Razvoj igre Zombi Apokalipsa u Unity Engine

Svetec, Mario

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:600057

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Odjel za informatiku

Informacijski i komunikacijski sustavi

Mario Svetec

Razvoj igre Zombi Apokalipsa u Unity Engine

Diplomski rad

Mentor: doc. dr. sc. Marina Ivašić-Kos

Rijeka, kolovoz 2018.

Sadržaj

Sažetak 1
1.Uvod 2
2. Sučelje Unity-a 3
2.1. Inspector 4
2.2. Hierarchy 5
2.3. Scene View 6
2.4. Game View 6
2.5. Assets i Console 7
3. Skripte 8
4. Igra Zombie-Apocalypse 10
4.1. Model igrača 10
4.1.1. Kretanje igrača 11
4.1.2. Zdravlje igrača 13
4.1.3. Glad igrača 14
4.2. Oružja 16
4.2.1 Pucanje 20
4.2.2. Animacije oružja 21
4.2.3. Zvukovi oružja 23
4.3.1. Inventory 24
4.3. 3D objekti 28
4.3.1. Energetski napitci 28
4.3.2. Hrana 31
4.3.3. Prva pomoć i bandaže 31
4.3.4. Metci 32
4.3.5. Auto 32
4.4. Neprijatelj (Zombie) 36
4.4.1. Kretanje neprijatelja 37
4.4.2. Napadanje igrača 40
4.4.3. Zdravlje neprijatelja 41
4.5. Teren 42
4.6. Sučelje igre 43
4.6.1. Spremanje stanja igre 44
4.6.2. Učitavanje spremljenog stanja 45
4.6.2. Promjena kontroli za igranje. 46
5. Zaključak 48
LITERATURA 49

Sažetak

U ovom diplomskom radu ću opisati način izrade 3D igre u Unity-u. Igra će biti u prvom licu i nosit će naziv Zombie Apocalypse. Prvo ću opisati sami alat Unity, njegovo sučelje i način korištenje. Nakon opisa alata, slijedi opis izrade igre. Opisat ću izradu modela igrača. opisati kretanje igrača, zdravlje i glad. Uz modela igrača, opisat ću i model neprijatelja te njegovu inteligenciju, tj. način interakcije s igračem. U igri se nalazi još nekoliko objekata kao što su oružja, hrana i automobil. Kod objekta oružja ću opisati način pucanja, animacije i logiku koja upravlja osobinama svakog oružja. Nakon opisa svih objekata u igri, opisat ću sučelje igre, opciju mijenjanja kontroli igre i način spremanja igre.

Ključne riječi: PlayerHealth, WeaponManager, Equipment, EquipmentManager, EnemyMove, EnemyAttack

1.Uvod

U ovom diplomskom radu će biti opisana igra za stolna računala Zombi Apokalipsa i njen razvoj.. Igra je izrađena u Unity Enginu [13] u kojem su ujedno podešeni i svi 3D modeli koji se nalaze u samoj igri. Uz navedeni alat za kreiranje igara, korišteni alati su bili i Gimp [14], Audacity [15] i Blender [16].

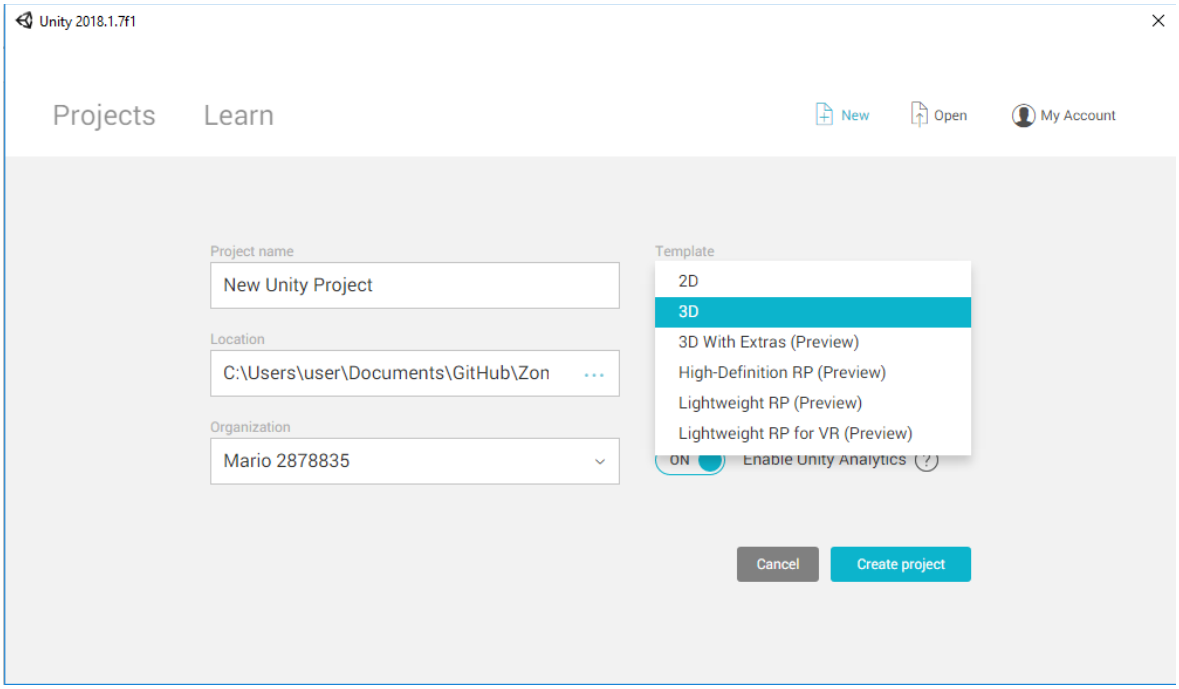
Igra Zombi Apokalipsa igra pucanja i preživljavanja u prvom licu po uzoru na trenutno najpopularnijoj igri na tržištu, PUBG [17] i popularnu televizijsku seriju The Walking Dead [18]. Cilj igre Zombi Apokalipsa je pronaći mjesto na ogromnom terenu koje vodi u sigurno zonu. Igra započinje tako da se igrač pojavi na određenom dijelu terena bez bilo kavih pomagala te pokušava pronaći oružja i ostala pomagala koja će mu pomoći da ga neprijatelji (Zombie) ne ubiju. Igra nije vremenski ograničena, tj. igra se prekida kada igrač pronađe sigurnu zonu ili ako umre. Igrač u igri ne mora samo voditi brigu o neprijateljima koji su raspoređeni na terenu, nego i o svojem zdravlju i gladi. To uvelike otežava preživljavanje u igri jer igrač mora uz oružja, tražiti i hranu.

