementSpeed = 10f;

    public float gravity = -9.81f;

    public Transform groundCheck;
    public float groundDistance = 0.4f;
    public LayerMask groundMask;
    bool isGrounded;
    public float jumpHeight = 3f;

    Vector3 velocity;
    // Start is called before the first frame update
    void Start()
    {

    }
```

Isječak koda 18. Skripta PlayerMovement.cs 1.dio

Na igraču se nalazi Character controller koji je zadužen za pomicanje igraća pomoću Move() funkcije te on omogućuje jednostavno kretanje igrača koje je ograničeno sudaračima (eng. colliders) te uklanja potrebu za radom s komponentom Rigidbody (Unity 2023).

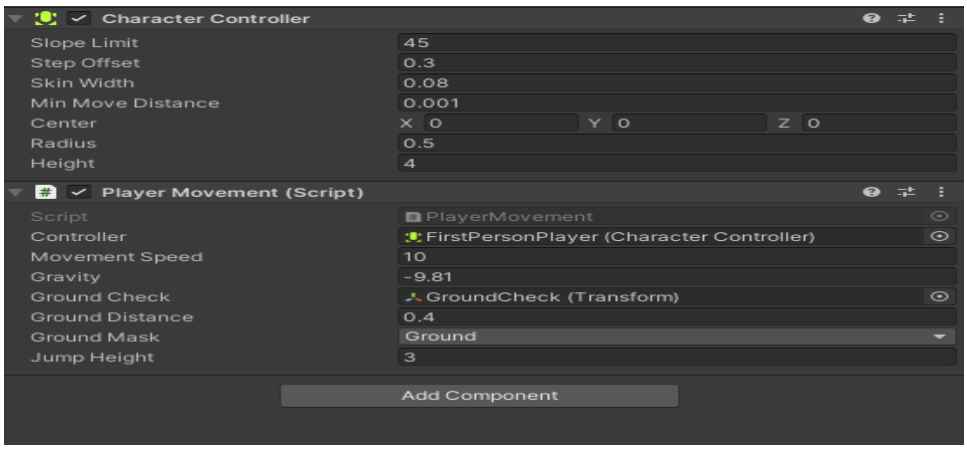
Javna varijabla movementSpeed (hrv. brzina kretanja) tipa float je postavljena na vrijednost 10. Varijabla je javna zato što će se brzina kretanja mijenjati kroz igru. Također je postavljena vrijednost gravitacije i ona iznosi 9.81 te visina skoka (eng. jump height) koja iznosi 3. Definirana je Vector3 varijabla velocity. Vector3 se u Unityju koristi za prosljeđivanje položaja i smjera te sadrži funkcije za izvođenje uobičajenih vektorskih operacija.

```
    // Update is called once per frame
    void Update()
    {
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance,groundMask);
        if(isGrounded && velocity.y<0){
            velocity.y=-2f;
        }

        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        Vector3 move = transform.right * x + transform.forward * z;
        controller.Move(move * movementSpeed * Time.deltaTime);

        if(Input.GetButtonDown("Jump") && isGrounded){
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
        }

        velocity.y += gravity * Time.deltaTime;

        controller.Move(velocity * Time.deltaTime);
    }
}
```

Isječak koda 19. Skripta PlayerMovement.cs 2.dio

Prva linija u Update() metodi provjerava ako je igrač prizemljen odnosno ako se sudarač sudara s tlom. Maska sloja (eng. layer mask) je cijeli broj od 32 bita kojem su sve vrijednost nula. Kada je aktiviran svi bitovi poprimaju vrijednost jedan. Koristi se kada se želi izolirati neki sloj te u ovom slučaju želi se izolirati sloj tla kako bi se provjerilo da li je igrač prizemljen. Ako pozicija groundCheck objekta dodiruje tlo odnosno njihovi sudarači se preklapaju vrijednost isGrounded će biti istinite vrijednosti (Watts, 2022).

Igrač se pokreće putem Move() metode kojoj se predaje Vector3 sa smjerom i pozicijom kretanja koji se množi s brzinom kretanja igraća kroz vrijeme. Ako se pritisne tipku razmak (eng. space) igrač će skočiti tako što se na njegovu y os doda vrijednost velocity.y. Slika 14 prikazuje komponente Character Controller i PlayerMovement.cs u inspektoru.



Slika 14. Prikaz komponenti CharacterController i PlayerMovement na objektu igre Player

U inspektoru se mogu postaviti vrijednosti visine skoka, brzine kretanja, gravitacije te je nužno postaviti Transform objekta GroundCheck kako bi se provjeravalo da li igrač dodiruje tlo te GroundMask odnosno masku sloja tla.

5.2.2 Sustav igračevih životnih bodova

Kako bi igrač mogao primiti štetu, liječiti se, preživjeti ili umrijeti, on mora imati životne bodove. Skripta koja je odgovorna za ispravan rad životnih bodova je PlayerHealth.cs i ona će biti prikazana u isječku koda 20 te potom objašnjena.

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

public class PlayerHealth : MonoBehaviour
{
    public int maxHealth = 10;
    public int health = 10;

    public void playerDie(){
        if(health<=0){
            Destroy(gameObject);
        }
    }

    public void upadateMaxHealth(int modificator){
        maxHealth +=modificator;
        health +=modificator;
    }

    public void updateHealth(int modificator){
        health+=modificator;
        if(health>maxHealth){
            health=maxHealth;
        }
        if(health<=0){
            playerDie();
        }
    }

    public void swordCurse(){
        health=1;
    }

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

Isječak koda 20. Skripta PlayerHealth.cs

Sustav životnih bodova za igrača je napravljen tako da mu je dodijeljen maxHealth odnosno brojčana vrijednost koja predstavlja njegovo zdravlje. Varijabla health predstavlja igračevo trenutno zdravlje. Ona ne može biti veća od varijable maxHealth, ali može doći do iste vrijednosti. Ako vrijednost životnih bodova padne na vrijednost 0 igrač umire te igra završava (Hijazi, 2021).

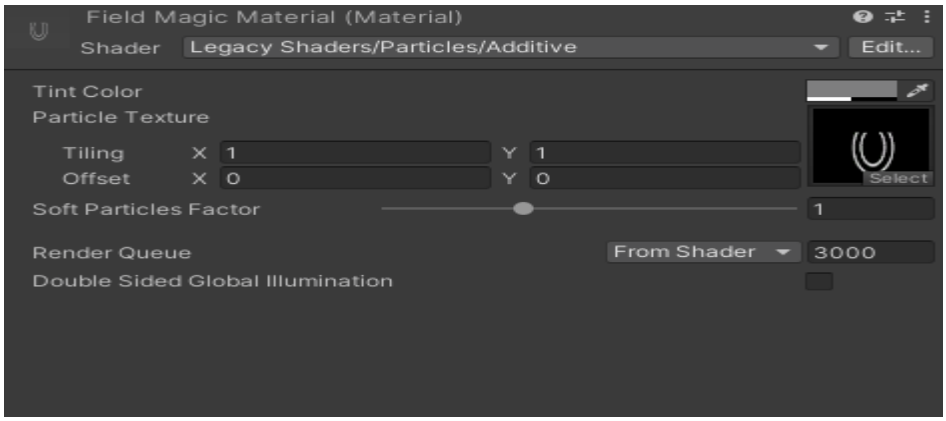
Funkcija updateMaxHealth(int modificator) prima cjelobrojnu vrijednost modificator-a koja uvećava igračeve maksimalne životne bodove (maxHealth varijablu) i trenutne životne bodove igrača (health varijablu) za proslijeđenu vrijednost modificator. Funkcija updateHealth(int modificator) također kao i prethodna funkcija prima cjelobrojnu vrijednost modificator koja se dodaje vrijednosti igračevih trenutnih životnih bodova. Ako vrijednost trenutnih životnih bodova postane veća od vrijednosti maksimalnih životnih bodova, vrijednost trenutnih životnih bodova se postavlja na vrijednost maksimalnih životnih bodova. Ako je vrijednost trenutnih životnih bodova (health varijable) manja ili jednaka broju nula, igrač umire. Metoda SwordCurse() je metoda koja se poziva pri uzimanju određene stvari (eng. item) te ona postavlja vrijednost trenutnih životnih bodova igrača na 1.

5.2.3 Sustav magije

Magični krug je slika napravljena u programu bojanje (eng. Paint) koristeći alat za eklipse i prikazan je na slici 15. Napravljena su dva kruga od kojih je manji krug unutar veće kruga. Potom im je obrisan dio kružnice.

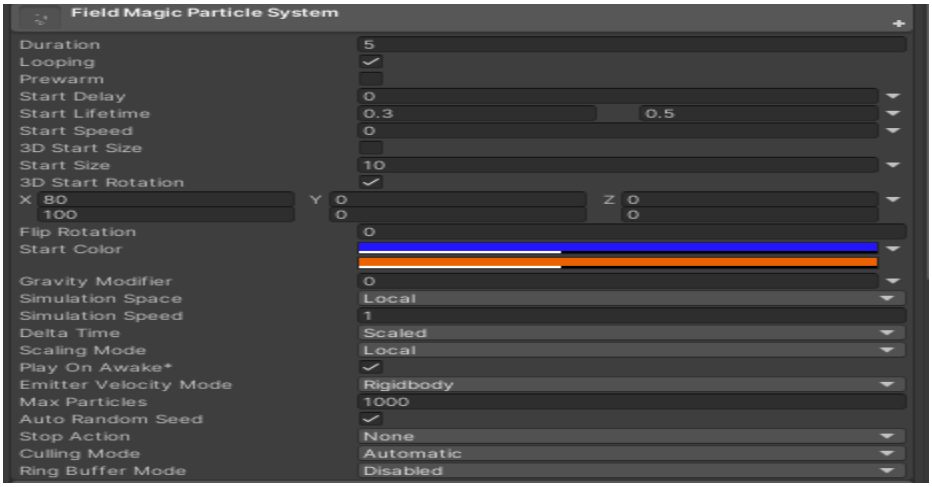
Slika 15. Magični krug prije pretvorbe u sustav čestica

Potom je pomoću magičnog kruga prikazanog na slici 15 napravljen novi materijal koji je prikazan na slici 16.



Slika 16. Izrađeni materijal korištenjem slike magičnog kruga

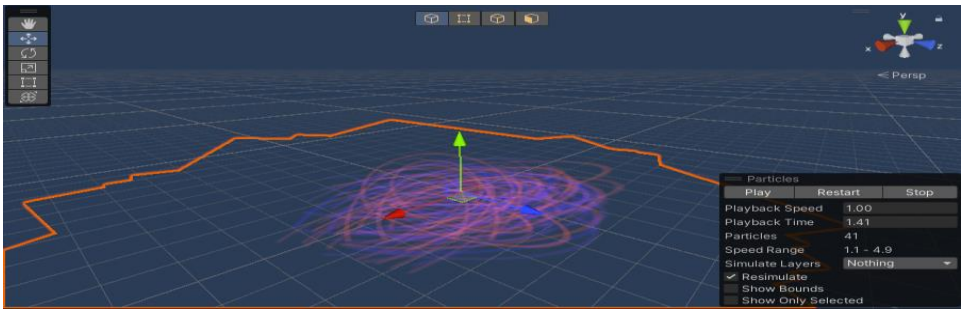
Nakon toga je slika dodana u Unity okruženje te je napravljen novi materijal kojem je kao tekstura čestice postavljena slika 15. Nakon toga napravljen je novi efekt sustav čestica (eng. particle system) te je na njega postavljen prethodno napravljeni materijal. Na slici 17 je prikazan sustav čestica magičnog kruga.



Slika 17. Sustav čestica magičnog krug

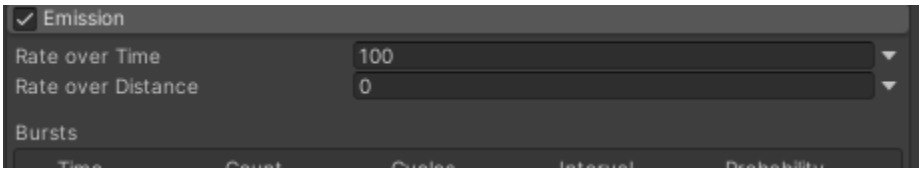
Vrijeme trajanja (eng. duration) predstavlja koliko će sustav čestica trajati u sekundama. „Looping“ je postavljen na istinitu vrijednost što znači da kada sustav čestica završi, pokrenuti će se ispočetka. Početna odgoda (eng. start delay) je postavljena na vrijednost nula što znači da sustav čestica nema vrijeme čekanja prije nego što se pokrene. Početni životni vijek (eng. start lifetime) je vrijednost koja predstavlja koliko će dugo čestica živjeti. U slučaju sustava čestica magičnog polja vrijeme života jedne čestice je između 0.3 i 0.5 sekundi. Početna boja (eng. start color) je postavljena na dvije boje koje su također vidljive

na slici 17. Početna rotacija (eng. start rotation) je postavljena samo na x osi i može biti bilo koja vrijednost u rasponu od 80 do 100. Pokrenuti sustav čestica je prikazan na slici 18.



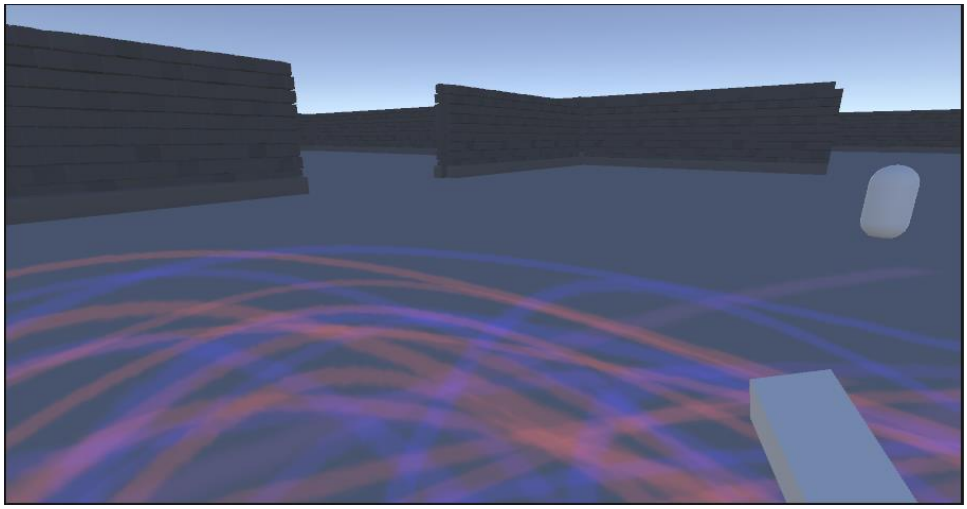
Slika 18. Pokrenuti sustav magičnih čestica

Na slici 19 je prikaz emisije sustava čestica kojem je promijenjena vrijednost atributa „Rate over time“, što znači da će se stvoriti više čestica u istom vremenskom periodu.



Slika 19. Emisija sustava čestica.

Na slici 20 je prikazano magično polje koje igrač aktivirao. Vidljiv je veliki broj lukova koji zbog rotacije na x osi stvaraju izgled koji je prikazan na slici.



Slika 20. Prikaz magičnog polja unutar igre

U nastavku će biti objašnjena skripta MagicCircle.cs koju prikazuje isječak koda 21.

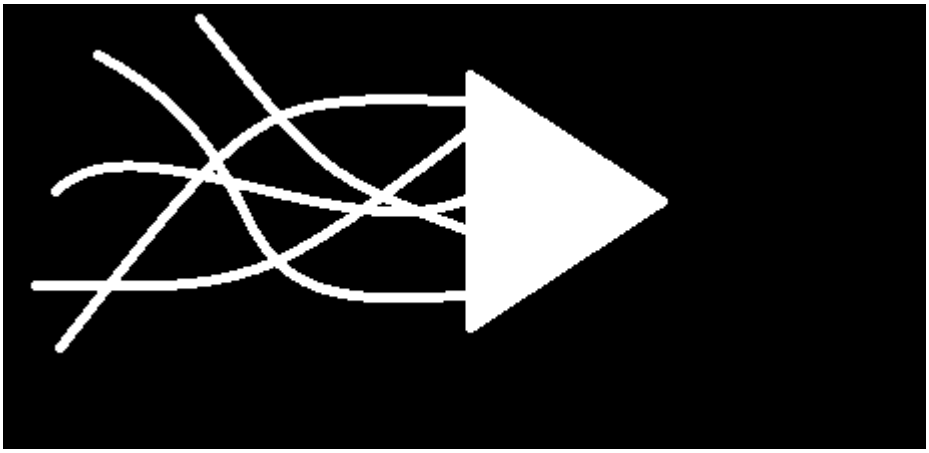
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MagicCircle : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }

    private void OnTriggerEnter(Collider other){
        if(other.tag=="Enemy"){
            Enemy enemyHealth = other.GetComponent<Enemy>();
            if(enemyHealth != null){
                // health.Damage(damage);
                enemyHealth.Damage(1);
            }
        }
    }
}
```

Isječak koda 21. Skripta MagicCircle.cs

Sudaračima (eng. colliders) se može postaviti vrijednost isTrigger na istinitu vrijednost te ako je ta vrijednost postavljena na istinitu onda se blokira Rigidbody komponenta i potrebno je koristiti jednu od OnTrigger funkcija. Razliku između OnTriggerEnter() i OnCollisionEnter() se može promatrati tako da kada se želi aktivirati neki zvuk koristi se OnTriggerEnter(), a kada se neki objekt želi sudariti sa zidom koristi se OnCollisionEnter() (Dávila, 2021).

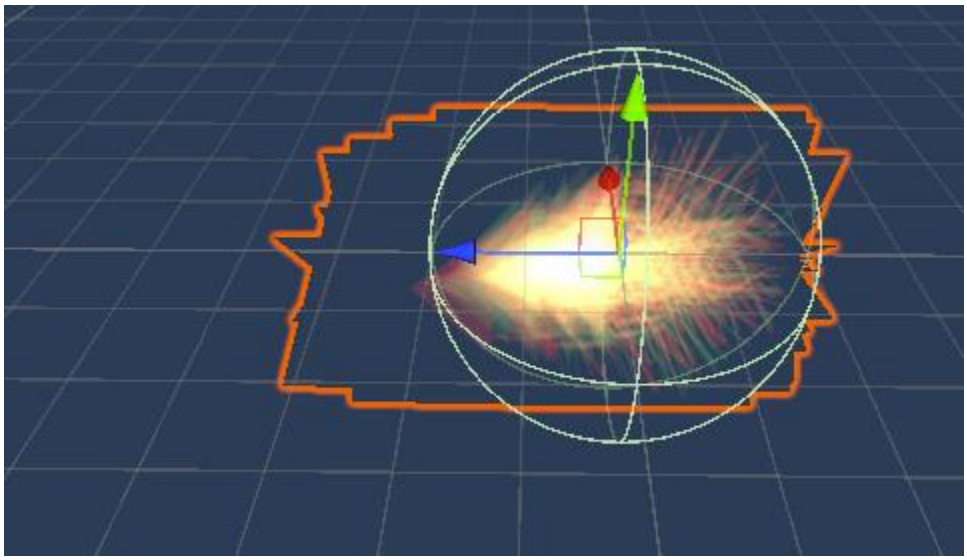
MagicCircle.cs sadži OnTriggerEnter(Collider other) metodu u kojoj se provjerava da li je oznaka objekta koji je ušao u sudarač jednak oznaci „Enemy“. Ako je dohvaća se komponenta EnemyHealth s objekta igre neprijatelj te se provjerava da li komponenta postoji. Ako komponenta postoji, metodi Damage(int dmg) komponente Enemy se prosljeđuje cjelobrojna vrijednost koji nanosi štetu njegovim životnim bodovima.

U skripti PlayerMagic.cs postoje definirane varijable canCastMagicCircle tipa bool koja je postavljena na istinitu vrijednost i cjelobrojna varijabla magicCircleCounter koja je postavljena na 0. Također su definirana dva objekta igre magicCircle i magicCircleVisual. Varijabla magicCircleVisual se odnosni na prethodno navedeni sustav čestica koji stvara lukove oko igrača kako bi dočaralo da se oko igrača nalazi magično polje, a magicCircle varijabla se odnosi na sudarač koji se nalazi na magičnom polju. Unutar funkcije Update() se provjerava ako je igrač pritisnuo tipku E i ako je varijabla canCastMagicCircle istinite vrijednosti onda se izvršava metoda castMagicCircleCycle(). Slika 21 prikazuje magičnu strijelu koja je napravljena u programu bojanje na način da je napravljen trokut koji je ispunjen bijelom bojom te su nacrtane zakrivljene linije koje izlaze iz tog trokuta.



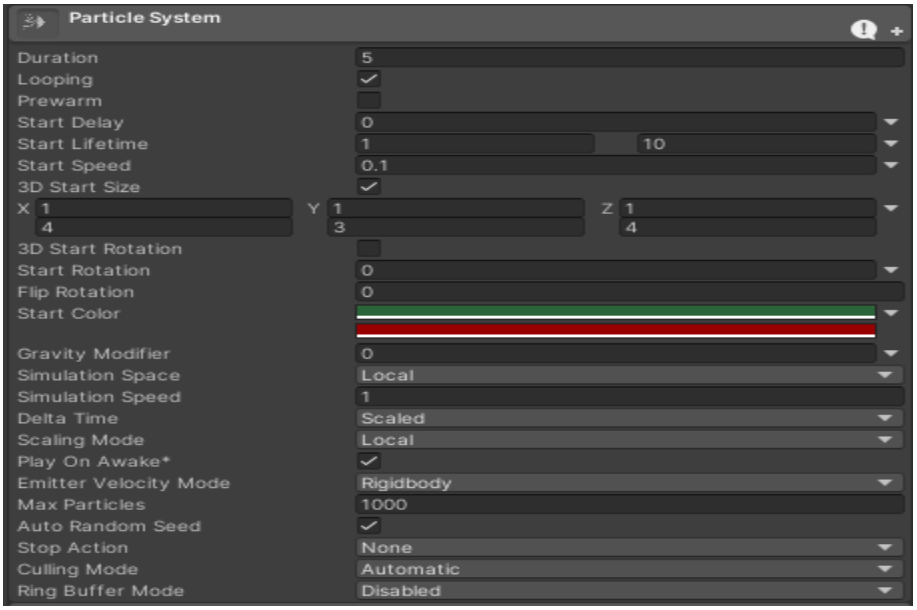
Slika 21. Slika magične strijele prije pretvorbe u sustav čestica

Magični projektil je napravljen tako da je stvoren novi materijal kojem je shader postavljen na additive te je dodana gore navedena slika na taj materijal. Potom je taj materijal dodan na sistem čestica. Sustav čestica je dinamičan sustav u kojem se tijekom vremena čestice u sustavu mijenjanju. One mijenjaju svoj oblik, poziciju, odnosno kreću se i nakon nekog vremena prestaju postojati dok se u sustav dodaju nove čestice. Svaka čestica ima životni ciklus u koji spada stvaranje čestice, njeno kretanje te smrt te čestice. Na slici 22 prikazan je pokrenuti sustav čestica magične strijele.