Igra se sastoji od nekoliko važnih dijelova koji će biti dalje u radu detaljno obrađeni. Neki od važnih dijelova su korisnička sučelja od kojih razlikujemo dvije vrste. Prilikom pokretanja same igre prvo se pojavljuje početno korisničko sučelje preko kojeg možemo pokrenuti novu igri ili učitati spremljenu igru. Druga vrsta sučelja je korisničko sučelje koje se pokreće tijekom same igre na želju igrača. Navedeno sučelje omogućuje dodatne operacije spremanja igre i promjena kontroli.

Također važni dijelovi igre su objekti koji se pojavljuju u igri te njihova međusobna povezanost. Objekte koje razlikujemo u igri su: igrač, neprijatelji, oružja, sredstva za smanjenje gladi, sredstva za povećanja zdravlja, razni metci za oružja i automobil kojim se igrač može voziti po terenu.

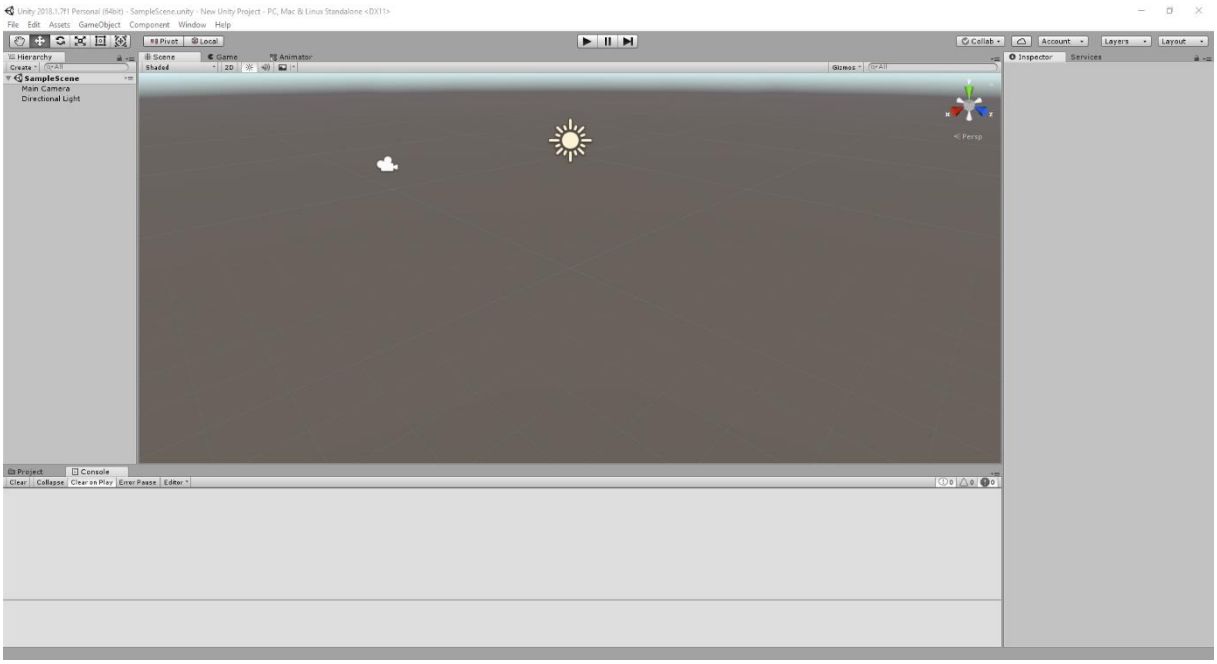
2. Sučelje Unity-a

Unity je cross-platform game engine izrađen od strane Unity Technologies koji je izdan u lipnju 2005. godine. Cross-platform znači da se igre koje su programirane u njemu mogu lansirati za gotovo svaki uređaj, neovisno o kojem se uređaju i operacijskom sustavu radi [1]. Kada pokrenemo Unity, prvo nam se otvori sučelje koje služi kako bi smo kreirali novi projekt ili pokrenuli već postojeći. U početnom sučelju su nam prikazani već postojeći projekti koje možemo, klikom na njihovo ime pokrenuti ili pritiskom na opciju New stvoriti novi. Kada odaberemo opciju New, Unity nam ponudi da izaberemo koju vrstu igre želimo kreirati, kao što je vidljivo na slici 1. Možemo birati između nekoliko opcija, a te opcije su: 2D igre, 3D igre s nekoliko varijanti postavke grafike i opcija za igre virtualne stvarnosti. Za ovu igru odabrana je opcija 3D.



Slika 1. Kreiranje projekta

Kao što vidimo na slici 2, Unity se sastoji od nekoliko elemenata, a to su: Inspector, koji se nalazi s desne strane. Kada prvi puta pokrenemo Unity, na lijevoj strani nalazi se Hierarchy. U sredini samog programa nalaze se Scene View i Game View te na dnu se nalaze Assets i Console. Pozicije navedenih elementa možemo poslije mijenjati tako da ih jednostavno Drag and Drop metodom pomaknemo. U nastavku će biti opisani svi navedeni elementi Unity-a.



Slika 2. Sučelje Unity-a

2.1. Inspector

Inspektor nam pokazuje osobine nekog objekta koji se nalazi u igri. Kada odaberemo neki objekt u Hierarchy ili Scene View-u otvori nam se na desnoj strani Inspektor koji pokazuje svojstva objekta, koja ujedno možemo mijenjati i dodavati neka nova odabirom opcije Add Component [2]. Neke od važnijih svojstva koje možemo dodati objektu su Rigidbody, razni Collider-i, Animator i razne skripte pomoću kojih definiramo ponašanje objekta. Rigidbody je komponenta koja objekt stavlja pod kontrolu Unity Physics Engine-a. Navedena komponenta objektu automatski, bez ikakvog dodavanja koda, dodjeljuje gravitacijska svojstva i svojstva reagiranja kod sudara s dolaznim objektom [3].

Postoji nekoliko vrsta Collider-a, koji obavljaju svi jedan zadatak, a to je da 3D objekt učine fizički interaktivnim, tj. omogućuju da ostali objekti ne mogu proći kroz njega. Animator-i su komponente koje upravljaju animacijama naših objekta te pomoću raznih skripti pokreću zadane animacije nad objektima.

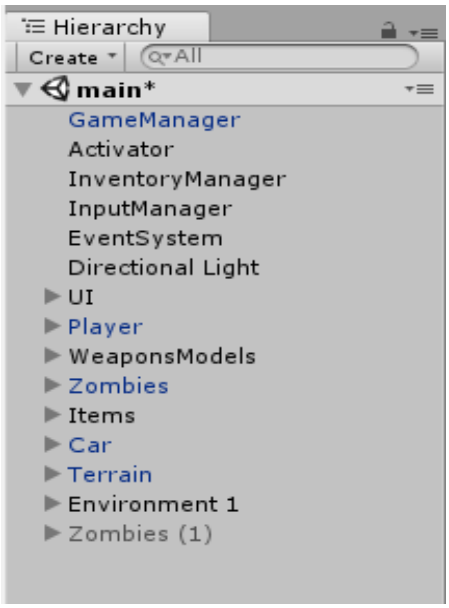
Posljednja vrsta važnijih komponenti su skripte koje sami napišemo u odabranom programskom jeziku koje objektu daju određene osobine te opisuju sam objekt. Programiranje svojstava objekta je moguće u dva programska jezika, a to su C# i Javascript. Igrica o kojoj rad govori pisana je u C# programskom jeziku. Unity omogućuje i kombinaciju dvaju navedenih programskih jezika, ali se ispostavilo da je C# puno moćniji i jednostavniji jezik te se on puno

više koristi. Nove verzije Unity-a dolaze sa ugrađenim Visual Studio Community verzijom koja je besplatna te omogućuje pisanje skripti za Unity.

2.2. Hierarchy

Hierarchy je prozor koji nam pokazuje koji se sve objekti nalaze u našoj sceni igre, tj. predstavlja hijerarhijski prikaz objekata koje smo dodali u samu igru. Hierarchy prozor nam omogućuje da stavimo gotove 3D modele u scenu igre ili da ih sami izradimo pomoću nekih objekata koje nam sam Unity nudi, kao što su modeli kocke, krugova, prazni modeli [4]…