Slika 22. Pokrenuti sustav čestica magične strijele

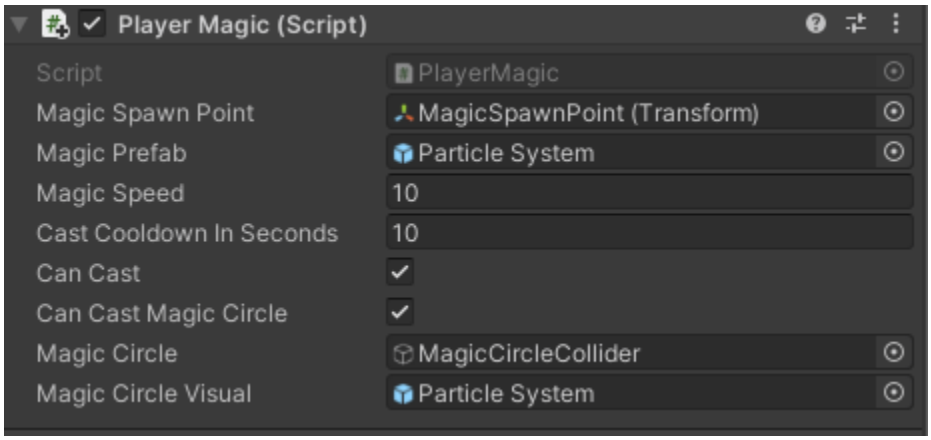
Slika 23 prikazuje parametre sustava čestica magične strijele.



Slika 23. Sustav čestica magične strijele

Postavljene su boje, veličina, početna brzina te „looping“ sustava čestica. Kako bi se postiglo da 2D slika izgleda kao 3D objekt potrebno je povećati stopu tijekom vremena (eng. rate over time) te postavljanjem rotacije na x osi. Kada je brzina rotacije dovoljno velika, 2D objekt će izgledati kao 3D objekt.

Kako bi igrač mogao koristiti magiju potrebno je dodijeliti komponentu PlayerMagic objektu igre igrač (eng. player) i to je prikazano na slici 24.



Slika 24. Prikaz PlayerMagic komponente na igraču

Varijable koje su korištene za magični projektil su magicSpawnPoint, magicPrefab, magicSpeed, canCast i castCooldownInSeconds. MagicSpawnPoint je mjesto gdje će se stvoriti projektil koje je prikvačeno na igrača kao prazan objekt igre. Objekt igre magicPrefab je prethodno pokazani sustav čestica napravljen od slike trokuta i linija koje izlaze iz trokuta. Varijabla magicSpeed tipa float određuje brzinu kretanja projektila. Varijabla castCooldownInSeconds označava koliko sekundi mora proći prije nego što igrač ponovo može izvesti magiju. Bool varijabla canCast označava može li igrač izvesti magiju te je postavljena na istinitu vrijednost jer s početkom igrice igrač je može odmah izvesti. Skripta PlayerMagic.cs je prikazana u isječcima koda 22 i 23.

```
// Update is called once per frame
void Update()
{
        if(Input.GetKeyDown(KeyCode.Q)){
            castAttackMagic();
        }
        if(Input.GetKeyDown(KeyCode.E)){
            if(canCastMagicCircle){
                castMagicCircleCycle();
            }
        }
}

public void reduceCooldown(float cdReduction){
    castCooldownInSeconds-=cdReduction;
}

void castAttackMagic(){
    if(!canCast){
        return;
    }
    var magic  = Instantiate(magicPrefab,magicSpawnPoint.position,magicSpawnPoint.rotation);
    magic.transform.Rotate(0,-90,0,Space.Self);
    magic.GetComponent<Rigidbody>().velocity = magicSpawnPoint.forward * magicSpeed;
    StartCoroutine(StartCooldown());
}

public IEnumerator StartCooldown(){
    canCast = false;
    yield return new WaitForSeconds(castCooldownInSeconds);
    canCast = true;
}
```

Isječak koda 22. Skripta PlayerMagic.cs 1.dio

U funkciji Update() koja se izvodi svaki frame se provjerava ako je pritisnuta tipka Q. Ako je pritisnuta pokreće se funkcija castAttackMagic(). Ako igrač ne može izvesti čaroliju ništa se neće desiti, a ako može stvorit će se objekt magicPrefab na poziciji magicSpawnPointa s njegovom rotacijom. Nakon toga će se zakrenuti objekt za –90 stupnjeva na y osi. Potom se izvodi ubrzanje magije kako bi se magija kretala u igrici prema naprijed. Zatim se poziva korutina StartCooldown() u kojoj se canCast postavlja na lažnu vrijednost te se nakon castCooldownInSeconds sekundi postavlja na istinitu vrijednost.

Korutine se koriste za izvršavanje koda kroz više frame-ova i mogu se koristiti za neprestano izvršavanje dijela koda dok joj se ne naredi da prestane. Korutine također sadržavaju „yield” instrukciju koja će čekati zadano vrijeme (npr. yield return new WaitForSeconds(5) će čekati 5 sekundi te će onda izvršiti kod koji se nalazi nakon te linije koda) i pokreće se tako da se koristi StartCorutine() metoda koja kao argument prima korutinu i potrebne parametre (Carillo, 2022).

```
private void castMagicCircleCycle(){
    canCastMagicCircle = false;
    castMagicCircle();
    Invoke("disableMagicCircle",1.0f);
    Invoke("castMagicCircle",1.01f);
    Invoke("disableMagicCircle",2.0f);
    Invoke("castMagicCircle",2.01f);
    Invoke("disableMagicCircle",3.0f);
    Invoke("castMagicCircle",3.01f);
    Invoke("disableMagicCircle",4.0f);
    Invoke("castMagicCircle",4.01f);
    Invoke("disableMagicCircle",5.0f);
    Invoke("enableCastingMagicCircle",30.0f);


}
private void castMagicCircle(){
    magicCircle.SetActive(true);
    if(magicCircleCounter==0){
        magicCircleVisual.SetActive(true);
    }
    magicCircleCounter++;
}
private void disableMagicCircle(){
    magicCircle.SetActive(false);
    magicCircleCounter++;

    if(magicCircleCounter == 10){
        magicCircleVisual.SetActive(false);
        magicCircleCounter = 0;
    }
}
private void enableCastingMagicCircle(){
    canCastMagicCircle = true;
}
```

Isječak koda 23. Skripta PlayerMagic.cs 2.dio

Metoda castMagicCircleCycle() pokreče ciklus izvođenja magičnog kruga. Postavlja se vrijednost canCastMagicCircle varijable na lažnu vrijednost te se poziva metoda castMagicCircle() unutar koje se objekt igre magicCircle postavlja u aktivno stanje, odnosno vidljiv je u igrici. Ako je magicCircleCounter jednak broju nula onda se postavlja i magicCircleVisual na vrijednost active, odnosno i on se prikazuje u svijetu. Potom se koristi funkcija Invoke() koja poziva neku drugu funkciju nakon određenog vremena. Nakon jedne sekunde poziva funkciju disableMagicCircle() koja postavlja objekt igre magicCircle u neaktivno stanje odnosno ne prikazuje ga u igrici. Nakon što se sve iteracije izvedu vrijednost varijable magicCircleCounter se postavlja na vrijednost nula te se nakon 30 sekundi od aktivacije polja poziva metoda enableCastingMagicCircle() koja postavlja vrijednost varijable canCastMagicCircle na istiniti vrijednost što znači da igrač ponovo može stvoriti magični krug.

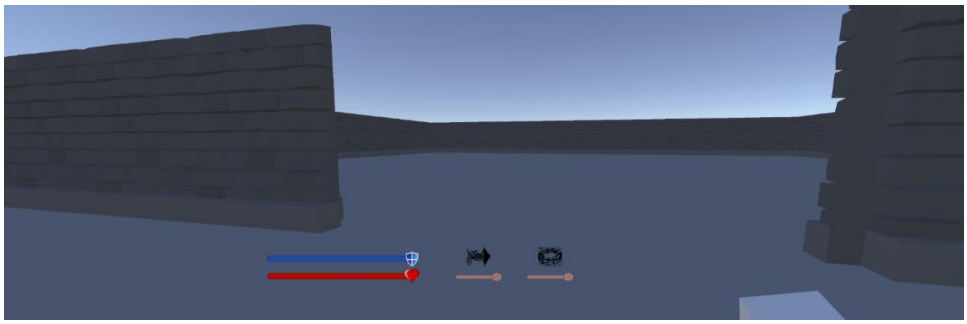
5.3 Korisničko sučelje igrice

U korisničko sučelje spada prikaz igračevih životnih bodova, panel s igračevim atributima, korištenje magije, izbornik pauze, izbornik smrti i vremenski brojač igre koji se prikazuju na ekranu. Skripte za korisničko sučelje se nalaze na glavnom platnu igre te one su odgovorne za ispravan rad UI elemenata igrice. Skripte koje se nalaze na platnu su PauseMenu.cs, StatsUI.cs, HealthUI.cs, MagicUI.cs te će one biti pobliže objašnjene u nastavku. Glavno platno s odgovarajućim skriptama je prikazano na slici 25.

Slika 25. Prikaz komponenti koje se nalaze na objektu igre platno

5.3.1 Korisničko sučelje životnih bodova

Korisničko sučelje životnih bodova se sastoji od dva klizača koji predstavljaju štit i životne bodove. Plavi klizač s ikonom štita prikazuje vrijednost igračevog štita, a crveni klizač s ikonom srca prikazuje igračeve životne bodove. Oba klizača su prikazana na slici 26.



Slika 26. Prikaz korisničkog sučelja životnih bodova

Skripta koje je odgovorna za ispravni rad klizača i prikaz štita i životnih bodova na platnu je HealthUI.cs koja se prikazuje u isječku koda 24.

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

public class HealthUI : MonoBehaviour
{
    private PlayerHealth playerHealth;
    public Slider healthSlider;
    public Slider armorSlider;
    // Start is called before the first frame update
    void Start()
    {
        playerHealth = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerHealth>();

    }

    // Update is called once per frame
    void Update()
    {
        healthSlider.maxValue = (float)playerHealth.maxHealth;
        armorSlider.maxValue = (float)playerHealth.maxPlayerArmor;
        healthSlider.value = (float)playerHealth.health;
        armorSlider.value = (float)playerHealth.playerArmor;

    }
}
```

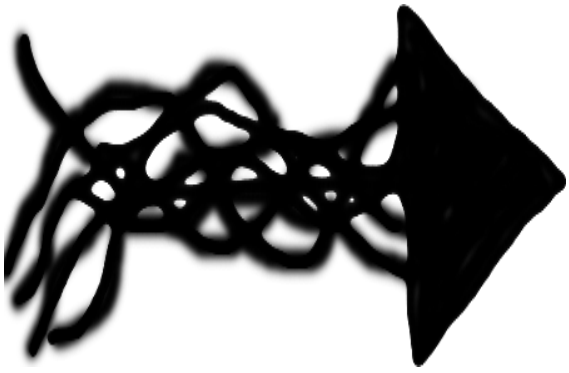
Isječak koda 24. Skripta HealthUI.cs

U skripti HealthUI.cs u Start metodi se pronalazi objekt igre koji ima oznaku „Player" te se dohvaća njegova komponenta PlayerHealth. U Update() metodi se maksimalna vrijednost healthSlider klizača postavlja na vrijednost varijable maxHealth komponente PlayerHealth. Kako bi se vrijednost mogla postaviti na klizač, maxHealth varijabla je pretvorena u float vrijednost iz cjelobrojne vrijednosti. Varijabla armorSlider.maxValue odnosno maksimalna vrijednost klizača armorSlider se postavlja na vrijednost varijable maxPlayerArmor komponente PlayerHealth te je pretvorena u tip float. Vrijednosti healthSlider.value i armorSlider.value označavaju trenutnu vrijednost igračevih životnih bodova i njegovog štita. Također se dohvaćaju iz komponente PlayerHealth i pretvaraju se u float vrijednost kako bi se njihova vrijednost mogla dodijeliti odgovarajućem klizaču.

5.3.2 Korisničko sučelje igračeve magije

Korisničko sučelje igračeve magije sastoji se od 2 klizača i dvije slike koje se nalaze iznad klizača kako bi označile koji se klizač odnosi na koju magiju. Slike 27 i 28 prikazuju ikone korištene za magiju.

Slika 27. Slika ikone magičnog polja



Slika 28. Prikaz ikone magične strijele

Obje slike su napravljene u Photopea-i mrežnom uređivaču fotografija i grafike koji se koristi za uređivanje slika, izradu ilustracija, web dizajn ili pretvaranje između različitih formata slika.

Skripta koja je odgovorna za prikaz magije na glavnom platnu je MagicUI.cs i prikazana je u isječcima koda 26 i 27.

```
public class MagicUI : MonoBehaviour
{
    public Slider magicAttackCooldown;
    public Slider magicFieldCooldown;
    private float attackMagicTimer;
    private float magicFieldTimer;
    PlayerMagic playerMagic;
    // Start is called before the first frame update
    void Start()
    {
        playerMagic = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMagic>();
        magicAttackCooldown.maxValue = (float)playerMagic.castCooldownInSeconds;
        magicFieldCooldown.maxValue = (float)playerMagic.magicCircleCooldown;
    }
    // Update is called once per frame
    void Update()
    {
        if(!(playerMagic.canCast) && attackMagicTimer>=0){
            attackMagicTimer -= Time.deltaTime;
            magicAttackCooldown.value = attackMagicTimer;
        }else{
            magicAttackCooldown.value = playerMagic.castCooldownInSeconds;
            attackMagicTimer = magicAttackCooldown.maxValue;
        }
        if(!(playerMagic.canCastMagicCircle)){
            magicFieldTimer -=Time.deltaTime;
            magicFieldCooldown.value = magicFieldTimer;
            }else{
            magicFieldCooldown.value = playerMagic.magicCircleCooldown;
            magicFieldTimer = magicFieldCooldown.maxValue;
        }
```

Isječak koda 25. Skripta MagicUI.cs 1.dio

Igrač ima dvije magije koje može koristiti, prva magija je igrač pošalje svjetlosnu strijelu prema neprijatelju, a druga magija je da igrač stvori magično polje oko sebe koje radi štetu neprijateljevim životnim bodovima. MagicUI.cs skripta se koristi kako bi se igraču prikazalo kada može koristiti koji tip magije. U Start metodi se pronalazi objekt igre s oznakom “Player” te se dohvaća njegova komponenta PlayerMagic. Potom se maksimalne vrijednosti klizača postavljaju na vrijednosti tipa float koje određuju koliko sekundi mora proći do sljedeće mogućnosti korištenja magije nakon što ju je igrač iskoristio. U Update() metodi se provjerava ako je igrač iskoristio magiju i ako je postavlja se timer (hrv. mjerač vremena) jednak vremenu koje mora proći do sljedeće mogućnosti korištenja magije koje je definirano varijablama attackMagicTimer za magičnu strijelu i magicFieldTimer za magično polje. Svaka magija ima svoj mjerač vremena i neovisne su jedna o drugoj.

```
        if(magicAttackCooldown.maxValue != (float)playerMagic.castCooldownInSeconds){
            magicAttackCooldown.maxValue = (float)playerMagic.castCooldownInSeconds;
        }
        if( magicFieldCooldown.maxValue != (float)playerMagic.magicCircleCooldown){
            magicFieldCooldown.maxValue = (float)playerMagic.magicCircleCooldown;
        }
    }
}
```

Isječak koda 26. Skripta MagicUI.cs 2.dio

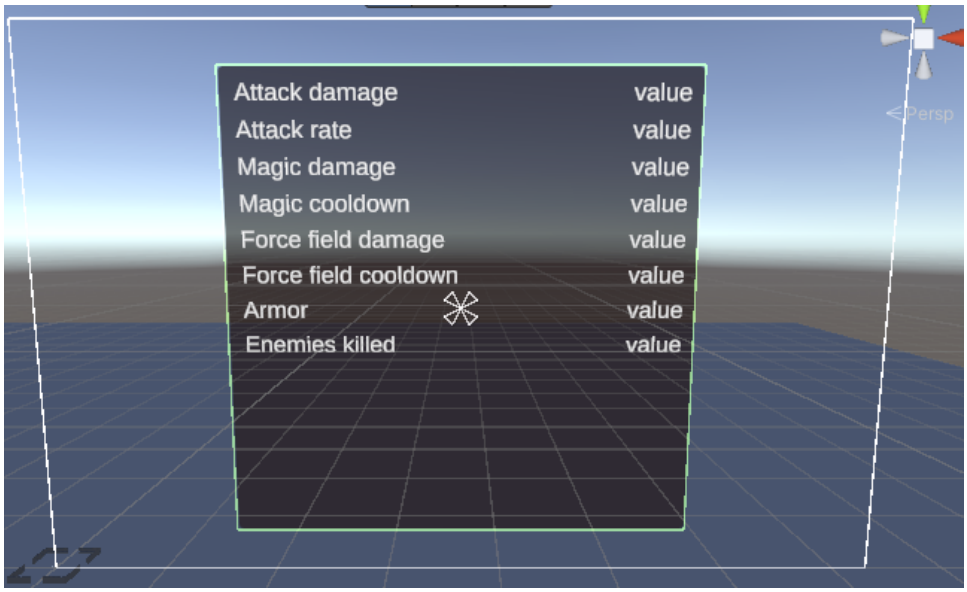
Potrebno je također provjeravati da li su se vrijednosti castCooldownInSeconds i magicCircleCooldown komponente PlayerMagic promijenile, ako jesu potrebno je postaviti maksimalnu vrijednost odgovarajućeg klizača kako klizač ne bi imao praznine u sebi.

5.3.3 Korisničko sučelje igračevih atributa

Igrač ima velik broj atributa te kada bi se ti atributi cijelo vrijeme prikazivali na ekranu, prostor bi bio pretrpan. Iz tog razloga su atributi odvojeni u novi panel kako bi im igrač mogao brzo pristupiti te ih lako pregledati. Slika 29 prikazuje igračeve atribute.

Slika 29. Prikaz igračevih atributa

Slika iznad prikaz StatsUI panela u igrici. Tekst s lijeve strane je uvijek isti, ali se tekst s desne strane mijenja putem skripte. Slika 30 prikazuje igračeve atribute prije pokretanja igre.



Slika 30. Prikaz igračevih atributa prije pokretanja igrice

U scenskom prikazu je moguće vidjeti da je lijeva strana identična onoj u igrici, ali desna strana ima vrijednost „value”. Sve i jedna vrijednost value će biti zamijenjena nekom drugom vrijednosti putem skripte. U nastavku će biti objašnjena skripta StatsUI.cs koja je prikazana u isječcima koda 27 i 28.

```
public class StatsUI : MonoBehaviour
{
    public GameObject player;
    private PlayerMagic playerMagic;
    public MagicCircle magicCircle;
    private PlayerAttack playerAttack;
    private  PlayerHealth PlayerHealth;
    public GameObject[] statsArray = new GameObject[8];
    private TMP_Text temp;
    public GameObject statsUI;
    private EnemySpawnAndKillCount esakc;
    float tempFloatValue;
    int tempIntValue;
    private bool statsUIisActive = false;

    // Start is called before the first frame update
    void Start()
    {
        playerAttack = player.GetComponent<PlayerAttack>();
        PlayerHealth = player.GetComponent<PlayerHealth>();
        playerMagic = player.GetComponent<PlayerMagic>();
        esakc = player.GetComponent<EnemySpawnAndKillCount>();
    }

    // Update is called once per frame
    void Update(){
        if(Input.GetKeyDown(KeyCode.I)){
                if(!statsUIisActive){
                    UpdateUIValues();
                    statsUI.SetActive(true);
                    statsUIisActive=true;
                }else{
                    statsUI.SetActive(false);
                    statsUIisActive=false;
                }
        }
    }
```

Isječak koda 27. Skripta StatsUI.cs 1.dio

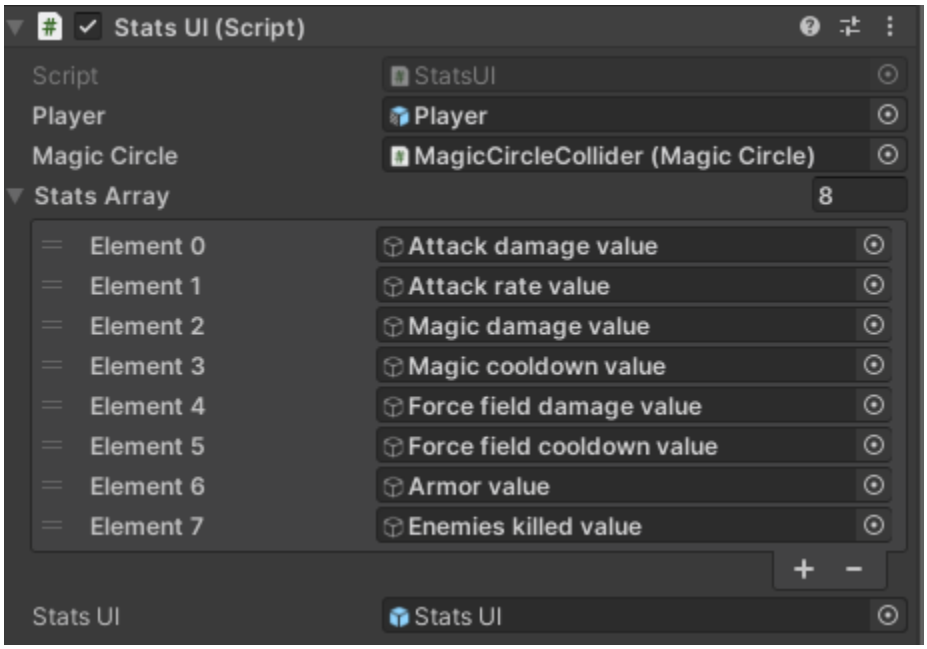
Unutar Update() metode provjerava se da li je igrač pritisnuo tipku I. Ako je tipka pritisnuta i vrijednost statsUIIsActive je postavljena na lažnu vrijednost odnosno StatsUI nije aktivan, poziva se metoda UpdateUIValues(), statsUI se postavlja u aktivno stanje odnosno moguće ga je vidjeti unutar igrice te statsUIisActive se postavlja na istinitu vrijednost kako bi se znalo da je UI aktivan. Ako je pritisnuta tipka I, a vrijednost statsUIIsACtive je istinite vrijednosti onda se statsUI.SetActive postavlja na lažnu vrijednost odnosno više nije vidljiv unutar igre i vrijednost statsUIisActive se postavlja na lažnu kako bi idućim pritiskom tipke I bilo moguće ponovo otvoriti panel s informacijama o igraču.