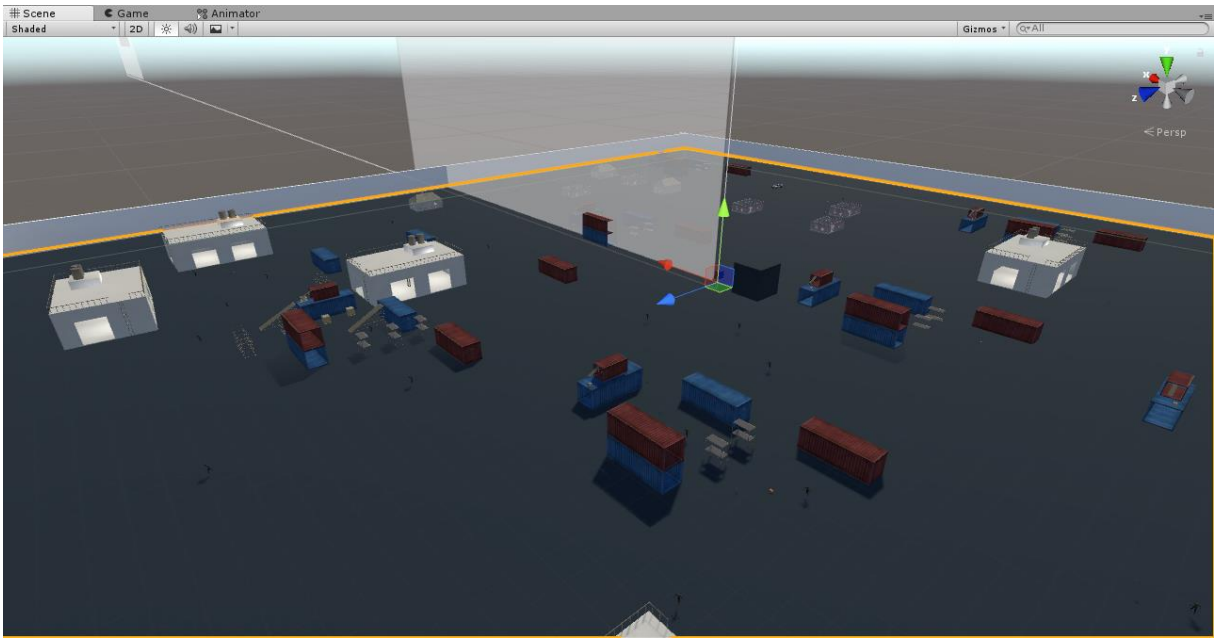
Prazni modeli koje možemo dodati u Hierarchy prozor su vrlo korisni, oni predstavljaju folder u kojeg možemo grupirati sve objekte koji nam, kao u ovom primjeru opisuju teren na kojem se scena igre odvija. Kao što je i vidljivo na slici 3, prazni objekti nam još mogu poslužiti kao neki upravljači. U ovom slučaju su to: GameManager, InventoryManager, Activator, InputManager. Upravljači takve vrste su posebni jer su nevidljivi u samoj igri, ali sadrže skripte koje upravljaju igrom. Pošto se navedeni prazni objekti nalaze u sceni igre, skripte se bez problema izvršavaju bez da mi to vidimo u samoj igri.



Slika 3. Hierarchy

2.3. Scene View

Scene View je prozor koji nam omogućuje da objekte koje smo stavili u Hierarchy pozicioniramo po želji u prostor koji predstavlja scenu naše igre. Ujedno nam i prikazuje kako će naša igra izgledati. Scene View nam omogućuje da iz različitih perspektiva promatramo igru, kako bi smo što je najbolje moguće podesili objekte na sceni. Uz pogled iz raznih perspektiva, omogućuje još i 2D pogled na objekte [5]. Na slici 4. je prikazan Scene View igre Zombi Apokalipsa čiji razvoj se predstavlja.



Slika 4. Scene View

2.4. Game View

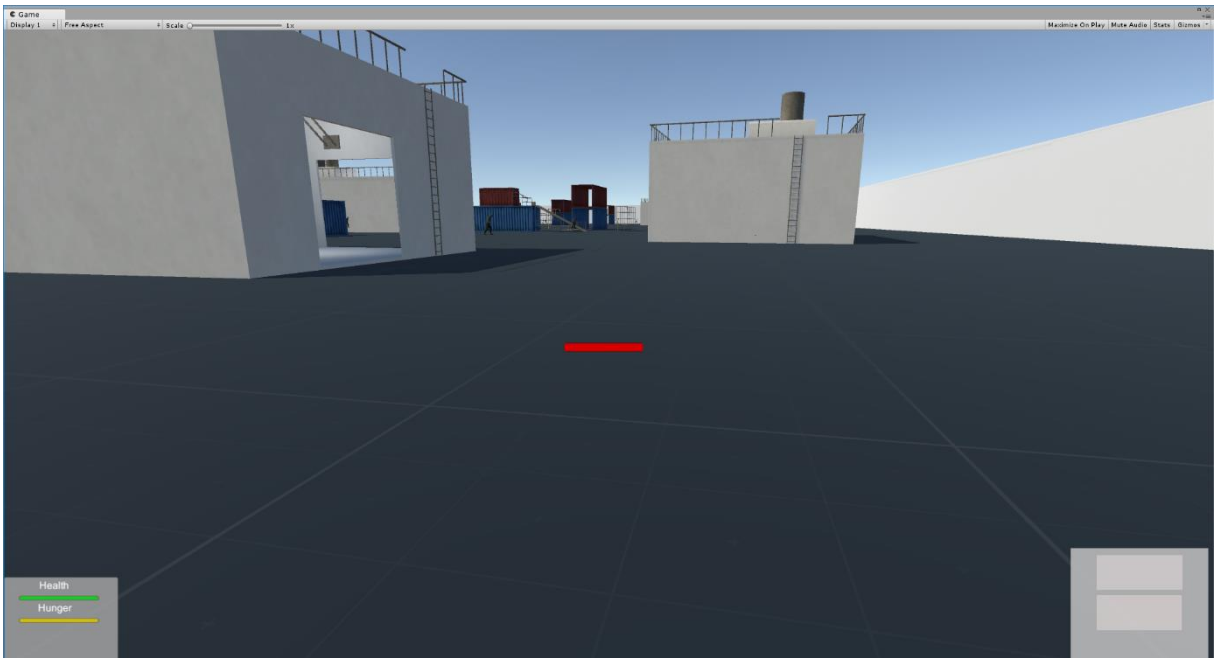
Game View je prozor koji pokazuje što će kamera prikazati, tj. što će igrač igre vidjeti kada pokrene igru. Važna komponenta Game View-a je kamera. Uvijek mora postojati najmanje jedna glavna kamera, moguće je imati više kamera u igri, ali one ne mogu biti aktivne u isto vrijeme, tj. moramo pomoću skripte isključiti prijašnju kameru ako želimo uključiti novu.

Kada pritisnimo tipku Play, pokrene se igra u Game View-u pa ju onda možemo testirati. Navedeni prozor nam ujedno nudi i nekoliko korisnih opcija, kao što su:

- Pause - opcija koja pauzira igru te možemo u miru promatrati parametre u inspektoru za svaki objekt te proučiti kako se oni u određenoj situaciji ponašaju;
- Stats - opcija koja nam pokazuje koliko naša igra koristi računalnih resursa;

- Gizmos - opcija koja nam omogućava da tijekom testiranja igre uključujemo ili isključujemo neke od efekata ili objekta;
- Aspect - opcija koja nam omogućava da tijekom igranja igre možemo vidjeti kako će ona izgledati na različitim rezolucijama monitora [6].

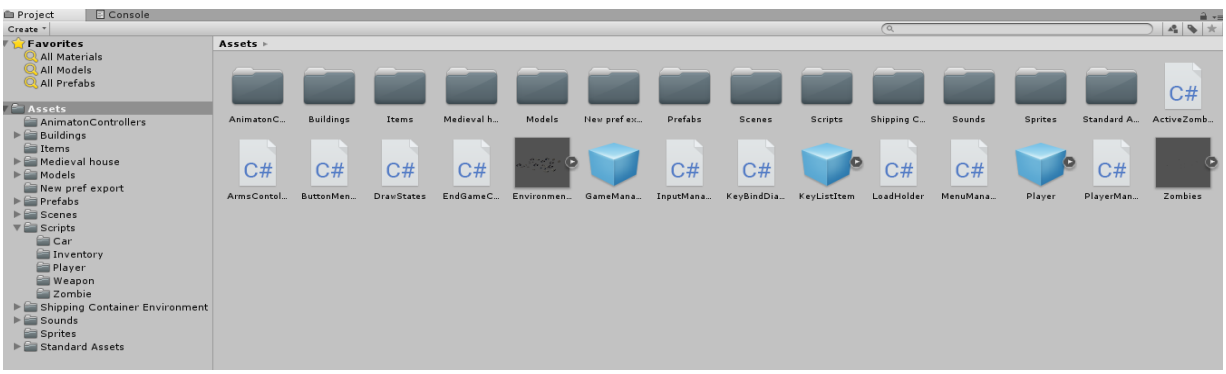
Na slici 5. možemo vidjeti kao u Game View izgleda scena igre Zombi Apokalipsa.



Slika 5. Game View

2.5. Assets i Console

Assets je glavni direktorij igre u kojem se nalaze svi objekti, skripte, zvukovi, animacije. Kao što je vidljivo na slici 6. Assets prozor omogućuje grupiranje komponenata u posebne direktorije kako bi ih lakše pronašli.

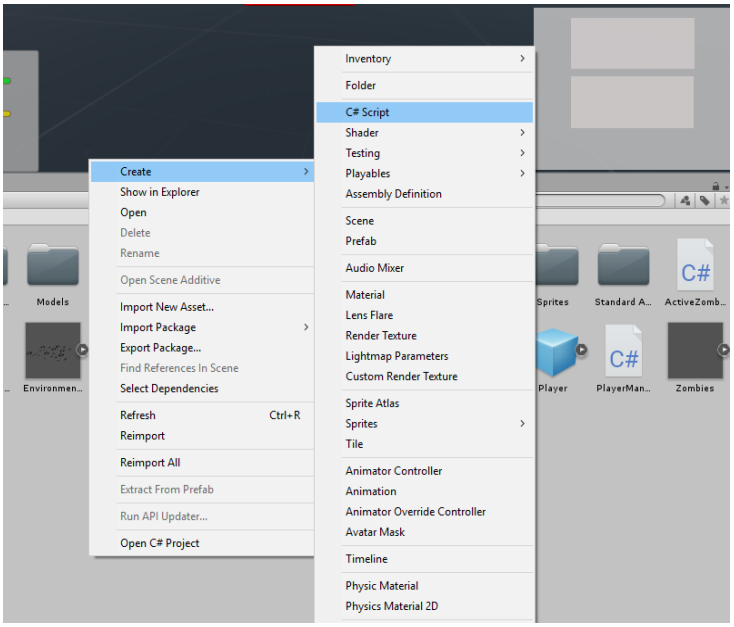


Slika 6. Assets

Console je prozor koji se otvara tijekom pokretanja igre. Prozor nam služi kako bi smo vidjeli razne greško i upozorenja koja se javljaju tijekom razvoja igre.

3. Skripte

Za izradu skripti igre koristi se C# programski jezik. Postoje dva načina kreiranja skripti u Unity-u. Prva je ta da desnim klikom u Assets prozoru odaberemo opciju Create C# Script, kao što je vidljivo na slici 7. Te nakon kreiranje, skriptu jednostavno Drag and Drop metodom stavimo u inspektor objekta za koji je ona namijenjena. Drugi način kreiranja skripti je nešto brzi: u inspektoru željenog objekta, odaberemo opciju Add Components.