```
    private void UpdateUIValues(){
        tempIntValue = playerAttack.meeleDamage;
        statsArray[0].GetComponent<TMP_Text>().text = tempIntValue.ToString();
        tempFloatValue = 1.0f /playerAttack.timeToAttack;
        statsArray[1].GetComponent<TMP_Text>().text = tempFloatValue.ToString();
        tempIntValue = playerAttack.magicDamage;
        statsArray[2].GetComponent<TMP_Text>().text = tempIntValue.ToString();
        tempFloatValue = playerMagic.castCooldownInSeconds;
        statsArray[3].GetComponent<TMP_Text>().text = tempFloatValue.ToString();
        tempIntValue = magicCircle.magicCircleDamage;
        statsArray[4].GetComponent<TMP_Text>().text = tempIntValue.ToString();
        tempFloatValue = playerMagic.magicCircleCooldown;
        statsArray[5].GetComponent<TMP_Text>().text = tempFloatValue.ToString();

        tempIntValue = PlayerHealth.playerArmor;
        statsArray[6].GetComponent<TMP_Text>().text = tempIntValue.ToString();

        tempIntValue = esakc.enemyKillCount;
        statsArray[7].GetComponent<TMP_Text>().text = tempIntValue.ToString();

    }
}
```

Isječak koda 28. Skripta StatsUI.cs 2.dio

Metoda UpdateUIValues() prolazi kroz ćelije polja statsArray te ih postavlja na određene vrijednosti. Sve vrijednosti su tekstualne te im se mijenja vrijednost u prethodno definiranu vrijednost. Prva vrijednost će dohvatiti štetu koju igrač može načiniti jednim udarcem te će je postaviti kao tekst prve ćelije polja tako što tu vrijednost pretvorimo u string. U drugoj ćeliji je vrijednost „attack rate” odnosno koliko igrač puta može izvesti napad u sekundi. Treći element će imati vrijednost štete koju igrač može napraviti magijom. Četvrti element polja će biti vrijednost koliko igrač sekundi mora sačekati da bi se magija opet mogla iskoristiti. U petoj ćeliji polja će imati vrijednost štete koju igrač radi magičnim poljem svake sekunde. U šestoj ćeliji polja je vrijednost sekundi koje igrač mora sačekati kako bi opet

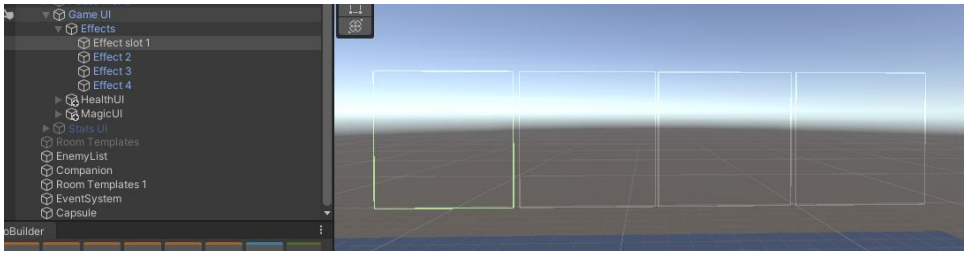
mogao pokrenuti magično polje. Sedmi element polja sadrži vrijednost njegovog štita te osmi element polja sadrži vrijednost koliko je igrač ubio neprijatelja. Slika 31 prikazuje dodijeljene objekte igre StatsUI.cs skripti u inspektoru.



Slika 31. Prikaz dodijeljenih objekata StatsUI skripti

5.3.4 Korisničko sučelje efekata nad igračem

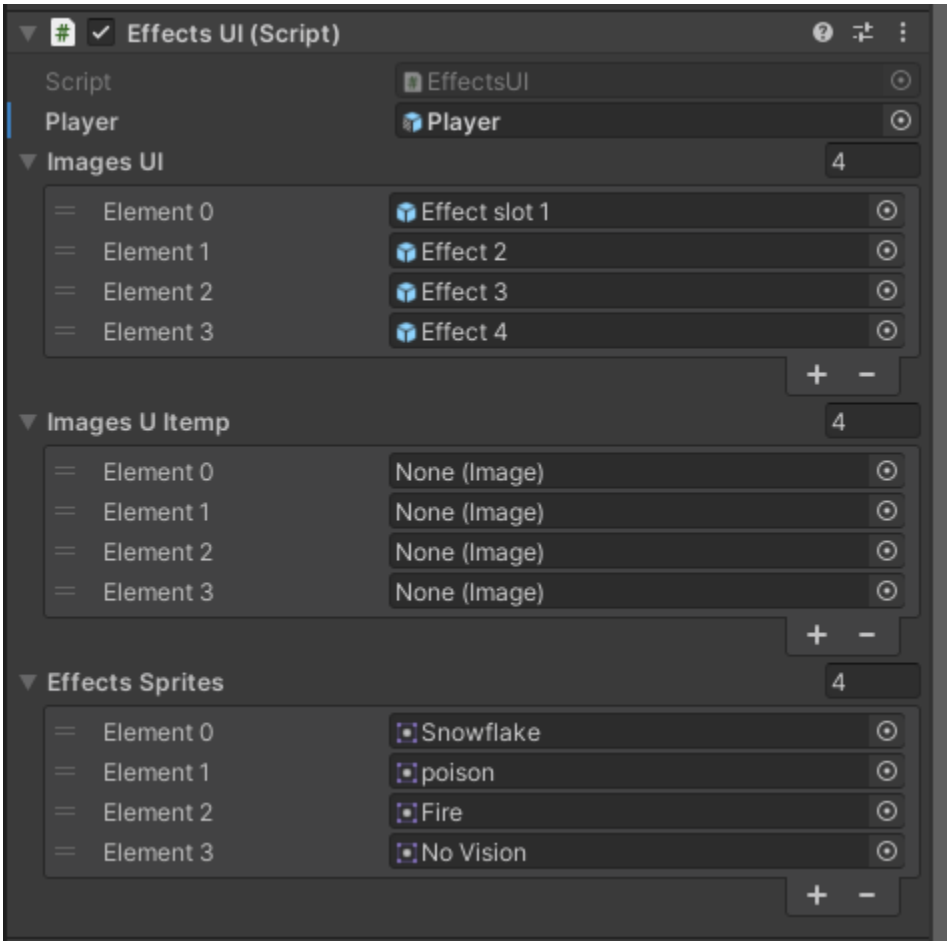
Korisničko sučelje efekata nad igračem prikazuje efekte pod čijim utjecajem je igrač. Efekti mogu biti otrov (eng. poison), opekline (eng. burn), smetnja vida (eng. visual impairment) te smrzavanje (eng. freeze). Slika 32 prikazuje red u koji se postavljaju slike kada je igrač pod nekim od efekata.



Slika 32. Red slika koje se koriste za prikaz efekata nad igračem

Korisničko sučelje efekata na igračem je napravljeno na način da su postavljena četiri UI elementa koji su tipa slika (eng. image) u koje se postavlja slika kada je neki od efekata

nad igračem aktivan. Slika 33 prikazuje skriptu EffectsUI.cs u inspektoru te slike i objekte igre koji su joj predani.



Slika 33. Slike i objekti igre predani skripti EffectsUI.cs

ImagesUI sadrži četiri UI elementa slike koji se kasnije koriste za postavljanje slike efekta. Effects Sprites su spriteovi koji se postavljaju u ImagesUI elemente. Slika 34 prikazuje efekt nad igračem unutar igre.

Slika 34. Prikaz efekta nad igračem unutar igre

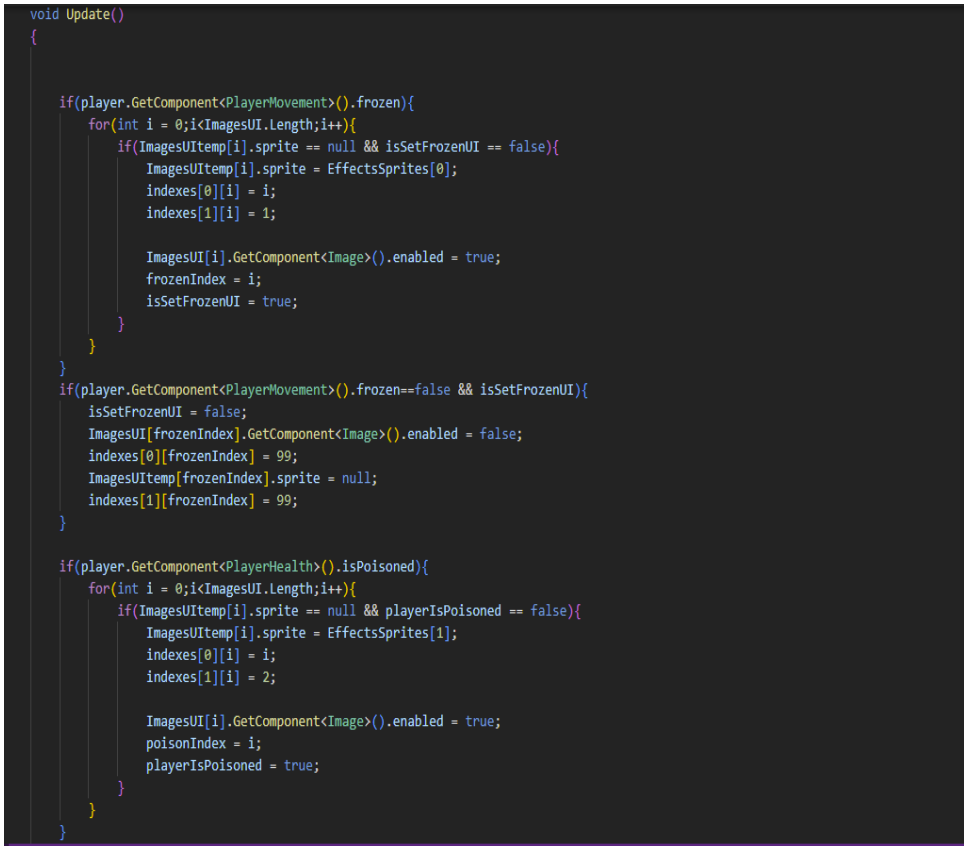
EffectsUI.cs skritpa će biti objašnjena u nastavku te će biti prikazana u isječcima koda 29, 30 i 31.

```
public class EffectsUI : MonoBehaviour
{
    public GameObject player;

    // public List<GameObject> ImagesUI = new List<GameObject>();
    public GameObject[] ImagesUI;
    public Image[] ImagesUItemp;
    public Sprite[] EffectsSprites;
    public int[][] indexes = new int[2][];


    int frozenIndex,poisonIndex,burnIndex,visionIndex;
    bool isSetFrozenUI,playerIsPoisoned,playerIsBurning,visionIsAffected = false;
    // Start is called before the first frame update
    void Start()
    {
        indexes[0] = new int[4];
        indexes[1] = new int[4];

        for(int i=0;i<2;i++){
            for(int j=0;j<4;j++){
                indexes[i][j]=99;
            }
        }

        indexes[0][0]=99;
        for(int i = 0;i<ImagesUI.Length;i++){
            ImagesUItemp[i] = ImagesUI[i].GetComponent<Image>();
        }
    }
```

Isječak koda 29. Skripta EffectsUI.cs 1.dio

Public GameObject[] ImagesUI je polje podataka tipa GameObject te ti podatci su zapravo prethodno navedeni UI elementi tipa Image. Image[] ImagesUItemp je pomoćno polje podataka koji se koristi za dohvaćanje komponente slike svakog objekta igre polja ImagesUI. Public Sprite[] je polje spriteova koji se koriste za postavljanje slike na platnu.

Public int[][] indexes je dvodimenzionalno polje, od 2 reda i 4 stupca. Prvo se definiraju 2 reda, a zatim u Start() metodi se postavljaju stupci. Cjelobrojne vrijednosti frozenIndex, poisonIndex, burnIndex i visionIndex se koriste za pamćenje indexa na kojem se nalazi neki od aktivnih efekata, odnosno efekt koji ima sliku. Bool vrijednosti isSetFrozenUI, playerIsPoisoned, playerIsBurning, visionIsAffected su postavljene na lažnu vrijednost. U Start() metodi se svi indeksi postavljaju na broj 99 zato što int tip polja ne može imati vrijednost null pa je iz tog razloga odabrana vrijednost koja će biti vrijednost provjere. Potom se prolazi petljom dužine polja ImagesUI[] te se postavlja komponenta slike kao ćelija niza ImagesUITemp[i] gdje je i broj prolaska kroz petlju.



```
void Update()
{


    if(player.GetComponent<PlayerMovement>().frozen){
        for(int i = 0;i<ImagesUI.Length;i++){
            if(ImagesUItemp[i].sprite == null && isSetFrozenUI == false){
                ImagesUItemp[i].sprite = EffectsSprites[0];
                indexes[0][i] = i;
                indexes[1][i] = 1;

                ImagesUI[i].GetComponent<Image>().enabled = true;
                frozenIndex = i;
                isSetFrozenUI = true;
            }
        }
    }
    if(player.GetComponent<PlayerMovement>().frozen==false && isSetFrozenUI){
        isSetFrozenUI = false;
        ImagesUI[frozenIndex].GetComponent<Image>().enabled = false;
        indexes[0][frozenIndex] = 99;
        ImagesUItemp[frozenIndex].sprite = null;
        indexes[1][frozenIndex] = 99;
    }

    if(player.GetComponent<PlayerHealth>().isPoisoned){
        for(int i = 0;i<ImagesUI.Length;i++){
            if(ImagesUItemp[i].sprite == null && playerIsPoisoned == false){
                ImagesUItemp[i].sprite = EffectsSprites[1];
                indexes[0][i] = i;
                indexes[1][i] = 2;

                ImagesUI[i].GetComponent<Image>().enabled = true;
                poisonIndex = i;
                playerIsPoisoned = true;
            }
        }
    }
```

Isječak koda 30. Skripta EffectsUI.cs 2.dio

Unutar Update() metodi se provjeravaju učinci neprijatelja na igrača. Ako je igrač smrznut provjerava se postoji li sprite u komponenti slike te da li je kontrolna vrijednost isSetFrozenUI varijable jednaka laži. Ako se komponenti slike postavlja sprite, onda se spremaju indeksi mjesta na kojem se nalaze te tipa efekta. ImagesUI[i] slika se postavlja na omogućeno (eng. enabled) te se pamti frozenIndex odnosno indeks i. Također se

IsSetFrozenUI postavlja na istinitu vrijednost. Ako igrač nije smrznut, a bio je smrznut potrebno je maknu sliku efekta i postaviti vrijednost IsSetFrozenUI varijable na lažnu vrijednost. Komponenta slike ImagesUI[frozenIndex] se postavlja na lažnu vrijednost te se indeksi postavljaju na kontrolnu vrijednost 99. Također vrijednost sprite-a se postavlja na vrijednost null. Isti kod vrijedi za sve ostale efekte, samo što se provjeravaju druge vrijednosti ovisno o tipu efekta.

```
private void UpdateUI(){
    int indexValue;
    int typeValue;

    for(int i=0;i<3;i++){
        if(indexes[0][i]!=99){
            Debug.Log("Index [0]["+i+"]is equal to"+indexes[0][i]);

        }
        for(int j=0;j<3;j++){
            if(indexes[0][j]==99 && indexes[0][j+1]!=99){
                indexValue = indexes[0][j+1];
                typeValue= indexes[1][j+1];
                indexes[0][j+1] = 99;
                indexes[1][j+1] = 99;
                indexes[0][j] = indexValue;
                indexes[1][j] = typeValue;
                ImagesUItemp[j].sprite = EffectsSprites[0];
                ImagesUI[j].GetComponent<Image>().enabled = true;
                ImagesUI[j+1].GetComponent<Image>().enabled = false;
                if(typeValue==1){
                    ImagesUItemp[j].sprite = EffectsSprites[0];
                    frozenIndex=j;
                }
                if(typeValue==2){
                    ImagesUItemp[j].sprite = EffectsSprites[1];
                    poisonIndex = j;
                }
                if(typeValue==3){
                    ImagesUItemp[j].sprite = EffectsSprites[2];
                    burnIndex = j;
                }
                if(typeValue==4){
                    ImagesUItemp[j].sprite = EffectsSprites[3];
                    visionIndex = j;
                }
```

Isječak koda 31. Skripta EffectsUI.cs 3.dio

UpdateUI funkcija provjerava nalazi li se vrijednost 99 u prvom redu 2D arraya. Ako se nalazi vrijednost 99 u indexes[0][j] ćeliji i ako se nalazi vrijednost koja nije 99 u indexes[0][j+1] ćeliji. Ako se nalazi vrijednost potrebno ih je zamijeniti, vrijednost iz j+1 ćeliji postaviti u vrijednost j ćelije. Za to nam služe pomoćne varijable koje pamte indeks i tip efekta. Komponenta Image od ImagesUI[j] se postavlja na istinitu vrijednost, a slika u ćeliji ImagesUI[0][j+1] se postavlja na false. Potom se provjerava tip efekta kako bi se stavio pripadajući sprite te postavila odgovarajuća vrijednost indeksa u ImagesUI[0][j].

5.3.5 Korisničko sučelje neprijateljevih životnih bodova

Kako bi igrač znao koliko neprijatelj ima životnih bodova, potrebno je napraviti sučelje životnih bodova. Životni bodovi neprijatelja se prikazuju putem skripte EnemyHealthSlider.cs koja je prikazana u isječku koda 32.

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


public class EnemyHealthSlider : MonoBehaviour
{

    public Enemy enemy;
    public Slider enemyHealth;
    private Camera camera;
    public Canvas canvas;

    // Start is called before the first frame update
    void Start()
    {
        enemyHealth.maxValue = enemy.health;
        camera = GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>();
        canvas.worldCamera = camera;

    }

    // Update is called once per frame
    void Update()
    {
        enemyHealth.value = (float)enemy.health;
    }
}
```

Isječak koda 32. Skripta EnemyHealthSlider.cs

Public Enemy enemy (hrv. neprijatelj) je varijabla koja se odnosi na komponentu Enemy objekta igre neprijatelj. Enemy komponenta je skripta Enemy.cs. Klizač enemyHealth je UI element klizač kojem će se smanjivati vrijednost kada neprijatelj primi štetu svojim životnim bodovima. Varijabla camera se odnosi na kameru te je u Start() metodi ta vrijednost postavljena na kameru koja ima oznaku „Main Camera“. U Update() funkciji se vrijednost klizača postavlja na vrijednost varijable health iz skripte Enemy.cs odnosno na trenutne životne bodove neprijatelja. Slika 35 prikazuje platno s neprijateljevim životnim bodovima.

Slika 35. Prikaz neprijateljevih životnih bodova

S obzirom na to da se neprijatelji stvaraju nakon što se stvori mapa i to ulaskom u prostoriju nije moguće stvoriti neprijatelju UI za životne bodove već svaki neprijatelj mora imati svoje platno kojem je način renderiranja postavljen na prostor svijeta te mu je dodijeljena kamera igrača preko prije navedene skripte. Razlog tome je što neprijatelji i igrač će se pomicati po mapi, a ako način renderiranja (eng. render Mode) ne bi bio postavljen na prostor svijeta onda se neprijateljevo platno ne bi rotiralo s neprijateljem već bi njegova vrijednost bila statična negdje na ekranu.

5.3.6 Izbornik pauze

Izbornik pauze se unutar igre prikazuje kada igrač pritisne tipku escape. Nakon što se izbornik pauze prikaže igrač može pritiskom na „nastavi“ (eng. resume) nastaviti igru, pritiskom na „postavke“ gumb otvoriti postavke koje su identične onima u glavnom izborniku osim što je dodan klizač s osjetljivosti kursora kako bi igrač mogao ubrzati ili usporiti kretanje kursora. Slika 36 prikazuje izbornik pauze unutar igre.