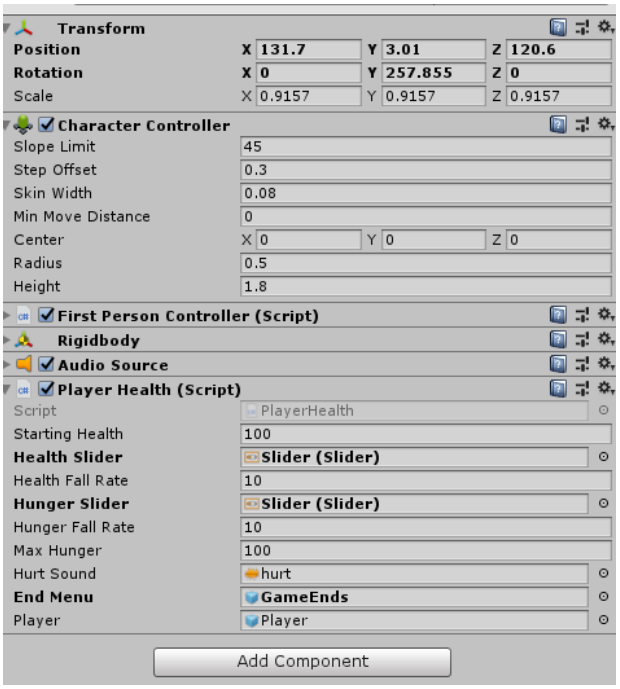


Slika 7. Kreiranje skripte

Kada u Visual Studio-u otvorimo novo kreiranu skriptu vidimo da je Unity već kreirao Start i Update metode.

U Start() metodi se inicijaliziraju razne komponente koju su definirane i postavljaju se početne vrijednosti varijabli, a sama metoda se pokreće tijekom aktivacije objekta koji sadrži skriptu. Update je jedna od najbitnijih metoda u Unity-u koja provjerava da li je došlo do neke promjene u vrijednostima varijabli. Najčešće u nju stavljamo metode i varijable koje nam služe za, npr. praćenja stanja igrača, praćenje interakcije između objekata kao i praćenje unosa preko vanjskih perifernih jedinica kao što je miš i tipkovnica. Tako ćemo u Update() metodi igre Zombi Apokalipsa provjeravati da li je korisnik pritisnuo tipku za pucanje ili za kretanje.

Kako bi smo upravljali objektima preko skripti potrebno je u Editoru definirati objekte i varijable koje utječu na ponašanje objekta. Ako su varijable i objekti zadani kao public, oni će ujedno biti i vidljivi u inspektoru objekta koji sadrži skriptu. Modifikator public nam omogućuje da tijekom testiranja igre mijenjamo varijablu te tako vidimo kako će se objekt ponašati ako promijenimo vrijednost varijable. Ako je varijabla tipa GameObject to nam omogućuje da skripta može dohvatiti metode i varijable od drugog objekta. Pa tako jednostavno u inspektor Drag and Drop metodom premjestimo objekt do čijih skripti želimo pristupiti. Na slici 8 vidimo primjer prikazivanja javnih varijabli u inspektoru. U ovom slučaju skripta Player Health sadrži pet varijabli tipa GameObject koje smo Drag and Drop metodom dohvatili, a to su: Health i Hunger Slider, Hurt Sound, End Menu i Player. Slideri su obični Slide Bar -ovi te ih mi u skripti aktiviramo i mijenjamo im vrijednost, što nam omogućuje vizualan prikaz zdravlja igraća.



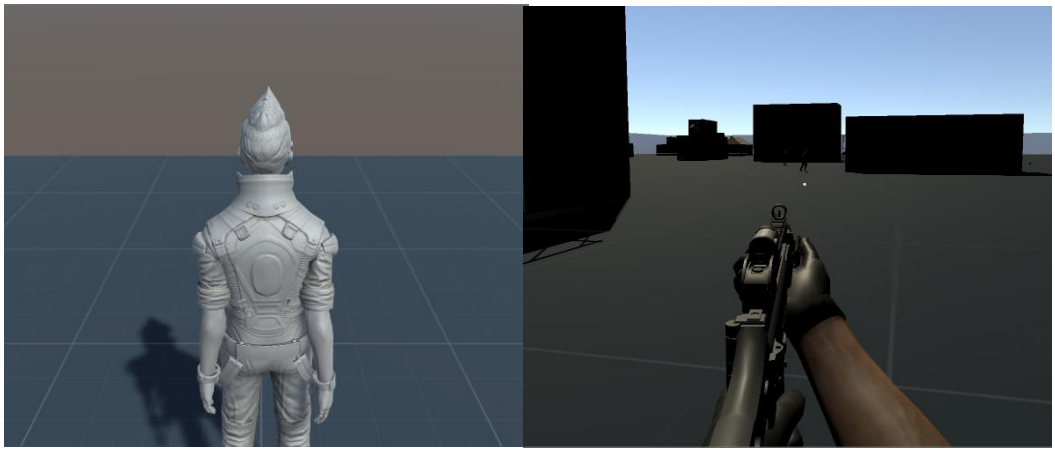
Slika 8. Javne varijable

4. Igra Zombie-Apocalypse

U nastavku će biti opisana sama igra, tj. sami dijelovi od kojih se igra sastoji.