Slika 36.Prikaz izbornika pauze unutar igre

Pritiskom na gumb „glavni izbornik“ igrač odlazi u glavni izbornik igre te napušta trenutnu igru odnosno predaje je. Pritiskom na gumb za izlaz iz igrice igrač prekida rad aplikacije. U nastavku će biti objašnjen kod izbornika pauze. Kod skripte PauseMenu.cs će biti prikazan u isječcima koda 33 i 34.

```
public class PauseMenu : MonoBehaviour
{

    public bool gameIsPaused=false;
    public bool gameCanBePaused = true;
    public GameObject pauseMenuUI;
    public GameObject player;
    private DestroyList dl;
    private RunStats runStats;
    private PlayerHealth PlayerHealth;

    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        PlayerHealth = player.GetComponent<PlayerHealth>();
        dl = GameObject.FindGameObjectWithTag("ItemsToDestroy").GetComponent<DestroyList>();
        runStats = GameObject.FindGameObjectWithTag("RunStatsSaver").GetComponent<RunStats>();

    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape)){
            if(gameIsPaused){
                Resume();
            }else{
                if(gameCanBePaused){
                    Pause();
                }
            }
        }
    }
```

Isječak koda 33. PauseMenu.cs 1.dio

U Start() funkciji se dohvaćaju potrebni objekti igre odnosno njihove komponente. U Update() funkciji se provjerava ako je igrač pritisnuo tipku escape te ako je igra pauzirana. Ako je igra pauzirana i igrač pritisne tipku Escape igra se nastavlja, a ako igra nije pauzirana i igrač pritisne tipku Escape igra se pauzira.

```
    private void Pause(){
        Time.timeScale = 0f;
        pauseMenuUI.SetActive(true);
        gameIsPaused = true;
        Cursor.lockState = CursorLockMode.None;
    }

    public void Resume(){
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        gameIsPaused = false;
        Cursor.lockState = CursorLockMode.Locked;

    }


    public void Quit(){
        if(PlayerHealth.health>0){
            runStats.QuitARun();
        }
        Application.Quit();

    }

    public void LoadMainMenu(){
        if(PlayerHealth.health>0){
            runStats.QuitARun();
        }
        dl.DestroyerToMenu();
        Time.timeScale = 1f;
        gameIsPaused = false;
        SceneManager.LoadScene("MainMenu");
        DestroyGameObject();
    }

     void DestroyGameObject(){
        Destroy(player);
    }
}
```

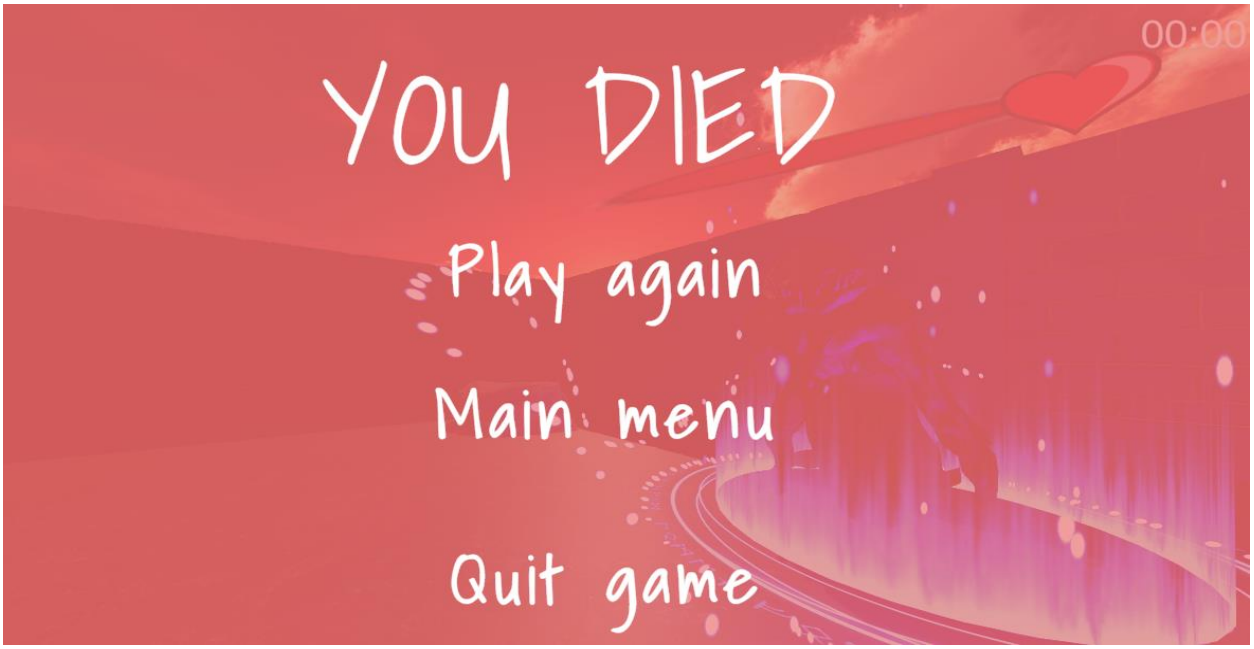
Isječak koda 34. Skripta PauseMenu.cs 2.dio

U Pause() funkciji se otključava kursor i Time.timeScale se postavlja na nulu, odnosno zaustavlja se vrijeme u igrici. U Resume() funkciji se zaključava kursor te se Time.timeScale postavlja na vrijednost odnosno igra se nastavlja normalnom brzinom.

U funkciji Quit() se vrši provjera igračevih životnih bodova te ako su oni veći od nule aplikacija prestaje s radom, a ako nisu potrebno je uvećati broj izgubljenih igara za jedan te potom prestati s radom aplikacije. Unutar funkcije LoadMainMenu() potrebno je uništiti sve objekte koji na sebi imaju skriptu DontDestroyOnLoad.cs kako ne bi bilo duplikata unutar glavne scene. Također je potrebno postaviti Time.timeScale na vrijednost jedan jer odlaskom u glavni izbornik igra više nije pauzirana.

5.3.7 Izbornik smrti

Izbornik smrti se prikazuje nakon što igračevi životni bodovi postanu jednaki ili manji od broja nula. Slika 38 prikazuje izbornik smrti u igrici.



Slika 37. Prikaz izbornika smrti u igrici

Igrač ima tri opcije, a to su pokretanje igrice iz početka, odlazak u glavni izbornik te prekidanje rada aplikacije. Isječak koda 35 prikazuje kod skripte DeathScreen.cs.

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


public class DeathScreen : MonoBehaviour
{
    private DestroyList dl;
    private RunStats runStats;

    void Start(){
    dl = GameObject.FindGameObjectWithTag("ItemsToDestroy").GetComponent<DestroyList>();
    runStats = GameObject.FindGameObjectWithTag("RunStatsSaver").GetComponent<RunStats>();
    }


    PauseMenu pauseMenu;
    public void RestartGame(){
        dl.DestroyAndReset();
        runStats.RestartARun();
        SceneManager.LoadScene(2);
        Time.timeScale = 1f;
    }

}
```

Isječak koda 35. Skripta DeathScreen.cs

Odlazak u glavni izbornik i prekid rada aplikacije su isti kao i kod izbornika pauze. Ponovno pokretanje igrice se vrši kada igrač umre i pritisne gumb „igraj ponovo“ (eng. play again). Izvršava se funkcija RestartGame() u kojoj je potrebno uništiti odgovarajuće objekte za ispravan rad igre. Potom se uvećava broj pokrenutih igri za jedan jer igrač započinje novu igru s istim pratiocem kao prije, ukoliko želi odabrati novog pratioca potrebno je otići u glavni izbornik. Također se preskače uvodna scena te se igrač odmah stvara u labirintu.

5.4 Svijet unutar igre

Svijet unutar igre se sastoji od terena koji je proceduralno stvoren, neprijatelja, zamki koje igrač i neprijatelji mogu aktivirati, pratioca koji pomaže igraču, stvari koje igrač može pokupiti te kristala koje igrač može aktivirati.

5.4.1 Proceduralno stvaranje terena

Proceduralno stvaranje terena kreće iz ulazne prostorije koja ima četiri ulaza odnosno izlaza. Na svakom izlazu je na istoj udaljenosti postavljen prazan objekt igre na kojem se

nalazi sudarač u obliku kutije koji je postavljen kao okidač te je on odgovoran za stvaranje prostorije. Detaljnije će biti objašnjeno u nastavku ovog poglavlja rada.

5.4.1.1 Skripte proceduralnog stvaranja terena

Proceduralno stvaranje terena se sastoji od nekoliko skripti, a to su RoomSpawner.cs, RoomTemplates.cs, AddRoom.cs i Destroyer.cs. Prvo će biti objašnjena skripta RoomSpawner.cs čiji kod je prikazan u isječcima koda 36 i 37.

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

public class RoomSpawner : MonoBehaviour
{
    public int OpeningDirection;
    private RoomTemplates templates;
    private int rand;
    public bool spawned = false;

    void Start(){
        templates=GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();
        Invoke("Spawn",0.1f);
    }
    void Spawn()
    {
       if(spawned==false){
        if(OpeningDirection==1){
            //OpeningDirection 1 znaci da trebamo sobu sa vratima prema dolje
            rand=Random.Range(0,templates.bottomRooms.Length);
            Instantiate(templates.bottomRooms[rand],transform.position, templates.bottomRooms[rand].transform.rotation);
            }else if(OpeningDirection==2){
            //OpeningDirection 2 znaci da trebamo sobu sa vratima prema lijevo
            rand=Random.Range(0,templates.leftRooms.Length);
            Instantiate(templates.leftRooms[rand],transform.position, templates.leftRooms[rand].transform.rotation);

            }else if(OpeningDirection==3){
            //OpeningDirection 3 znaci da trebamo sobu sa vratima prema gore
            rand=Random.Range(0,templates.topRooms.Length);

            Instantiate(templates.topRooms[rand],transform.position, templates.topRooms[rand].transform.rotation);

            }else if(OpeningDirection==4){
            //OpeningDirection 4 znaci da trebamo sobu sa vratima prema desno

            rand=Random.Range(0,templates.rightRooms.Length);
            Instantiate(templates.rightRooms[rand],transform.position, templates.rightRooms[rand].transform.rotation);
```

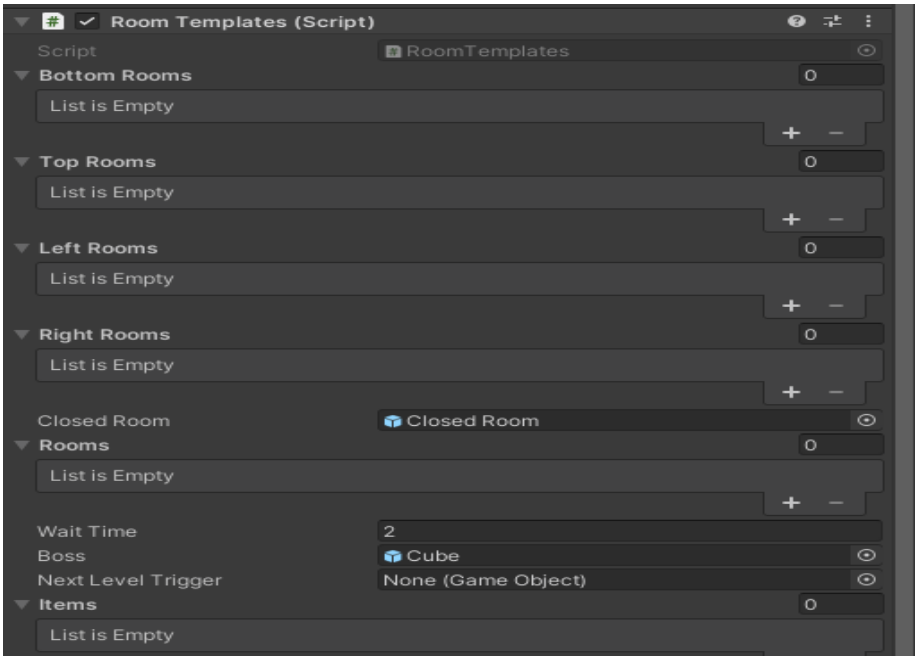
Isječak koda 36. Skripta RoomSpawner.cs 1.dio

U skripti RoomSpawner.cs se izvršava kod koji stvara teren igrice. U Start() metodi se pronalazi objekt koji ima oznaku Rooms te se dohvaća njegova komponenta RoomTemplates. Svaka točka stvaranja u prostoriji ima smjer stvaranja koji određuje koja će se prostorija stvoriti. Kada se prostorija stvori, nakon određenog vremena će se pozvati metoda Spawn(). Ako je vrijednost spawned jednaka lažnoj vrijednosti onda će se stvoriti određena prostorija ovisno o varijabli OpeningDirection (hrv. smjer otvaranja) te će se varijabla spawned postaviti na istinitu vrijednost.

```
    void OnTriggerEnter(Collider other){
        if(other.CompareTag("SpawnPoint")){
            if(other.GetComponent<RoomSpawner>().spawned == false && spawned == false){
                Instantiate(templates.closedRoom, transform.position,Quaternion.identity);
                Destroy(gameObject);
            }
            spawned=true;
        }
    }
}
```

Isječak koda 37. Skripta RoomSpawner.cs 2. dio

Ako se točka stvaranja sudarila s drugom točkom stvaranja i vrijednost spawned obje točke stvaranja su lažne vrijednosti stvorit će se zatvorena prostorija. Slika 38 prikazuje RoomTemplates.cs skriptu u inspektoru.



Slika 38. Prikaz objekta s RoomTemplates.cs komponentom u inspektoru

RoomTemplates.cs sadrži četiri liste od kojih svaka predstavlja tip sobe. Bottom Rooms su prostorije koje imaju otvor prema gore, Top Rooms su prostorije koje imaju otvor prema dolje. Left Rooms su prostorije koje imaju otvor u desno. Right Rooms su prostorije koje imaju otvor u lijevo. Closed Room je prazna prostorija koja nema otvor već se koristi u iznimnom slučaju kada dvije prostorije trebaju stvoriti prostoriju na istom mjestu te se onda stvara prazna prostorija. Rooms lista je lista u koju se dodaju sve stvorene prostorije kako

bi se mogao stvoriti šef, okidač za odlazak na idući nivo te stvari koji pomažu igraču tijekom igre. Isječci koda 38 i 39 prikazuju kod skripte RoomTemplates.cs.

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

public class RoomTemplates : MonoBehaviour
{
    public GameObject[] bottomRooms;
    public GameObject[] topRooms;
    public GameObject[] leftRooms;
    public GameObject[] rightRooms;
    public GameObject  closedRoom;
    public List<GameObject> rooms;
    public List<GameObject> items;
    public ItemList itemList;
    public float waitTime;
    private bool spawnedBoss = false;
    public GameObject boss;
    public GameObject nextLevelTrigger;
    bool itemSpawned = false;
    private int rand;
    int randomRoom;
    EnemySpawner enemySpawner;

    void Start(){
        itemList = GameObject.FindGameObjectWithTag("ItemList").GetComponent<ItemList>();
        items = itemList.items;
    }
```

Isječak koda 38. Skripta RoomTemplates.cs 1.dio

RoomTemplates.cs je skripta koja se koristi za postavljanje prostorija u četiri polja od kojih se svako polje odnosi na odgovarajući tip prostorije, odnosno bottomRooms su prostorije koje imaju vrata prema gore, topRooms su prostorije koje imaju vrata prema dolje, leftRooms su prostorije s vratima prema desno i rightRooms su prostorije koje imaju vrata prema lijevo. Javni objekt closedRoom je prostorija koja se stvara kada više prostorija treba stvoriti prostoriju na istom mjestu pa se onda stvara prostorija koja nema svrhu osim stvaranja zida između te dvije prostorije. Javna lista objekata igre rooms sadrži svaku prostoriju koja je stvorena. Javna varijabla waitTime (hrv. vrijeme čekanja) se koristi za stvaranje odgode pri odabiru prostorije u kojoj će se nalaziti glavni neprijatelj nivoa i prostorije u kojoj će se nalaziti stvar koju igrač može pokupiti. Javni objekt igre šef je glavni neprijatelj nivoa, a nextLevelTrigger je pokretač novog nivoa. Lista objekata igre stvari sadrži stvari koje igrač može pokupiti tijekom igre te varijabla itemSpawned označava ako je stvorena stvar koju igrač može pokupiti.

```
    void Update(){

        if(waitTime<=0){
            if(spawnedBoss==false){
                for(int i=0;i<rooms.Count;i++){
                    if(i==rooms.Count-1){
                        enemySpawner = rooms[i].GetComponentInChildren<EnemySpawner>();
                        enemySpawner.setBoss();
                    }
                }
            }
            if(itemSpawned == false){
                int rC = rooms.Count - 2;
                rand=Random.Range(0,rC);
                enemySpawner = rooms[rand].GetComponentInChildren<EnemySpawner>();
                int randomItem = Random.Range(0,items.Count);
                Vector3 newPos = new Vector3(rooms[rand].transform.position.x,2,rooms[rand].transform.position.z);
                Instantiate(items[randomItem],newPos,items[randomItem].GetComponent<Transform>().rotation);
                items.RemoveAt(randomItem);
                itemSpawned=true;
                enemySpawner.setItemRoom();
            }
        }
        else{
            waitTime -= Time.deltaTime;
        }
    }

}
```

Isječak koda 39. Skripta RoomTemplates.cs 2.dio

U Update() metodi se provjerava ako je vrijednost varijable waitTime manja ili jednaka nuli te ako nije vrijednost će se umanjiti za Time.deltaTime. Time.deltaTIme je vrijeme koje je bilo potrebno da se izvrši posljednji frame. Odnosno ovo svojstvo je vrijeme između trenutnog i prethodnog frame-a (Booth, 2022).

Ako je vrijednost varijable waitTime manje ili jednako broju nula provjerava se ako je vrijednost varijable spawnedBoss lažna te ako je potrebno je stvoriti glavnog neprijatelja nivoa i portal koji vodi igrača na sljedeću razinu. Prolazi se kroz listu prostorija te se pronalazi zadnja prostorija i u njoj se stvaraju te se vrijednost varijable spawnedBoss postavlja na istinitu vrijednost.

Potom se provjerava da li je vrijednost itemSpawned jednaka laži te ako je stvara se stvar koju igrač može pokupiti u jednoj od prostorija osim u prostorije u kojoj se nalazi glavni neprijatelj. Također se vrijednost varijable itemSpawned postavlja na istinitu vrijednost jer je stvar koju igrač može pokupiti stvorena. Kako bi se znalo koje su prostorije stvorene koristi se skripta AddRoom.cs čiji je kod prikazan u isječku koda 40.

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

public class AddRoom : MonoBehaviour
{
    private RoomTemplates templates;

    void Start(){
        templates=GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();
        templates.rooms.Add(this.gameObject);
    }
}
```

Isječak koda 40. Skripta AddRoom.cs

Pri stvaranju prostorije pokreće se metoda Start() u kojoj se dohvaća GameObject s tagom Room koji ima komponentu RoomTemplates. U komponentu RoomTemplates u listu Rooms se dodaje ta prostorija.

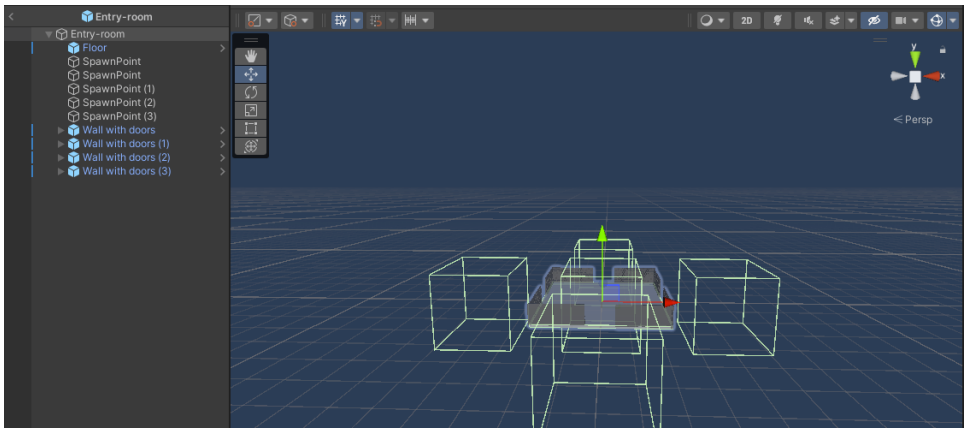
Kako bi se izbjeglo dupliciranje mape nekoliko puta potrebno je koristiti Destroyer.cs skriptu koja je prikazana u isječku koda 41. Ova skripta se nalazi na SpawnPoint objektu ulazne prostorije te ako se sudari sa SpawnPointom druge prostorije uništit će taj SpawnPoint te se tako se neće stvoriti još jedna prostorija na mjestu već stvorene prostorije odnosno polazne prostorije.

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

public class Destroyer : MonoBehaviour
{
    void OnTriggerEnter(Collider other){
        if(other.CompareTag("SpawnPoint")){
            Destroy(other.gameObject);
            }
    }
}
```

Isječak koda 41. Skripta Destroyer.cs

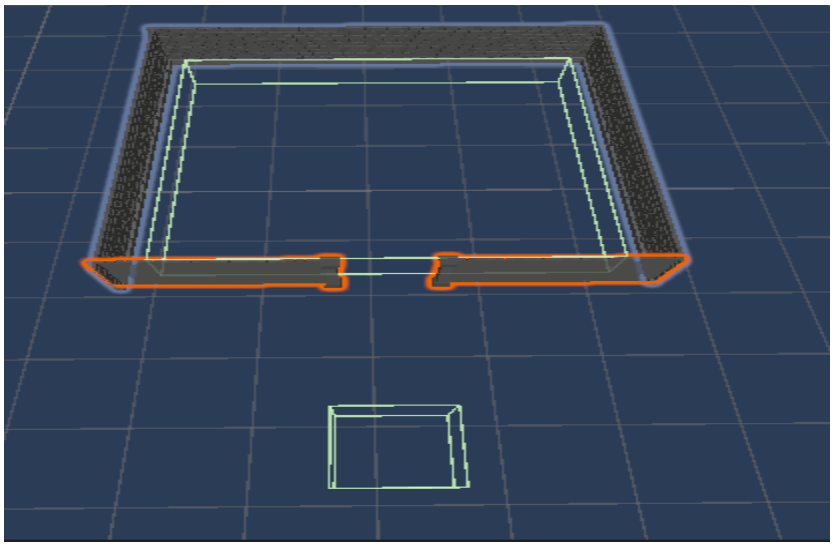
5.4.1.2 Prostorije za proceduralno stvaranje

Polazna prostorija (eng. entry room) je početna prostorija svakog nivoa. Ta prostorija ima 4 izlaza odnosno ulaza. Pri pokretanju nivoa iz polazne prostorije će se stvoriti cijela mapa, odnosno to je početna točka. Za svaki od otvora se može stvoriti jedan od više tipova prostorija koje imaju (npr. Otvor s brojem 2 može stvoriti svaki tip L prostorije, odnosno prostorije koja dolazi s desne strane). Slika 39 prikazuje početnu prostoriju.



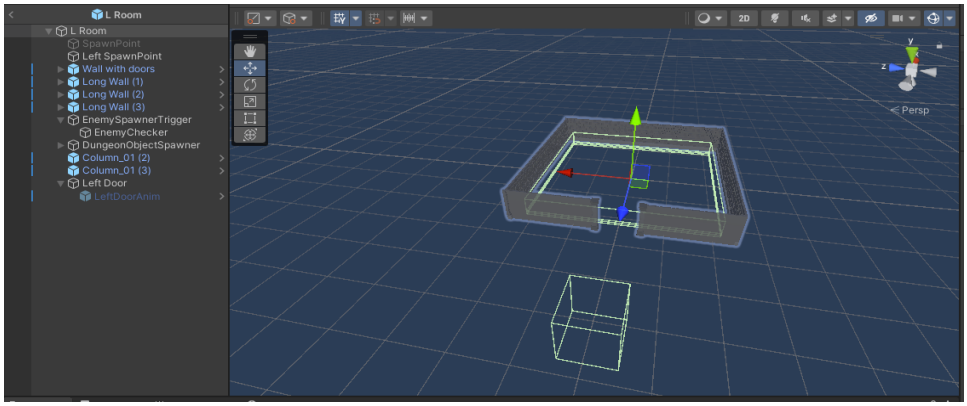
Slika 39. Početna prostorija

Slika 40 prikazuje donju prostoriju.



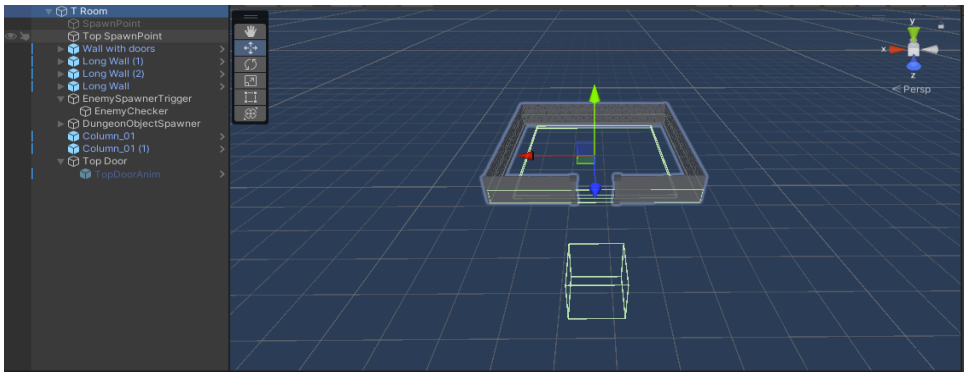
Slika 40. Donja prostorija

Slika 41 prikazuje lijevu prostoriju.



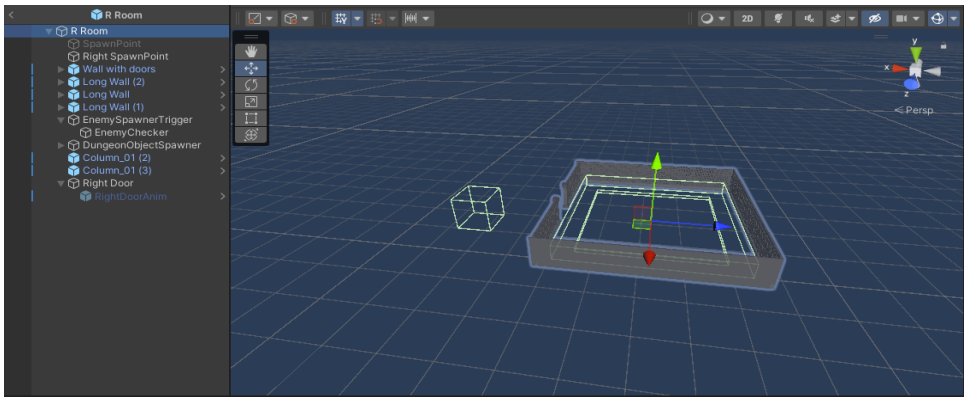
Slika 41. Lijeva prostorija

Slika 42 prikazuje gornju prostoriju.



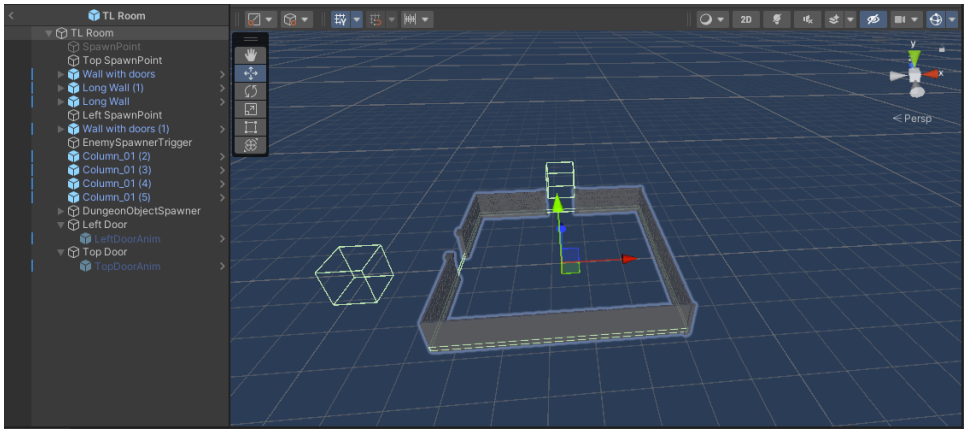
Slika 42. Gornja prostorija

Slika 43 prikazuje desnu prostoriju.



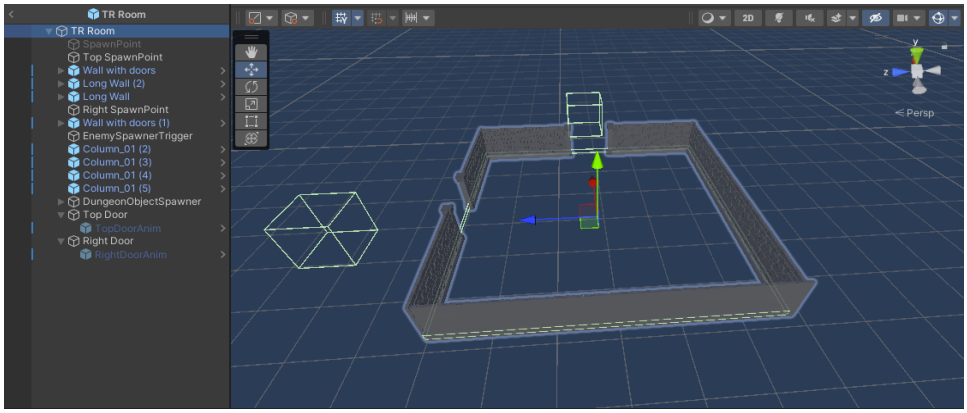
Slika 43. Desna prostorija

Slika 44 prikazuje prostoriju gore lijevo.



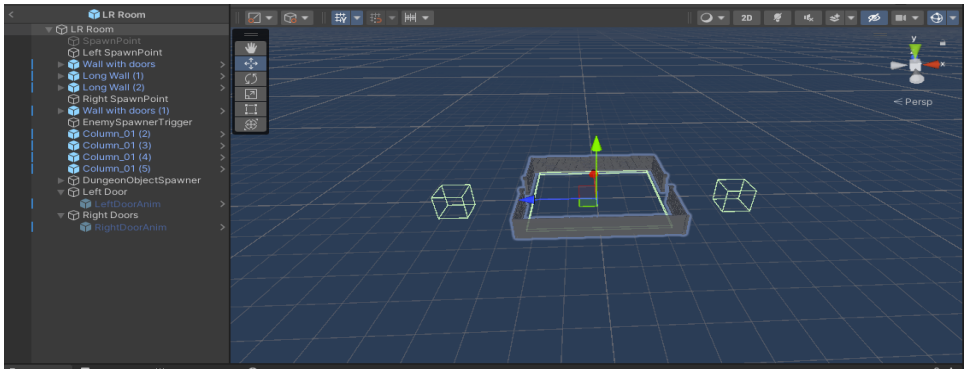
Slika 44. Prostorija gore lijevo

Slika 44 prikazuje prostoriju gore desno.



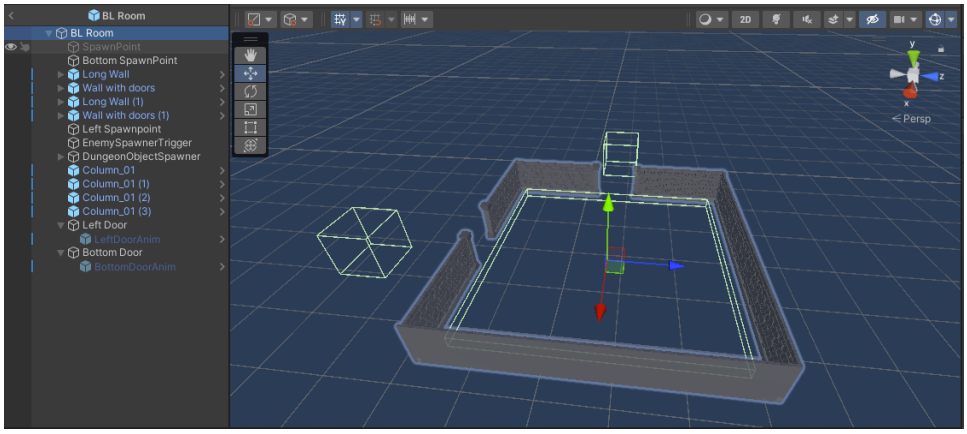
Slika 45. Prostorija gore desno

Slika 46 prikazuje prostoriju lijevo desno.



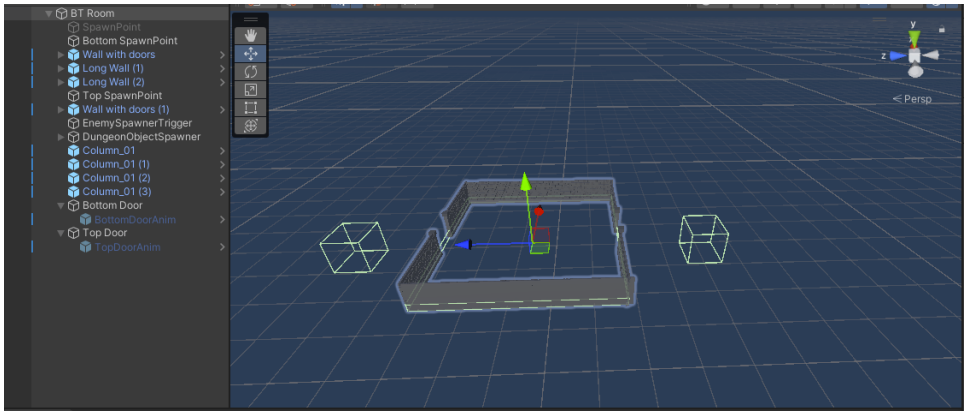
Slika 46. Prostorija lijevo desno

Slika 47 prikazuje prostoriju dolje lijevo.



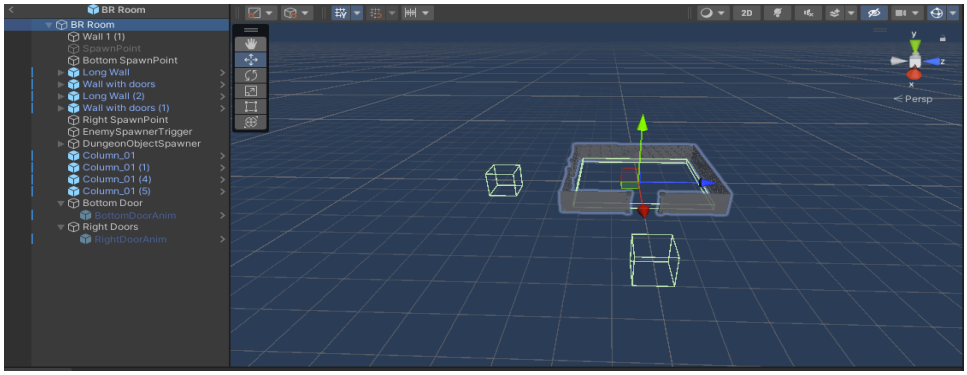
Slika 47. Prostorija dolje lijevo

Slika 48 prikazuje prostoriju gore dolje.



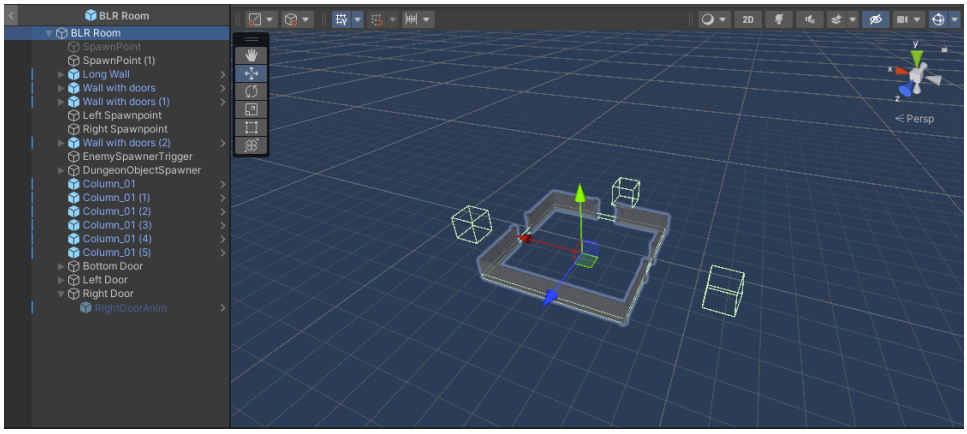
Slika 48. Prostorija gore dolje

Slika 49 prikazuje prostoriju gore lijevo.



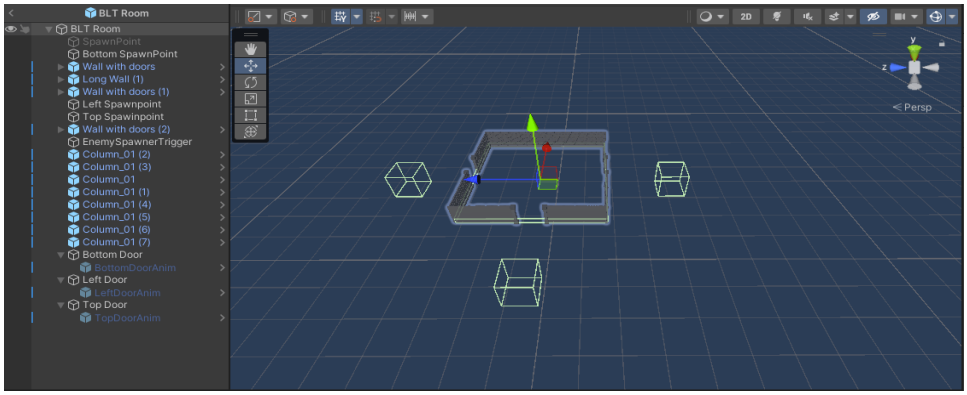
Slika 49. Prostorija gore lijevo

Slika 50 prikazuje prostoriju dolje lijevo desno.



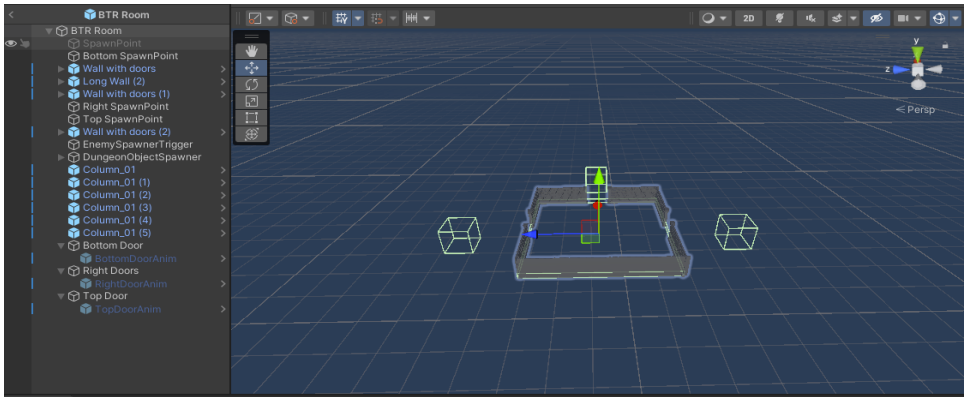
Slika 50. Prostorija dolje lijevo desno

Slika 51 prikazuje prostoriju dolje lijevo gore.



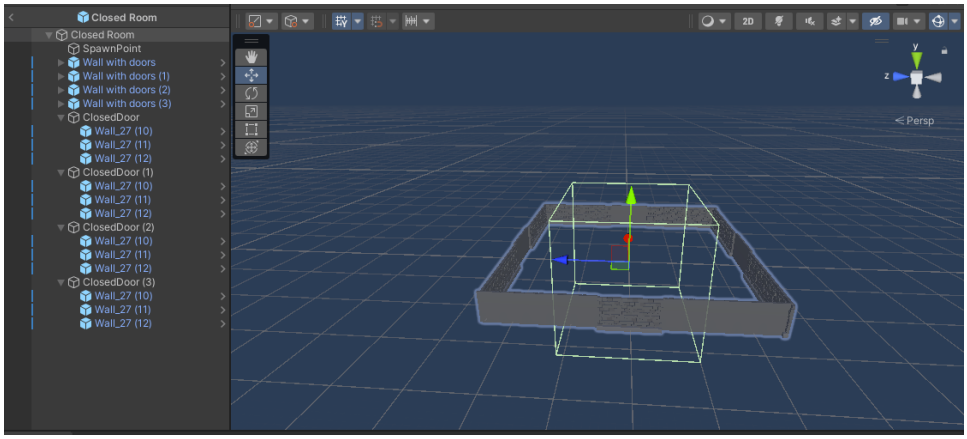
Slika 51. Prostorija dolje lijevo gore

Slika 52 prikazuje prostoriju dolje gore desno.



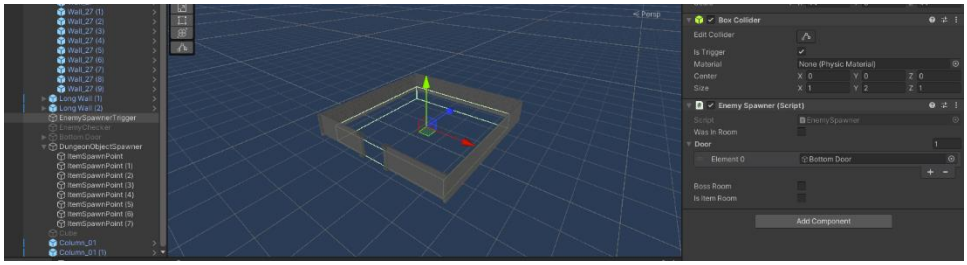
Slika 52. Prostorija dolje gore desno

Slika 53 prikazuje zatvorenu prostoriju.



Slika 53. Zatvorena prostorija

Slika 53 prikazuje okidač za stvaranje neprijatelja.



Slika 54. Okidač za stvaranje neprijatelja

EnemySpawnerTrigger je prazan objekt igre koji na sebi ima sudarač u obliku kutije kojem je vrijednost isTrigger postavljena na istinitu vrijednost. Sudarač je manji od veličine prostorije zato što igrač mora u potpunosti uči u prostoriju da bi se okidač izvršio. Također na njega je postavljena skripta EnemySpawner.cs koja je zadužena za stvaranje neprijatelja ili stvari koje pomažu igraču u igri. Prikaz koda je skripte EnemySpawner.cs je vidljiv u isječcima koda 42, 43, 44 i 45. Komponenta EnemySpawner koja je postavljena na prazan objekt igre u svakoj prostoriji na sebi ima sudarač kojem je vrijednost isTrigger postavljena na istinitu vrijednost.

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

public class EnemySpawner : MonoBehaviour
{
    public bool wasInRoom = false;
    EnemyList enemyList;
    private int rand;
    private int randomRoll;
    public GameObject[] door;
    EnemySpawnAndKillCount esakc;
    public bool bossRoom = false;
    public bool isItemRoom = false;
    private RoomTemplates templates;



    // Start is called before the first frame update
    void Start()
    {
        esakc = GameObject.FindGameObjectWithTag("Player").GetComponent<EnemySpawnAndKillCount>();
        enemyList=GameObject.FindGameObjectWithTag("EnemyList").GetComponent<EnemyList>();
        randomRoll=  Random.Range(2,6);
        templates=GameObject.FindGameObjectWithTag("Rooms").GetComponent<RoomTemplates>();

    }

    // Update is called once per frame
```