4.1. Model igrača

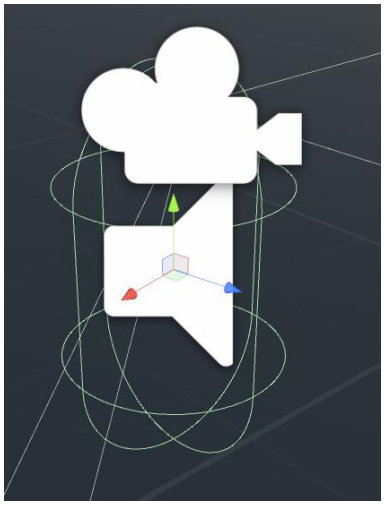
Model igrača se sastoji od nekoliko dijelova. Kao što su: objekt First Person Character, kamere i oružja koje igrač može koristiti. First Person Character predstavlja pogled iz prvog lica. Uz navedeni objekt, u Unity-u postoji također objekt koji se koristi ako želimo kreirati igru u trećem licu, usporedbu navedenih pogleda na igru možemo vidjeti na slici 9.



Slika 9. Pogledi na igru

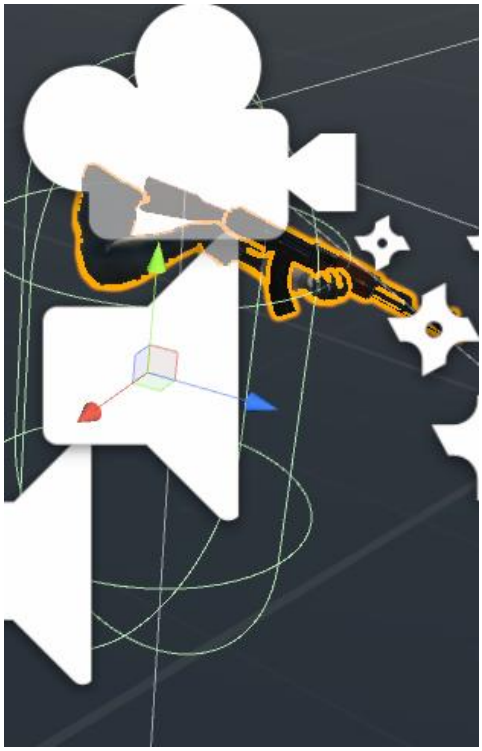
4.1.1. Kretanje igrača

Postoje dvije mogućnosti realizacije kretanja objekta. Ako se radi o nekom objektu koji nema ljudske osobine tada moramo sami kreirati skriptu koja omogućuje kretanje. Pošto se u ovom slučaju radi o modelu koji treba imati osobine čovjeka, koristi se već naveden First Person Character objekt, koji se sastoji od nevidljivog objekta Character Contoller, kamere i skripte koju Unity sam generira te omogućuje da se Character Contoller miče u sve pravce. Tako da neke stvari ne moramo programirati. Na slici 10 vidimo First Person Character u Scene View-u.

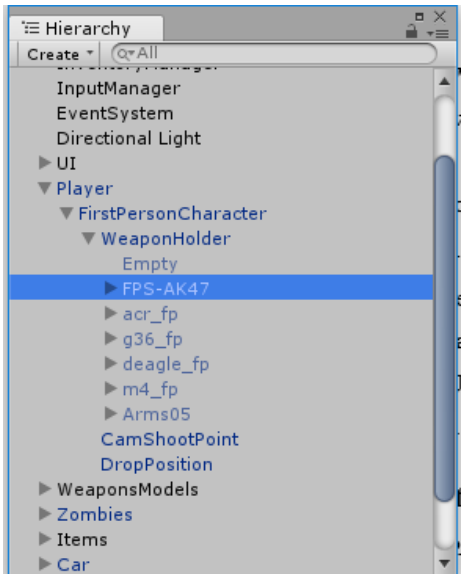


Slika 10. Model igrača

Kako bi smo dobili efekt da se stvarno radi u ljudskom biću, ispred kamere koja nam simulira oči, stavimo model ruku i oružja, kao što je vidljivo na slici 11. Kako bi se model ruku i oružja micao zajedno s pokretima miša, moramo modele staviti kao djecu od kamere. To učinimo tako da u Hierarchy-u željene modele stavimo kao poddirektorij direktorija kamere. Kao što je vidljivo na slici 12.



Slika 11. Model oružja



Slika 12. Poddirektorij oružja

Vidimo da se u poddirektoriju kamere, tj. FirstPersonCharacter modela nalaze sva oružja koja se mogu koristiti u igri te se one aktiviraju ovisno o tome da li je igrač pokupio oružje sa poda.

Skripta FirstPersonCharacter koja nam omogućuje kretanje igrača u igri je vrlo kompleksna i opširna. Pa tako preko inspektora možemo mijenjati razna svojstva, kao što su: osobine kretanja, skakanja, promjena svojstva miša za okretanje kamere, te ostala svojstva kao

što su udaljenost objekta od poda, brzina usporavanja kada prestanemo trčati itd. Na slici 13 vidimo navedenu skriptu u inspektoru igrača.

Slika 13. Skripta kretanja

4.1.2. Zdravlje igrača

Skripta koja upravlja zdravljem igrača, kao i s varijablama koje simuliraju glad igrača, naziva se PlayerHealth. U navedenoj skripti definirana je varijabla startingHealth koja je tipa int i koja definira koliko će zdravlja igrač imati tijekom pokretanja igre. Kako bi se vizualno prokazala razina zdravlja igrača koristi se Unity-ev objekt zvan Slider, kojeg dodamo u Hierarchy, desnim klikom te pod opcijom UI. Kao što je vidljivo u slijedećem kodu PlayerHealth skripte definiramo objekt tipa Slider naziva healthSlider te mu početnu vrijednost stavimo vrijednost varijable startingHealth.