Isječak koda 42. Skripta EnemySpawner.cs 1.dio

U Start() metodi se dohvaćaju potrebne komponente, a to su: EnemySpawnAndKillCount komponenta koja se nalazi na objektu igre s oznakom “Player”, EnemyList komponenta koja se nalazi na objektu igre s oznakom „EnemyList“, RoomTemplates koja se nalazi na objektu igre s oznakom „Rooms“. Također se uzima nasumičan cijeli broj koji će se koristiti pri stvaranju neprijatelja.

```
// Update is called once per frame
void Update()
{
    if(esakc.checkIfAllKilled() && wasInRoom){
    for(int i =0;i<door.Length;i++){
            door[i].GetComponent<DoorAnimatorController>().doorIsClosing = true;
        }

        Invoke("OpenDoors",3.0f);
    }
}
```

Isječak koda 43. Skripta EnemySpawner.cs 2.dio

U Update() metodi provjerava se ako je esakc.checkIfAllKilled() istinite vrijednost i ako je wasInRoom istinite vrijednosti. Ako su obje varijable istinite postavlja se vrijednost

doorIsClosing komponente DoorAnimatorController na istinitu vrijednost. Te se nakon 3 sekunde poziva funkcija OpenDoors().

```
void OnTriggerEnter(Collider other){
    if(other.CompareTag("Player")){
        if(!isItemRoom){
            if(wasInRoom){
                return;
            }
            wasInRoom=true;
            for(int i =0;i<door.Length;i++){
                door[i].SetActive(true);
                FindObjectOfType<AudioManager>().playSound("DoorSlide");

            }
            if(!bossRoom){
                esakc.spawnCount(randomRoll);
                for(int i=0;i<randomRoll;i++){
                    rand=Random.Range(0,enemyList.enemyList.Length);
                    Instantiate(enemyList.enemyList[rand], transform.position,Quaternion.identity);

                }
            }else{
                //Create boss and next level trigger.
                esakc.spawnCount(1);
                Instantiate(templates.boss,transform.position,Quaternion.identity);
                Instantiate(templates.nextLevelTrigger,new Vector3(transform.position.x,2,transform.position.z),Quaternion.identity);

            }
        }


    }
}
```

Isječak koda 44. Skripta EnemySpawner.cs 3.dio

Funkcija OnTriggerEnter() se poziva kada neki objekt uđe u prostor okidača. Potom se provjerava ako je oznaka drugog objekta jednak vrijednosti Player. Ako je provjerava se da li je vrijednost varijable IsItemRoom postavljana na lažnu vrijednost, odnosno ako se u toj prostoriji neće stvoriti stvar koju igrač može pokupiti. Ako prostorija nije prostorija u kojoj se nalazi stvar koju igrač može pokupiti provjerava se vrijednost varijable wasInRoom, ta varijabla označuje ako li je igrač bio u prostoriji te ako nije izvršit će se ostatak koda, a ako je koristeći return će se prekinuti izvršavanje ostatka koda u funkciji. Nakon što igrač prvi puta uđe u prostoriju vrijednost wasInRoom se postavlja na istinitu vrijednost te se zatvaraju vrata tako da se objektima unutar door[] polja vrijednost SetActive() postavi na istinitu vrijednost te se potom pomoću AudioManagera pokreće zvuk koji ima naziv DoorSlide. Ako prostorija nije prostorija u kojoj se nalazi šef nivoa, poziva se funkcija esakc.spawnCount(randomRoll) kojoj se predaje nasumični broj iz Start() funkcije. Time se povećava brojač stvorenih neprijatelja za proslijeđenu vrijednost. Zatim se stvara isti broj neprijatelja iz objekta enemyList koji su nasumično odabrani. Ako prostorija je bossRoom, odnosno vrijednost bossRoom je jednaka istini, izvršava se else blok koda u kojem se broj stvorenih neprijatelja povećava za jedan te se stvara neprijatelj šef i okidač za iduću razinu.

```
public void setBoss(){
    bossRoom=true;
}
public void setItemRoom(){
    isItemRoom = true;
}
private void OpenDoors(){
    for(int i =0;i<door.Length;i++){
            door[i].SetActive(false);
        }
}
```

Isječak koda 45. Skripta EnemySpawner.cs 4.dio

Skripta EnemySpawner radi tako da kada igrač uđe u jednu od stvorenih prostorija pomoću sudarača se okida funkcija OnTriggerEnter() u kojoj se provjerava ako je oznaka objekta koji je ušao u okidač jednaka “Player”. Ako je igrač ušao u prostoriju u kojoj do sad nije bio aktivirat će se sva vrata u prostoriji odnosno igrač neće moći napustiti prostoriju dok ne ubije sve neprijatelje. Broj neprijatelja je određen nasumično koristeći funkciju Random.Range() čiji se rezultat sprema u varijablu randomRoll. Ista varijabla se koristi pri pozivanju metode spawnCount(int x) koja povećava broj stvorenih neprijatelja.

U Update() funkciji se provjerava da li je jednak broj ubijenih i stvorenih neprijatelja te ako je vrata se otvaraju I igrač može nastaviti prema drugim prostorijama. Ako je igrač bio u sobi ništa se neće desiti. Stvaratelj objekata je postavljen u svakoj prostoriji te sadrži polje unaprijed definiranih točaka na kojima će se stvoriti objekti igre.

Za stvaranje objekata je odgovorna skripta DungeonObjectsSpawner.cs koja je prikazana u isječku koda 46. U Start() metodi se dohvaća lista objekata te se potom poziva Spawn() funkcija koja za svaki od objekata u polju instancira jedan objekt iz liste.

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

public class DunbeonObjectsSpawner : MonoBehaviour
{

    private DungeonObjectsList dunObjList;
    private int rand;
    public GameObject[] spawnPoints;

    // Start is called before the first frame update
    void Start()
    {
        dunObjList=GameObject.FindGameObjectWithTag("DungeonObjects").GetComponent<DungeonObjectsList>();
        Spawn();
    }

    void Spawn(){

        for(int i = 0;i<spawnPoints.Length;i++){
            rand=Random.Range(0,dunObjList.dungeonObjects.Length);
            if(dunObjList.dungeonObjects[rand].name=="Trap Explosion"){
                Instantiate(dunObjList.dungeonObjects[rand],new Vector3(spawnPoints[i].transform.position.x,-2.2f,spawnPoints[i].transform.position.z),
                 spawnPoints[i].transform.rotation);
            }
            else{
                Instantiate(dunObjList.dungeonObjects[rand],spawnPoints[i].transform.position, spawnPoints[i].transform.rotation);
            }
        }
    }

}
```

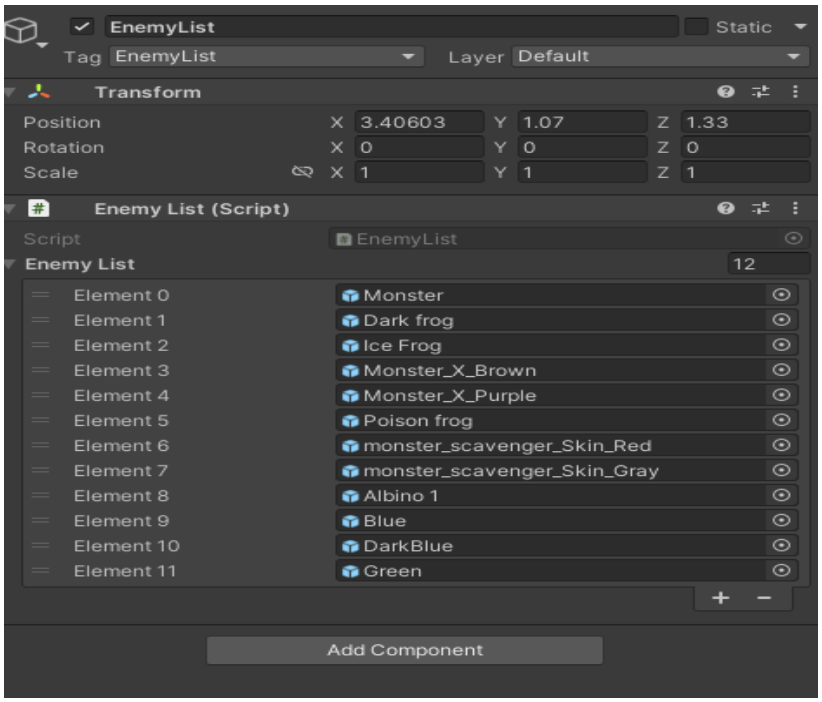
Isječak koda 46. Skripta DungeonObjectsSpawner.cs

Skripta EnemyList.cs u sebi sadrži javno polje tipa objekt igre koje je nazvano enemyList. To polje se koristi tako da se napuni objektima igre neprijatelj te EnemySpawner.cs skripta iz polja stvara nasumičnog neprijatelja. Definirano polje je vidljivo u isječku koda 47.

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

public class EnemyList : MonoBehaviour
{
    public GameObject[] enemyList;
}
```

Isječak koda 47. Skripta EnemyList.cs

Slika 55 prikazuje primjer popunjenog polja neprijatelja u inspektoru. Polje je veličine 12 te ako se želi povećati šansa stvaranja određenog neprijatelja potrebno je ponovo dodati isti objekt neprijatelja u listu.

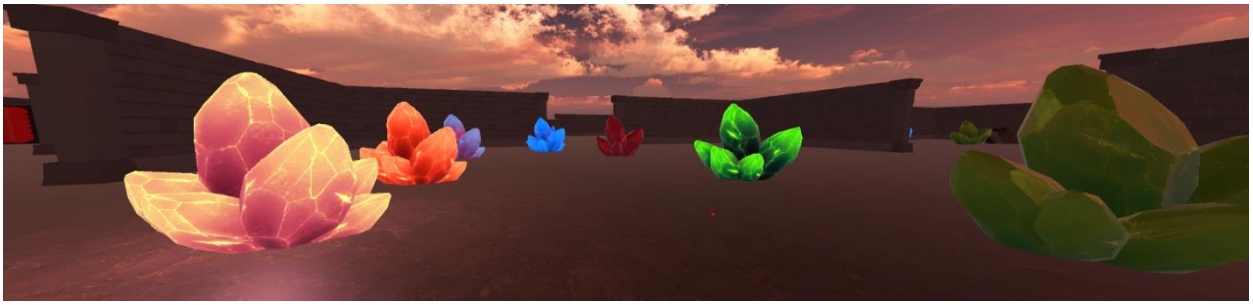


Slika 55. Lista neprijatelja u inspektoru

Lista neprijatelja je ispunjena Prefab objektima označenim oznakom „Enemy". Prefab sustav omogućuje stvaranje, postavljanje parametara i pohranjivanje objekta igre te pohranjivanje istog zajedno sa svim komponentama tog objekta igre, njihovim vrijednostima i njihovom djecom. Prefab objekti se mogu koristiti kao sredstvo višekratne upotrebe (Unity, 2023).

5.4.1.3 Magični kristal

Magični kristali su objekti igre koji pri aktivaciji imaju određeni efekt ovisno o tipu kristala. Kristali su prikazani na slici 56 te se jasno može vidjeti da se razlikuju po bojama. Svaki od kristala ima jedinstven efekt na igrača te ih je važno razlikovati.



Slika 56. Prikaz magičnih kristala u igri

U isječku koda 48 prikazan je prvi dio skripte CrystalEffects.cs

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

public class CrystalEffects : MonoBehaviour
{
    public GameObject crystalLight;

    private bool crystalActivated = false;
    public bool isHealthCrystal;
    public bool isDamageCrystal;
    public bool isPoisonCrystal;
    public bool isDangerousCrystal;
    public bool isMagicCrystal;
    public bool isSpeedCrystal;
    public bool isSlowCrystal;
```

Isječak koda 48. Skripta CrystalEffects.cs 1.dio

U prvom dijelu skripte CrystalEffects.cs je definiran objekt igre crystalLight koji je svijetlo koje će se postaviti na aktivno kada igrač aktivira kristal. Potom je definirana privatna varijabla crystalActivated koja označava ako je kristal aktiviran te je ona postavljena na lažnu vrijednost. Zatim slijede tipovi kristala od kojih je svaki tipa bool te se njihova vrijednost označuje u inspektoru. Isječak koda 49 prikazuje drugi dio skripte CrystalEffects.cs.

```
    void OnTriggerEnter(Collider other){
        if(other.tag == "Player"){
            if(!crystalActivated){
                crystalActivated = true;
                crystalLight.SetActive(true);
                if(isHealthCrystal){
                    other.GetComponent<PlayerHealth>().upadateMaxHealth(1);
                }
                if(isDangerousCrystal){
                    other.GetComponent<PlayerHealth>().upadateMaxHealth(-2);
                }
                if(isDamageCrystal){
                    other.GetComponent<PlayerAttack>().updateMeelePlayerAttack(1);
                }
                if(isPoisonCrystal){
                    other.GetComponent<PlayerHealth>().tickPoison();
                }
                if(isMagicCrystal){
                    other.GetComponent<PlayerMagic>().reduceForceFieldCooldown(1);
                    other.GetComponent<PlayerMagic>().reduceCooldown(1);
                }
                if(isSpeedCrystal){
                    other.GetComponent<PlayerMovement>().updateMovementSpeed(1);
                }
                if(isSlowCrystal){
                    other.GetComponent<PlayerMovement>().updateMovementSpeed(-1);
                }
            }
        }
    }
}
```

Isječak koda 49. Skripta CrystalEffects.cs 2.dio

U drugom dijelu CrystalEffects.cs skripte se izvršavaju efekti kristala kada im igrač priđe dovoljno blizu. Provjerava se ako je kristal aktivan i ako nije onda se postavlja na aktivno stanje u osvjetljava se tako što se objekt svijetla postavi na aktivnu vrijednost, također se vrijednost crystalActivated postavlja na istinitu vrijednost kako se više efekti ne bi izvršavali.

5.4.1.3 Zamke

Igrica sadrži tri vrste zamki, a to su: šiljci (eng. spikes), eksplozija (eng. explosion) i drobilica (eng. crusher). Prva zamka koja će biti opisana je drobilica čiji kod prikazuje isječak koda 50.

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

public class TrapCrusher : MonoBehaviour
{
    private int trapDamage = 10;
    // Start is called before the first frame update
    void Start()
    {
        FindObjectOfType<AudioManager>().playSound("TrapSlide");
    }

    void OnTriggerEnter(Collider other){
        if(other.tag=="Player"){
            if(other.GetComponent<PlayerHealth>().hasAntiTrapBracer == false){
                other.GetComponent<PlayerHealth>().updateHealth(-trapDamage/2);
            }
        }
        if(other.tag=="Enemy"){
            other.GetComponent<Enemy>().Damage(trapDamage);
        }
    }
}
```

Isječak koda 50. Skripta TrapCrusher.cs

Ako igrač ili neprijatelj uđe unutar drobilice te se njihovi sudarači sudare primit će štetu na životne bodove. Ako igrač posjeduje antiTrapBracer igrač neće primiti štetu od niti jedne zamke.

U ovom dijelu rada će biti opisana zamka eksplozije čiji kod je vidljiv u isječku koda 51.

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

public class TrapExplosion : MonoBehaviour
{

    public GameObject exposionOne,exposionTwo,bombOne,bombTwo,laser;
    // Start is called before the first frame update

    void OnTriggerEnter(Collider other){
        if(other.tag=="Player" || other.tag=="Enemy"){
            laser.SetActive(false);
            bombOne.SetActive(false);
            bombTwo.SetActive(false);
            exposionOne.SetActive(true);
            exposionTwo.SetActive(true);
            StartCoroutine(DestroyExplosionTrap());
        }
    }

    IEnumerator DestroyExplosionTrap(){
        yield return new WaitForSeconds(0.25f);
        Destroy(gameObject);

    }
}
```

Isječak koda 51. TrapExplosion.cs

Kada igrač ili neprijatelj uđe u sudarač zamke, ona se aktivira te se pokreću eksplozije i korutina u kojoj se nakon 0.25 sekundi uništava zamka.

Prvi dio koda zamke šiljaka je prikazan u isječku koda 52.

```
public class TrapSpikes : MonoBehaviour
{
    public int damage;
    public GameObject spikes;
    public Animator trapAnimator;
    bool trapIsActive = true;
    bool trapCanDealDamage = true;

    private void OnTriggerEnter(Collider other){
            if(other.tag=="Player" || other.tag=="Enemy"){
                if(trapIsActive){
                    trapIsActive= false;
                    spikes.SetActive(true);
                    trapAnimator.SetBool("trapIsActive",true);
                    FindObjectOfType<AudioManager>().playSound("SpikesTrap");
                    Invoke("PlayTrapCloseAnimation",2.0f);
                    trapCanDealDamage = true;
                }
                if(trapCanDealDamage && other.tag =="Player"){
                    if(other.GetComponent<PlayerHealth>().hasAntiTrapBracer == false){
                        other.GetComponent<PlayerHealth>().InflictDamage(damage);
                    }
                }
                if(trapCanDealDamage && other.tag =="Enemy"){
                    other.GetComponent<Enemy>().Damage(damage);
                    FindObjectOfType<AudioManager>().playSound("GetImpaledDummy");

                }
            }
        }

    private void PlayTrapCloseAnimation(){
        trapAnimator.SetBool("trapIsActive",false);
        trapCanDealDamage = false;
        Invoke("ResetTrap",2.0f);
    }
```

Isječak koda 52. Skripta TrapSpikes.cs 1.dio

Zamka s bodežima se aktivira kada neprijatelj ili igrač se sudare sa sudaračem od zamke. Potom se pokreće animacija te se nanosi šteta igraču ili neprijatelju i pokreće se zvuk putem upravitelja zvukom. Potom se poziva funkcija ResetTrap() koju prikazuje isječak koda 54.

```
    private void ResetTrap(){
        spikes.SetActive(false);
        trapIsActive = true;
    }
}
```

Isječak koda 53. Skripta TrapSpikes.cs 2.dio

U funkciji ResetTrap() se postavlja vrijednost trapIsActive na istinu što znači da zamka opet može oštetiti igrača ili neprijatelja.

5.4.2 Stvaranje pratoica

Skripta CompanionSpawner.cs je odgovorna za stvaranje pratilaca te će u nastavku biti opisana. Isječak koda 54 prikazuje prvi dio skripte CompanionSpawner.cs.

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

public class CompanionSpawner : MonoBehaviour
{

    public int val;
    public GameObject[] companions;

    void Awake(){
        if(GameObject.FindGameObjectWithTag("ValueHolder") != null){
            val = GameObject.FindGameObjectWithTag("ValueHolder").GetComponent<ValueHolder>().value;
        }

    }
```

Isječak koda 54. Skripta CompanionSpawner.cs 1.dio

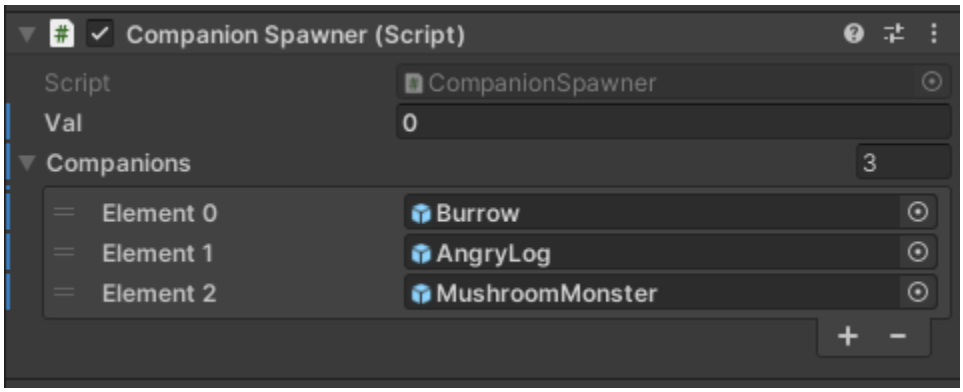
U metodi Awake() koja se izvodi prije Start() metoda se pronalazi objekt s oznakom „ValueHolder” te se provjerava ako uopće postoji. Ako se objekt pronađe dohvaća se vrijednost varijable value komponente ValueHolder. U nastavku će biti prikazan isječak koda 55 koji prikazuje drugi dio skripte CompanionSpawner.cs.

```
    void Start()
    {
        Instantiate(companions[val],new Vector3(transform.position.x,0,transform.position.z),companions[val].transform.rotation);
        Destroy(this);
    }

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

    }
}
```

Isječak koda 55. Skripta CompanionSpawner.cs 2.dio

U Start() metodi se koristeći funkciju Instantiate() se stvara objekt igre pratioca. Kada je potrebno stvoriti objekt igre putem koda onda se koristi funkcija Instantiate(). Prvi argument funkcije je objekt igre koji se treba stvoriti, drugi argument funkcije je Vector3 odnosno pozicija na kojoj se objekt treba stvoriti, te zadnji argument je rotacija objekta igre (French, 2022).

Nakon što se pratilac stvori u igrici, CompanionSpawner (hrv. stvaratelj pratioca) objekt se uništava jer više nije potreban, a ako igrač umre i pokrene igru stvaratelj pratioca će opet će ponovno biti stvoren te će vrijednost pratioca biti ista i preuzeta iz ValueHolder.cs skripte. Slika 57 prikazuje komponentu CompanionSpawner u inspektoru.



Slika 57. Prikaz komponente CompanionSpawner u inspektoru

Na praznom objektu igre se nalazi skripta CompanionSpawner.cs te su u polje objekata igre postavljeni objekti s oznakom „Companion". Vrijednost val je vrijednost koja se uzima iz skripte ValueHolder.cs a postavlja se tijekom odabira pratioca prije pokretanja igre.

5.4.2.1 Pratilac

U ovom dijelu rada će biti opisan rad pratioca s popratnim isječcima skripti. Isječke koda 56, 57 i 58 je moguće vidjeti u nastavku te oni prikazuju skriptu Companion.cs. U Start metodi se dohvaćaju sve potrebne komponente te ovisno o tipu pratioca se poziva odgovarajuća funkcija. Pronalazi se objekt igre s oznakom „Player" te se isti sprema u varijablu player. Zatim se pronalazi dijete objekta pod nazivom „HealingCircle" te se sprema u varijablu healingCircle.

Također se dohvaća komponenta EnemySpawnAndKillCount koja se sprema u esakc varijablu. Target se postavlja na igračevu poziciju zato što ako nema neprijatelja pratilac će pratiti igrača.

Potom se provjera vrijednost companionID varijable te ako je ona jednaka broju tri povećava se igračeva brzina kretanja za pet, a ako je companionID jednak broju dva poziva se funkcija enableCompanionDamageBoost() komponente PlayerAttack.

Varijabla nav se postavlja na komponentu NavMeshAgent kojoj se brzina postavlja na brzinu pratioca.

```
public class Companion : MonoBehaviour
{
    public int companionID;
    EnemySpawnAndKillCount esakc;
    private float healingCompanionCooldown = 10f;
    private bool companionCanHeal = true;
    private GameObject player;
    private Vector3 followPlayer ;
    public float speed = 5f;
    private Transform target;
    NavMeshAgent nav;
    GameObject enemy;
    public GameObject magicField;
    private Transform healingCircle;
    public int companionMagicDamage = 1;
    private bool magicCircleIsActive = false;
    private float magicAttackTickCooldown = 3.0f;
    private float timer = 0f;
    private bool canTick = true;


    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        healingCircle = player.transform.Find("HealingCircle");
        esakc = player.GetComponent<EnemySpawnAndKillCount>();
        target = player.transform;
        if(companionID==3){
            player.GetComponent<PlayerMovement>().updateMovementSpeed(5);
        }
        if(companionID==2){
            player.GetComponent<PlayerAttack>().enableCompanionDamageBoost();
        }

        nav = GetComponent<NavMeshAgent>();
        nav.speed = speed;
    }
```

Isječak koda 56. Skripta Companion.cs 1.dio

U Update() metodi se provjerava ako je companionID jednak broju jedan i ako je vrijednost varijable companionCanHeal istinita, onda se poziva funkcija healingCompanion(). Također se provjeravaju vrijednost varijabli enemySpawnCount i enemyKillCount te ako su jednaki pratilac će slijediti igrača. Potom će se isključiti njegov magični krug jer nema neprijatelja koje pratilac treba napadati. Ako vrijednosti ubijenih i stvorenih neprijatelja nisu jednake onda pratilac pronalazi neprijatelja te cijelo vrijeme ostaje na njegovo poziciji kako bi napravio štetu njegovim životnim bodovima. Također se provjerava može li pratilac oštetiti neprijatelja, te ako može radi mu štetu, a ako ne onda se vrijednost brojača uvećava za vrijednost Time.deltaTime.

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

        if(companionID==1 && companionCanHeal){
                healingCompanion();
            }
        if(esakc.enemySpawnCount==esakc.enemyKillCount){
            nav.SetDestination(new Vector3(target.position.x+5,0,target.position.z+5));
            if(magicCircleIsActive){
                magicField.SetActive(false);
                magicCircleIsActive=false;

            }

        }
        else{
            enemy = GameObject.FindGameObjectWithTag("Enemy");
            nav.SetDestination(enemy.GetComponent<Transform>().position);

        }

        if(timer>magicAttackTickCooldown){
            canTick=true;
        }else{
            timer+=Time.deltaTime;
        }

    }
```

Isječak koda 57. Skripta Companion.cs 2.dio

```
    private void OnTriggerStay(Collider other){
        if(other.gameObject.tag=="Enemy"){
                magicField.SetActive(true);
                magicCircleIsActive=true;
                if(canTick){
                    canTick = false;
                    other.gameObject.GetComponent<Enemy>().TakeMagicDamage(companionMagicDamage);
                    timer = 0;
            }
        }
    }

    void healingCompanion(){
        if(!(player.GetComponent<PlayerHealth>().health == player.GetComponent<PlayerHealth>().maxHealth)){
            player.GetComponent<PlayerHealth>().updateHealth(1);
            healingCircle.gameObject.SetActive(true);
            Invoke("TurnOffHealingCircle",1.5f);
            StartCoroutine(StartHealingCooldown());
        }

    }

    void TurnOffHealingCircle(){
        healingCircle.gameObject.SetActive(false);
    }

    public IEnumerator StartHealingCooldown(){
        companionCanHeal = false;
        yield return new WaitForSeconds(healingCompanionCooldown);
        companionCanHeal = true;

    }
}
```

Isječak koda 58. Skripta Companion.cs 3.dio

OnTriggerStay() metoda se koristi kako bi se provjeri sudara li se pratilac s neprijateljem te ako se sudara aktivira se magični krug te ako je vrijednost canTick istinite vrijednosti,

pratilac radi štetu neprijatelju za vrijednost varijable companionMagicDamage koje je prethodno definirana. Vrijednost varijable canTick se postavlja na lažnu vrijednost te se brojač (eng. timer) postavlja na vrijednost 0.

5.4.2.2 Efekti pratioca na igrača

Kada je odabran pratilac koji povećava napad, postavlja se vrijednost varijable companionDamageBoost na istinitu vrijednost što je vidljivo u isječku koda 59.

```
//COMPANION EFFECTS

public void enableCompanionDamageBoost(){
    companionDamageBoost=true;
}
```

Isječak koda 59. Skripta AttackArea.cs 1.dio

Unutar metode updateMeelePlayerAttack() se provjerava ako je vrijednost varijable companionDamageBoost jednaka istini, ako je svako povećanje štete koju igrač može nanijeti se dupla. Prikaz funkcije updateMeelePlayerAttack() je vidljiv u isječku koda 60.

```
public void updateMeelePlayerAttack(int damageMod){
    if(companionDamageBoost){
        meeleDamage=meeleDamage + damageMod*2;
    }else{
        meeleDamage+=damageMod;
    }
}
```

Isječak koda 60. Skripta AttackArea.cs 2.dio

Također pri pokretanju same igrice osnovna šteta koju igrač može nanijeti se također dupla što je vidljivo u isječku koda 61.

```
void Start(){
    if(companionDamageBoost){
        meeleDamage*=2;
    }
}
```

Isječak koda 61. AttackArea.cs 3.dio

5.4.3 Neprijatelji

Neprijatelji su objekti igre koji pokušavaju spriječiti igračev napredak kroz igru. Kao i igrač oni imaju životne bodove te štetu koju mogu nanijeti igraču. Za upravljanje neprijateljima je zadužena skripta Enemy.cs koja će biti prikazana kroz isječke koda 62, 63 i 64.

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

public class Enemy : MonoBehaviour
{
    EnemySpawnAndKillCount esakc;
    TriggerNextLevel tnl;
    PlayerHealth PlayerHealth;
    PlayerBlock playerBlocking;

    public int health = 10;
    public bool isBoss = false;
    public bool bossIsDead = false;
    public int damage = 5;
    public bool attacking = false;
    public bool isDying = false;
    public bool isTakingDamage = false;
    public bool isChasing = true;
    public NavMeshAgent nav;
    public Transform player;
    public float attackCooldown = 3.0f;
    public bool enemyAlreadyAttacked = false;
    public float sightRange,attackRange;
    public bool playerInSight, playerInAttackRange;

    void Start(){
        esakc = GameObject.FindGameObjectWithTag("Player").GetComponent<EnemySpawnAndKillCount>();
        player = GameObject.FindGameObjectWithTag("Player").transform;
        PlayerHealth = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerHealth>();
        playerBlocking = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerBlock>();
        nav = GetComponent<NavMeshAgent>();
    }
```

Isječak koda 62. Skripta Enemy.cs 1.dio

U Enemy.cs skripti potrebno je koristiti neke elemente drugih klasa kao što su EnemySpawnAndKillCount klasa, TriggerNextLevel, PlayerHealth i PlayerBlock. Također su definirane varijable koje opisuju klasu neprijatelja kao što su neprijateljevi životni bodovi, šteta koji neprijatelj može nanijeti igraču jednim udarcem, boolean varijable attacking, isDying, IsTakingDamage, isChasing i enemyAlreadyAttacked. Način na koji se te varijable koriste će biti opisan u nastavku. U Start() metodi se dohvaćaju potrebne komponente objekata igre koje su prethodno navedene. Javna varijabla isBoss predstavlja ako je neprijatelj šef neprijatelj, a varijabla bossIsDead predstavlja ako je neprijatelj šef mrtav. Ti neprijatelji se nazivaju šefovima iz razloga što su oni najjači neprijatelji. Predstavljaju hijerarhiju i igrici jer ako postoje slabi neprijatelji, logično je da postoje i jaki neprijatelji. S obzirom na to da igrica progresivno pojačava težinu, potrebno

je dodavati i nove i jače neprijatelje. Iz tog razloga svaki šef je jači od prethodnog i predstavlja drugačiji izazov. Jedna od prvih upotreba termina „Boss" za neprijatelja je bila u filmu Veliki Gazda (eng. The Big Boss) u kojem se glavnog neprijatelja naziva „The Big Boss" odnosno „Veliki Gazda". Naziv dolazi od šefova mafije (eng. mafia bosses) te se do danas zadržao u video igrama (Grayson, 2021).

```
void Update(){
        if(isChasing && !isDying){
            ChasePlayer();
        }
}

public void Damage(int damage){
    health = health - damage;
    isTakingDamage = true;
    CheckHealth();
}

private void CheckHealth(){
    if(health<=0){
        if(isBoss){
            bossIsDead = true;
            tnl = GameObject.FindGameObjectWithTag("NextLevelTrigger").GetComponent<TriggerNextLevel>();
            tnl.canTriggerNextLevel();

        }
        esakc.killCount();
        nav.SetDestination(transform.position);
        isDying = true;
        GetComponent<Collider>().enabled = false;
        Invoke("destoryEnemy",2.0f);
    }
    else{
        isTakingDamage = true;
    }
}
private void destoryEnemy(){
    Destroy(gameObject);
}

private void ChasePlayer(){
    nav.SetDestination(player.position);
}
```

Isječak koda 63. Skripta Enemy.cs 2.dio

U Update() metodi se provjerava ako neprijatelj naganja igrača te ako ga naganja i ne umire pozvat će se funkcija ChasePlayer(). Unutar funkcije Damage(int damage) koja prima cjelobrojnu varijablu se vrijednost neprijateljevih životnih bodova umanjuje za proslijeđenu vrijednost damage. Potom se varijabla isTakingDamage postavlja na istinitu vrijednost i ona se koristi pri pokretanju animacije. Također se poziva CheckHealth() funkcija. CheckHealth() funkcija provjerava životne bodove neprijatelja. Ako su oni manji od nule.

Ako jesu uvećava se broj ubijenih neprijatelja komponente EnemySpawnAndKillCount za jedan. Postavlja se položaj neprijatelja na njegov položaj kako se ne bi nigdje kretao po terenu jer umire te se vrijednost varijable isDying postavlja na istinitu vrijednost kako se

ne bi pozvala funkcija ChasePlayer() u Update() funkciji. Sudarač neprijatelja isključuje se kako se igrač i drugi neprijatelji ne bi više sudarali s njim te se nakon dvije sekunde poziva funkcija destoryEnemy() koristeći Invoke() funkciju. Također se provjerava da li je neprijatelj šef i ako je postavlja se vrijednost varijable bossIsDead na istinitu vrijednost te se dohvaća komponenta TriggerNextLevel i poziva se funkcija canTriggerNextLevel() odnosno igrač može proći na sljedeću razinu. Ako neprijatelj ima više životnih bodova od nule postavlja se vrijednost isTakingDamage na istinitu vrijednost kako bi se mogla pokrenuti prikladna animacija.

```
private void AttackPlayer(){
    nav.SetDestination(transform.position);
    transform.LookAt(player);
    isChasing = false;
    if(enemyAlreadyAttacked){
        return;
    }
    attacking = true;
    if(!(playerBlocking.playerIsBlocking)){
        PlayerHealth.updateHealth(-damage);
    }
    enemyAlreadyAttacked = true;
    Invoke("ResetAttack",1.0f);
}


private void ResetAttack(){
    enemyAlreadyAttacked = false;
}
private void OnTriggerEnter(Collider other){
    if(other.gameObject.tag=="Player"){
        AttackPlayer();
    }
}
 private void OnTriggerExit(Collider other){
    if(other.gameObject.tag=="Player"){
        attacking = false;
        isChasing = true;
    }
}
private void OnTriggerStay(Collider other){
    if(other.gameObject.tag=="Player"){
        attacking = true;
        AttackPlayer();

    }
}
```

Isječak koda 64. Skripta Enemy.cs 3.dio

Funkcija AttackPlayer() je funkcija koja se pokreće kada neprijatelj napada igrača odnosno kada mu je u dometu koji je određen sudaračem. Putem NavMeshAgent komponente se postavlja njegovo odredište na poziciju na kojoj se trenutno nalazi kako se ne bi micao dok izvršava udarac. Varijabla isChasing se postavlja na lažnu vrijednost

jer neprijatelj više ne ganja igrača već ga napada. Ako je neprijatelj već napao ostatak funkcije se neće izvršiti a ako nije postavlja se vrijednost varijable attacking na istinitu vrijednost te ako igrač ne blokira napad nanosi mu se šteta jednaka iznosu štete koju neprijatelj može napraviti. Potom se postavlja vrijednost varijable enemyAlreadyAttacked na istinitu vrijednost te se nakon jedne sekunde poziva funkcija ResetAttack() koristeći Invoke() metodu. U funkciji ResetAttack() se vrijednost enemyAlreadyAttacked varijable postavlja na lažnu vrijednost što znači da neprijatelj može ponovo napasti. OnTriggerEnter() metoda se poziva kada neki objekt uđe u sudarač objekta igre neprijatelj. Potom se provjerava oznaka drugog objekta te ako je ona jedna oznaci „Player" poziva se metoda AttackPlayer(). OnTriggerExit() odnosno objekt izađe iz sudarača objekta igre neprijatelja vrijednost varijable attacking se postavlja na lažnu vrijednost odnosno neprijatelj više ne napada i isChasing vrijednost se postavlja na istinitu vrijednost jer neprijatelj ponovo počinje ganjati igrača. OnTriggerStay() znači da će neprijatelj cijelo vrijeme kada mu je igrač u dometu napadati igrača.

Kako neprijatelji ne bi „klizali" po terenu potrebno im je dodati animacije te je za to odgovorna skripta AnimatorController.cs čiji će kod biti prikazan u isječcima koda 65 i 66.

```
public class AnimatorController : MonoBehaviour
{

    public Animator animator;
    Enemy enemy;
    void Start()
    {
        enemy = this.GetComponent<Enemy>();
    }

    // Update is called once per frame
    void Update()
    {
        if(enemy.isDying){
            playDeathAnimation();
            return;
        }else{
            if(enemy.attacking){
                playAttackAnimation();
            }
            if(!(enemy.attacking)){
                stopAttackAnimation();
            }

            if(enemy.isChasing){
                playRunningAnimation();
            }
            if(!(enemy.isChasing)){
                stopRunningAnimation();
            }

            if(enemy.isTakingDamage){
                playEnemyIsDamagedAnimation();
                Invoke("stopEnemyIsDamagedAnimation",1.0f);
            }
        }
```

Isječak koda 65. AnimatorController.cs 1.dio

Skripta AnimatorController.cs služi za upravljanje animacijama neprijatelja. Varijable koje su potrebne za upravljanje se nalaze u Enemy komponenti objekta igre neprijatelj.

Ako je varijabla isDying istinita pozvat će se metoda playDeathAnimation(), a ako nije provjerit će se napada li neprijatelj igrača pustit će se prikladna animacija pomoću metoda playAttackAnimation(). Ako je varijabla isChasing istinita pozvat će se metoda playRunningAnimation(), a ako je lažna pozvat će se metoda stopRunninAnimation().

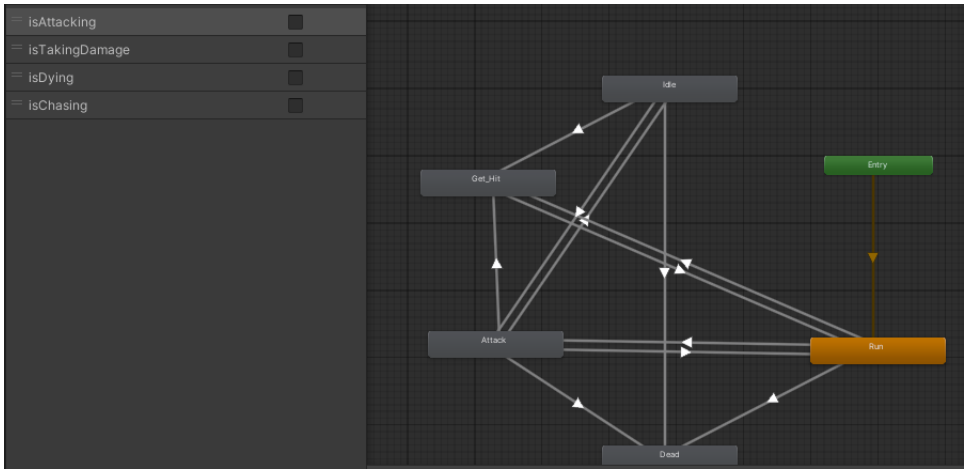
Ako neprijatelj primi štetu na svoje životne bodove varijabla isTakingDamage će biti istinita te će se pozvati metoda playEnemyIsDamagedAnimation() i pomoću Invoke() funkcije će se animacija zaustaviti nakon jedne sekunde pozivom metode stopEnemyIsDamagedAnimation().

```
    private void playAttackAnimation(){
            animator.SetBool("isAttacking",true);
    }
    private void stopAttackAnimation(){
        animator.SetBool("isAttacking",false);
    }

    private void playRunningAnimation(){
        animator.SetBool("isChasing",true);
    }

    private void stopRunningAnimation(){
        animator.SetBool("isChasing",false);
    }

    private void playDeathAnimation(){
        animator.SetBool("isDying",true);
    }
    private void playEnemyIsDamagedAnimation(){
        animator.SetBool("isTakingDamage",true);
    }
    private void stopEnemyIsDamagedAnimation(){
        enemy.isTakingDamage = false;
        animator.SetBool("isTakingDamage",false);

    }
}
```

Isječak koda 66. Skripta AnimatorController.cs 2.dio

Metoda playAttackAnimation() postavlja vrijednost parametra animatora isAttacking na istinitu vrijednost. Metoda stopAttackAnimation postavlja vrijednost parametra animatora isAttacking na lažnu vrijednost. Metoda playRunningAnimation postavlja vrijednost parametra animatora isChasing na istinitu vrijednost. Metoda stopRunningAnimation postavlja vrijednost parametra animatora isChasing na lažnu vrijednost. Metoda playDeathAnimation postavlja vrijednost parametra animatora isDying na istinitu vrijednost. . Metoda stopEnemyIsDamagedAnimation() postavlja vrijednost parametra animatora isTakingDamage na lažnu vrijednost te vrijednost varijable isTakingDamage komponente Enemy na lažnu vrijednost. Metoda playEnemyIsDamagedAnimation() postavlja vrijednost parametra animatora isTakingDamage na lažnu vrijednost

Slika 58 prikazuje animator neprijatelja goblin.



Slika 58. Animator neprijatelja goblin

Ulazno stanje animatora je animacija Run zato što se neprijatelji stvaraju na sredini prostorije te odmah se kreću prema igraču. Kada je parametar isAttacking istinit znači da je igrač u dometu neprijatelja te on može napasti igrača, tada prestaje animacija trčanja te započinje animaciju napada. Ako neprijatelj umre odnosno životni bodovi su mu jednaki nuli ili su manji od nule odmah se prekidaju sve druga animacije te se pokreće animacija umiranja neprijatelja. Isto tako vrijedi za Get_Hit animaciju odnosno animaciju kada igrač nanese štetu neprijateljevim životnim bodovima.

5.4.4 Brojač stvorenih i ubijenih neprijatelja

Kako bi se mogla otvarati i zatvarati vrata potrebno je provjeriti broj stvorenih i ubijenih neprijatelja, za to je zadužena skripta EnemySpawnAndKillCount.cs koja je razdvojena u dva dijela i vidljiva je u isječcima koda 68 i 69.

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

public class EnemySpawnAndKillCount : MonoBehaviour
{
    public int enemySpawnCount;
    public int enemyKillCount;
    private bool voicelinePlayed=false;
    int range;
    private AudioManager audioManager;

    void Start(){
        audioManager = GameObject.Find("AudioManager").GetComponent<AudioManager>();
    }

    public void spawnCount(int numberOfSpawned){
        enemySpawnCount+=numberOfSpawned;
        voicelinePlayed = false;
        range = Random.Range(0,6);
        if(range ==0){
            audioManager.playSound("NiceFellas");
        }
        if(range == 1){
            audioManager.playSound("LetMeTellYouSomething");
        }
        if(range == 3){
            audioManager.playSound("MonstersOverSnakes");
        }
        if(range == 4){
            audioManager.playSound("ScreamScared");
        }
        if(range == 5){
            audioManager.playSound("PutYourHandsDown");
        }
    }
```

Isječak koda 67. Skripta EnemySpawnAndKillCount.cs 1.dio

EnemySpawnAndKillCount.cs skripta se koristi za praćenje broja stvorenih i ubijenih neprijatelja. Također se koristi za pokretanje određenih zvukova u određenim trenutcima. U Start metodi se pronalazi upravitelj zvuka te se dohvaća njegova komponenta upravitelj zvuka. Definirana je metoda spawnCount koja kao argument prima cjelobrojnu varijablu numberOfSpawned (hrv. broj stvorenih). Vrijednost varijable enemySpawnCount se uvećava za vrijednost numberOfSpawned te se voicelinePlayed postavlja na lažnu vrijednost.

Potom se koristeći generator brojeva u rasponu brojeva koji radi tako da se preda minimalan broj koji se želi koristiti i maksimalan broj uvećan za jedan jer maksimalan broj nije uključen u raspon. U slučaju raspona (eng. range) varijable u ovom primjeru minimalna vrijednost je nula, a maksimalna šest, što znači da će raspon biti od nula do pet (French, 2021).

Potom se koristeći uvjetnu naredbu if provjerava vrijednost prethodno navedene varijable te se zvuk pokreće putem upravitelja zvuka ovisno o nasumičnoj vrijednosti.