Kada igrača napadne neprijatelj, tj. Zombie, skida se vrijednost Slider-a za neku određenu vrijednost koja je definirana u skripti koja upravlja napadom Zombie-a. Metoda koja upravlja zdravljem je TakeDemage(int _amount). Metoda ima jedan argument _amount, za čiju vrijednost se oduzima vrijednost Slider-a koji je odgovoran za zdravlje. Metoda je javna, tj. public što znači da je ostale skripte mogu pozivati. U ovom slučaju metoda TakeDemage(int _amount) je pozvana od strane Zombie-a, kada nas on napada. Kada je metoda aktivna oduzima se zdravlje igrača i ujedno se aktivira zvuk koji upozorava da smo napadnuti. Metoda ujedno

provjera da li je razina našeg zdravlja, tj. da li je vrijednost Slider-a veća od nula. Ako je veća živi smo, ako ne, pokreće se metoda Death() koja je odgovorna za umiranje igrača.

Umiranje igrača je realizirano na način da metoda Death() zaustavlja sve kretnje igrača i zombie-a te aktivira sučelje koje nam omogućuje da pokušamo ponovo ili da izađemo iz igre. Navedena metoda se nalazi u Update() funkciji koja sve svoje metoda i varijable neprestano izvršava.

```
[SerializeField]
private int startingHealth = 100;
public Slider healthSlider;

private void Awake()
{
        healthSlider.maxValue = startingHealth;
        healthSlider.value = startingHealth;
        endMenu.SetActive(false);

        fps = player.GetComponent<FirstPersonController>();
 }
public void TakeDemage(int _amount)
{
        isDemage = true;
        healthSlider.value -= _amount;
        PlaySound();
        if (healthSlider.value <= 0)
         {
             Death();
         }
 }
private void Death()
{
        endMenu.SetActive(true);
        LoadHolder.endGame = true;
        fps.enabled = false;
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
}
private void PlaySound()
{
        audioSource.PlayOneShot(hurtSound);
}
```

4.1.3. Glad igrača

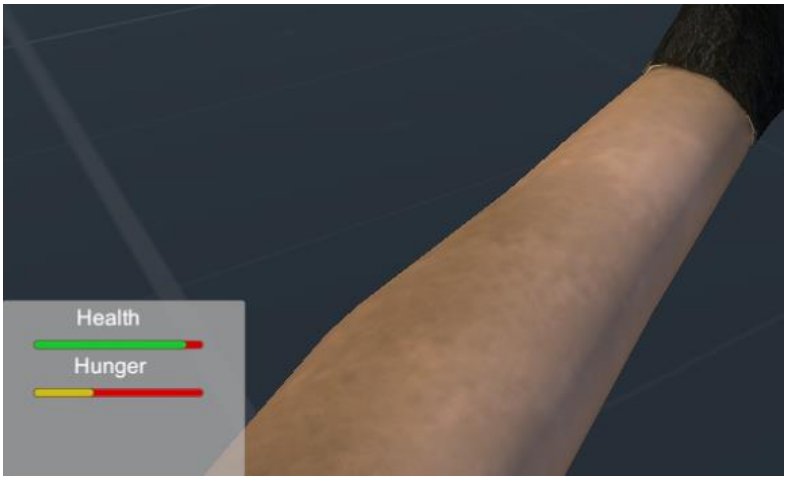
Kao što je već navedeno, skripta PlayerHealth upravlja i s glađu igrača. Glad je realizirana na način da se cijelo vrijeme smanjuje za neku određenu vrijednost te ako nismo pronašli ništa hrane u igri niti je pojeli, vrijednost koja simulira glad će pasti na nula te će se tada zdravlje početi smanjivati sve dok je vrijednost gladi nula.

Glad je, kao i zdravlje, vizualno realizirano preko Slider-a naziva hungerSlider. Vrijednost za koju se vrijednost Slider-a smanjuje definirana je pomoću varijable tipa int i naziva se hungerFallRate te ima zadanu vrijednost deset. U slijedećem kodu vidimo metodu skripte PlayerHealth, HungerControll() koja se također poziva u funkciji Update().

```
private int healthFallRate = 10;
public Slider hungerSlider;
[SerializeField]
private int hungerFallRate = 10;
[SerializeField]

private void HungerControll()
{
       if (hungerSlider.value >= 0)
        {
           hungerSlider.value -= Time.deltaTime / hungerFallRate*10;
        }
       if (hungerSlider.value <= 0)
        {
           healthSlider.value -= Time.deltaTime / healthFallRate*10;
        }
       if (healthSlider.value <= 0)
        {
            isDeath = true;
        }
}
```

HungerControll() metoda provjerava vrijednost hungerSlider-a te ako je vrijednost veća od nule ona se umanjuje svake sekunde za vrijednost koja se dobiva na način da se vrijednost koju vraća Time.deltaTime podijeli sa varijablom hungerFallRate koja u ovom slučaju ima vrijednost deset i pomnoži sa deset. Time.deltaTime je funkcija koja vraća neku vrijednost ovisno o broju slika po sekundi našeg računala. Navedena funkcija nam omogućuje da se vrijednost gladi na svim računalima smanjuje za istu vrijednost neovisno o broju slika u sekundi koje je moguće na računalu. Kada bi smo maknuli tu funkciju i metodu