```
    public void killCount(){
        ++enemyKillCount;
        if(enemyKillCount == 10){
            audioManager.playSound("10Kills");
        }
        if(enemyKillCount == 67){
            audioManager.playSound("67Kills");
        }


    }

    public bool checkIfAllKilled(){
        if(enemySpawnCount>0 && enemyKillCount>0 && enemySpawnCount == enemyKillCount){
            if(!voicelinePlayed){
                voicelinePlayed = true;
                range = Random.Range(0,2);
                Debug.Log(range);
                if(range ==0){
                    audioManager.playSound("AllEnemiesAreDead");
                }
                if(range == 1){
                    audioManager.playSound("AllEnemiesAreDead1");
                }
            }

            return true;
        }
        else{
            return false;
        }
    }
}
```

Isječak koda 68. EnemySpawnAndKillCount.cs 2.dio

Metoda killCount() se koristi kako bi se pratio broj ubijenih neprijatelja, odnosno svaki puta kada neprijatelj umre, vrijednost enemyKillCount (hrv. broj ubijenih neprijatelja) se uveća za jedan. Kada igrač ubije 10 neprijatelja i kada ubije 67 neprijatelja putem upravitelja zvukom se pokreću odgovarajući zvukovi. Unutar checkIfAllKilled() funkcije tipa bool se provjerava jesu li svi stvoreni neprijatelji ubijeni, potrebno je provjeriti ako su vrijednosti varijabli enemySpawnCount i enemyKillCount veće od nula jer ako nisu onda će pri pokretanju igrice ova funkcija vraćati istinitu vrijednost. Ako jesu vrijednosti jednake pokreće se odgovarajući zvuk putem upravitelja zvukom te vraća se istinita vrijednost , a ako nisu jednake vraća se lažna vrijednost.

5.4.5 Predmeti

Predmeti unutar igre pomažu igraču jer mu povećavaju vrijednost atributa poput životnih bodova. Skripta je razdvojena na 5 dijelova koje je moguće vidjeti u isječcima koda 69, 70, 71, 72 i 73.

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

public class Item : MonoBehaviour
{
    private Transform player;
    private AudioManager audioManager;
    void Start(){
        player = GameObject.Find("Player").GetComponent<Transform>();
        audioManager = GameObject.Find("AudioManager").GetComponent<AudioManager>();
    }

    void Update(){
        transform.LookAt(player);
    }
    [SerializeField] public string itemName;
    //Funkcija koja na koliziju s igracem dodijeli item vrijednost igraću
    private void OnTriggerStay(Collider other){
        //AKo je gameObject igrac, dohvati komponentu PlayerHealth i uvecaj mu health za healthValue
        if(Input.GetKeyDown(KeyCode.T)){
            if(other.gameObject.tag=="Player"){
                //Added item

                if(itemName == "SwordOfTheHephaestus"){
                    other.gameObject.GetComponent<PlayerAttack>().SwordOfTheHephaestus();
                    audioManager.playSound("ThisWillBurn");
                }
                //Added item

                if(itemName == "SwordOfBetrayal"){
                    other.gameObject.GetComponent<PlayerAttack>().SwordOfBetrayal();
                    audioManager.playSound("SwordOfBetrayal");

                }
```

Isječak koda 69. Skripta Item.cs 1.dio

U Start metodi se dohvaća Transform komponenta igrača i AudioManager. U Update() metodi se koristeći lookAt() funkciju, objekt igre predmet se rotira prema igraču odnosno konstantno „gleda” prema njemu. Kada je igrač unutar sudarača od stvari koje može pokupiti i on pritisne tipku „T“ pokupit će predmet. Svaki predmet ima poseban naziv te učinak na igrača.

```
if(itemName == "SwordOfTheCursed"){
    other.gameObject.GetComponent<PlayerHealth>().swordCurse();
    other.gameObject.GetComponent<PlayerAttack>().updateMeelePlayerAttack(10);
    audioManager.playSound("SwordOfTheCursed");

}
//Added item

if(itemName == "ChildsPlaySword"){
    other.gameObject.GetComponent<PlayerAttack>().updateMeelePlayerAttack(2);
    audioManager.playSound("WhyAmIGivenToys");

}
//Added item
if(itemName=="antiTrapBracer"){
    other.gameObject.GetComponent<PlayerHealth>().antiTrapBracer();
    audioManager.playSound("AntiTrapBracer");

}
//Added item
if(itemName == "SilverHelmet"){
    other.gameObject.GetComponent<PlayerHealth>().upadateMaxHealth(5);
    audioManager.playSound("SilverHelmet");

    // other.gameObject.GetComponent<PlayerHealth>().upadateMaxArmor(10);
}
//Added item
if(itemName == "Apple"){
    other.gameObject.GetComponent<PlayerHealth>().upadateMaxHealth(20);
    audioManager.playSound("Apple");
```

Isječak koda 70. Skripta Item.cs 2.dio

Stvar s nazivom SwordOfTheCursed (hrv. mač prokletih) postavlja vrijednost igračevih životnih bodova na vrijednost jedan, ali mu povećava vrijednost štete koju može nanijeti udarcem sjekire za 10. Također se pokreće odgovarajući zvuk putem AudioManagera te se za svaku stvar koju igrač pokupi pokreće odgovarajući zvuk. Stvar s nazivom ChildsPlaySword (hrv. mač za dječju igru) uvećava štetu koju igrač može nanijeti za dva. Stvar antiTrapBracer igraču daje imunitet na zamke, odnosno zamke mu više neće moći nanijeti štetu. SilverHelmet (hrv. srebrna kaciga) uvećava maksimalne životne bodove za pet, dok Apple (hrv. jabuka) uvećava životne bodove za 20.

```
if(itemName == "PotionOfStrength"){
    other.gameObject.GetComponent<PlayerAttack>().updateMeelePlayerAttack(5);
    audioManager.playSound("PotionPickup");

}
//Added item
if(itemName == "PotionOfMagicPower"){
    other.gameObject.GetComponent<PlayerAttack>().updateMagicDamage(5);
    audioManager.playSound("PotionPickup");

}
//Item added
if(itemName == "PotionOfCooldown"){
    other.gameObject.GetComponent<PlayerMagic>().reduceCooldown(3);
    audioManager.playSound("PotionPickup");

}
//Added item
if(itemName == "ForceFieldPower"){
    other.gameObject.GetComponent<PlayerAttack>().updateCircleDamage(3);
    audioManager.playSound("ItemPickup");


}
//Added item
if(itemName == "PotionOfMagicCircleCooldown"){
    other.gameObject.GetComponent<PlayerMagic>().reduceForceFieldCooldown(3);
    audioManager.playSound("ItemPickup1");



}
//Added item
if(itemName == "PotionOfLifeForceAbsorption"){
    other.gameObject.GetComponent<PlayerAttack>().lifesteal();
    audioManager.playSound("PotionPickup");
```

Isječak koda 71. Skripta Item.cs 3.dio

Napitak snage (eng. potion of strength) koji igraču uvećava štetu koju može nanijeti neprijatelju za pet. Napitak magične moći (eng. potion of magic power) koji igraču uvećava štetu koju nanosi magičnim napadom za pet. Napitak ohlađivanja (eng. potion of cooldown) smanjuje vrijeme koje je potrebno za izvođenje magičnog napada za tri sekunde. Napitak hlađenja magičnog kruga (eng. potion of magic circle cooldown) umanjuje vrijeme potrebno za izvođenje magičnog kruga za tri sekunde. Moć polja sile (eng. force field power) uvećava štetu koju nanosi magično polje za tri. Napitak apsorpcije životne sile (eng. potion of life force absorption) omogućava da se igrač regenerira životne bodove za svaki uspješni udarac nad neprijateljem.

```
if(itemName == "MercyOfAGod"){
    other.gameObject.GetComponent<PlayerHealth>().MercyOfAGod();
    audioManager.playSound("MercyOfAGod");

}
//Added item
if(itemName == "BootsOfHermes"){
    other.gameObject.GetComponent<PlayerMovement>().updateMovementSpeed(
    audioManager.playSound("BootsOfHermes");

}
//Added item
if(itemName == "ShieldOfBruised"){
    other.gameObject.GetComponent<PlayerHealth>().UpdateMaxArmor(10);
    audioManager.playSound("MoreArmor");

}
//Added item
if(itemName == "ShieldOfTheFallen"){
    other.gameObject.GetComponent<PlayerHealth>().UpdateMaxArmor(5);
    audioManager.playSound("MoreArmor");

}
//Added item
if(itemName == "ShieldOfTheWicked"){
    other.gameObject.GetComponent<PlayerHealth>().UpdateMaxArmor(20);
    audioManager.playSound("MoreArmor");

}
//Added item
if(itemName == "SupremeAidKit"){
    other.gameObject.GetComponent<PlayerHealth>().upadateMaxHealth(20);
    audioManager.playSound("AidKit");

}
//Added item
```

Isječak koda 72. Skripta Item.cs 4.dio

Stvar s nazivom Božja milost (eng. mercy of a God) je stvar koja igraču daje mogućnost da nastavi s igrom nakon što umre, odnosno kada igraču životni bodovi padnu na nulu ili ispod nule, igrač neće umrijeti već će mu se napuniti životni bodovi i štit. Stvari kojima naziv počinje s riječi štit (eng. shield) uvećavaju vrijednost igračevog štita za određenu vrijednost. Stvar koja nosi naziv Hermesove čizme (eng. boots of Hermes) uvećava brzinu kretanja igrača.

```
            if(itemName == "BasicAidKit"){
                other.gameObject.GetComponent<PlayerHealth>().upadateMaxHealth(5);
                audioManager.playSound("AidKit");

            }
            //Added item
            if(itemName == "AdvancedAidKit"){
                other.gameObject.GetComponent<PlayerHealth>().upadateMaxHealth(10);
                audioManager.playSound("AidKit");

            }

            if(itemName == "ForgeByBlood"){
                other.gameObject.GetComponent<PlayerHealth>().ForgeByBlood();
                audioManager.playSound("ForgeByBlood");

            }
            //Added item
            if(itemName == "BomberSpawner"){
                other.gameObject.GetComponent<PlayerAttack>().BomberSpawner();
                audioManager.playSound("BomberSpawner");

            }

            Destroy(gameObject);
        }
    }
}
```

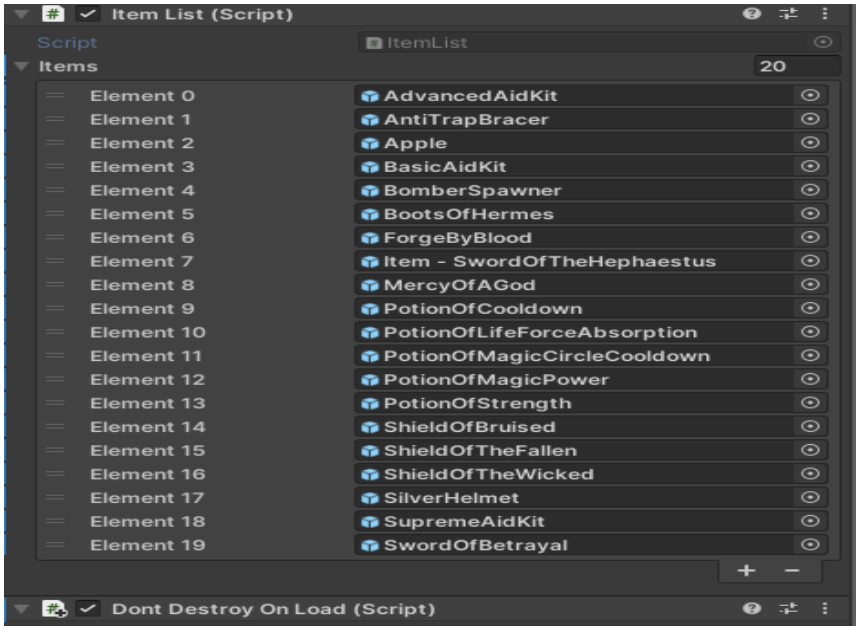
Isječak koda 73. Skripta Item.cs 5.dio

Stvari koje u svom nazivu imaju riječi pribor za pomoć (eng. aid kit) igraču uvećavaju maksimalni iznos životnih bodova za određeni iznos ovisno o stvari. Unikatna stvar u ovoj igrici je stvar pod nazivom kovanje krvlju (eng. forge by blood) te ono omogućuje igraču da pritiskom tipke X zamijeni životne bodove za štit. I zadnja stvar u igrici je stvaralac bombardera (eng. bomber spawner.) koji na udarac stvori bombu koja pronalazi neprijatelja i ide prema njemu te kada dođe do njega eksplodira ili ako nije došla do neprijatelja bomba eksplodira nakon četiri sekunde od nastajanja. Nakon što igrač pokupi stvar te se izvrši odgovarajući kod, ta stvar se uništava jer više nije potrebna te se tako naznačuje igraču da je pokupio stvar. U listi predmeta su sadržani svi predmeti u igrici koje igrač može pokupiti. Na prazni objekt igre je postavljena skripta ItemList.cs u kojoj se nalazi javna lista objekata igre nazvana items. Ta lista se dohvaća pri stvaranju svakog nivoa te se iz nje uzima jedan od predmeta koji se stvara. Skripta ItemList je vidljiva u isječku koda 74.

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

public class ItemList : MonoBehaviour
{

    public List<GameObject> items;

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

    }

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

    }
}
```

Isječak koda 74. Skripta ItemList.cs

U inspektoru se lista predmeta puni predmetima, odnosno potrebno je dovući objekte igre s oznakom „Item“. Slika 59 prikazuje listu predmeta u inspektoru.



Slika 59. Lista predmeta koji se mogu stvoriti unutar igrice

DestroyList.cs skripta je list objekata igre koja se koristi za uništavanje većeg broja objekata pri određenim radnjama. Primjerice igrač želi izaći iz igre u glavni izbornik, kako

bi se to moglo izvesti potrebno je uništiti objekte iz igre za koje je stavljeno da se ne uništavaju pri učitavanju scena. Moguće je vidjeti kod DestroyList.cs u isječku koda 75.

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

public class DestroyList : MonoBehaviour
{

    public List<GameObject> itemsToDestroy;
    public GameObject RunStatsSaver;

    // Start is called before the first frame update
    void Start()
    {
        RunStatsSaver = GameObject.FindGameObjectWithTag("RunStatsSaver");
        StartCoroutine(addItemsToList());

        // Invoke("addCompanionToTheArray",.5f);

    }

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

    }

    IEnumerator addItemsToList(){
        yield return new WaitForSeconds(0.2f);
        itemsToDestroy.Add(GameObject.FindGameObjectWithTag("Companion"));
        itemsToDestroy.Add(GameObject.FindGameObjectWithTag("Player"));
        itemsToDestroy.Add(GameObject.Find("AudioManager"));
        itemsToDestroy.Add(GameObject.Find("Canvas"));
        itemsToDestroy.Add(GameObject.FindGameObjectWithTag("ValueHolder"));

    }
}
```

Isječak koda 75. Skripta DestroyList.cs

U Start() metodi se pronalazi objekt igre s oznakom RunStatsSaver i pokreće se korutina addItemsToList() koja nakon 0,2 sekunde dodaje određene objekte u listu.

6. Zaključak

Kroz ovaj rad je opisana izrada roguelike igrice u Unity razvojnom okruženju te svi elementi koji su bili potrebni pri izradi igrice Silent Nut. U radu su objašnjeni pojmovi poput skripti, nasumične generacije brojeva, stvari koje igrač može pokupiti, okruženja igrice, proceduralnog stvaranja terena, vječne smrti i drugih elemenata unutar igre.

Igrač je upućen u priču igrice te su opisani svi UI elementi igrice, efekti neprijatelja i kako ti efekti utječu na igrača, utjecaj stvari u igrici i utjecaj kristala. Također je prikazano korištenje glavnog izbornika, izbornika pauze te izbornika smrti i generacije objekata unutar svake generirane prostorije. Igrač može pokrenuti igru iz početka kada umre bez da mora ponovo otvarati glavni izbornik. Također je objašnjeno pozivanje funkcija s odgodom korištenjem Invoke() funkcije i korutina.

Kako bi se poboljšala trenutna igrica moguće je dodati još neprijatelja te im dodati nove napade i efekte. Kako bi se produbilo korištenje stvari koje igrač može pokupiti moguće je dodati još stvari s različitim efektima koji će igraču pomoći ili odmoći tijekom njegove igre. Također je moguće povećati broj zamki koje mogu oštetiti neprijatelje i igrača. Također bi se igrica poboljšala stvaranjem novih animacija i efekata.

Zahtjevan je zadatak balansirati aspekte roguelike igrice jer igrica nagrađuje vještinu igrača, ali potrebno je uzeti u obzir da igrica ne smije biti nemoguća za pobijediti je, ali isto tako ne smije biti trivijalna. Potrebno se usredotočiti na jako dobre igrače te igru raditi prema njihovim zahtjevima, a nadati se da će manje iskusni igrači doći na isti nivo vještine kao iskusni igrači. Čitatelji ovog rada trebali bi pobliže shvatiti kako balansirati i upravljati svojom roguelike igricom te na koji način ostvariti roguelike elemente u igrici.

Sažetak

Tema ovog diplomskog rada je razvoj roguelike igrice u razvojnom okruženju Unity što obuhvaća opis procesa izrade video igrice, implementacije elemenata videoigre i prikaz videoigre. Kroz rad su objašnjene skripte koje su potrebne za ispravan rad te na koji su način one međusobno povezane. Objašnjen je način na koji radi proceduralno stvaranje terena, atributi igrača, neprijatelji te ostatak okruženja u kojem se igrač nalazi. Pojašnjeno je kako se igrač kreće svijetom te što sve može imati utjecaj na igrača. Čitatelji ovog diplomskog rada bi trebali steći potrebno znanje za razvoj slične roguelike igrice.

Ključne riječi: unity, roguelike, razvoj videoigre, programiranje, c#, videoigra

Abstract

The topic of this thesis is the development of a roguelike game in the Unity development environment, which includes a description of the video game creation process, the implementation of video game elements and the display of the video game. The thesis explains the scripts that are necessary for proper execution of the application and how they are interconnected. The way procedural terrain generation works, player attributes, enemies and the rest of the environment in which the player exists. It is clarified how the player moves through the world and everything that can have an impact on the player. Readers of this thesis should acquire the necessary knowledge to develop similar roguelike games.

Key words: unity, roguelike, game develompent, programming, c#, videogame

Literatura:

Amlin, J. (2021, August 13). *Working with Arrays Part II: For and Foreach Loops*. Medium. https://levelup.gitconnected.com/working-with-arrays-part-ii-for-and-foreach-loops-cc4fd1499e8c

Booth, O. (2022, May 9). *What is Delta Time?*. oliverbooth.dev. https://blog.oliverbooth.dev/2022/01/31/what-is-delta-time/

Bycer, J. (2021). *Game design deep dive: Roguelikes*. CRC Press.

Carrillo, U. (2022, June 23). *Using Coroutines in Unity*. LogRocket Blog. https://blog.logrocket.com/using-coroutines-unity/

French, J. (2022, October 20). *How to use random values in Unity (with examples)*. Game Dev Beginner. https://gamedevbeginner.com/how-to-use-random-values-in-unity-with-examples/

French, J. (2023, April 16). *How to spawn an object in Unity (using instantiate)*. Game Dev Beginner. https://gamedevbeginner.com/how-to-spawn-an-object-in-unity-using-instantiate/

Garbett, S. L. (2023, January 4). *How to code a physics-based character controller in Unity3D*. MUO. https://www.makeuseof.com/unity3d-physics-character-controller-build-code/

Grayson, N. (2021, February 19). *Why do we call the hardest video game enemies "bosses," anyway?* Kotaku. https://kotaku.com/why-do-we-call-the-hardest-video-game-enemies-bosses-a-1846301973

Haas, J. K. (2014). A history of the unity game engine. *Diss. Worcester Polytechnic Institute*, *483*(2014), 484.

Harris, J. (2021). *Exploring roguelike games*. CRC P.

Hijazi, M. (2021, July 1). *Tip of the day: Modular health system unity*. Medium. https://medium.com/nerd-for-tech/tip-of-the-day-modular-health-system-unity-8f5d2f187027

Pagán Dávila, D. (2021, April 6). *OnCollisionEnter vs. ontriggerenter - when to use them*. Medium. https://levelup.gitconnected.com/oncollisionenter-vs-ontriggerenter-when-to-use-them-56d42772dd22

Pagán Dávila, D. (2021c, May 15). *How to Play Sound Effects in Unity*. Medium. https://levelup.gitconnected.com/how-to-play-sound-effects-in-unity-6a122bb32970

Parker, R. (2017). The culture of Permadeath: Roguelikes and terror management theory. *Journal of Gaming & Virtual Worlds*, *9*(2), 123–141. https://doi.org/10.1386/jgvw.9.2.123_1

Unity documentation. Unity. (2023). https://docs.unity.com/

Watts, J. (2022, July 30). *Layer masks in Unity*. Medium. https://medium.com/@joshwatts592/layer-masks-in-unity-59a7df183676

Warnock, T. (1987). Random-number generators. *Los Alamos Science*, *15*(1987), 137-141.

Zhang, B., & Hu, W. (2017). Game special effect simulation based on particle system of Unity3D. *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. https://doi.org/10.1109/icis.2017.7960062

Silent Nut GitHub: https://github.com/VedranNejedly/Game

Popis slika:

Popis isječaka koda: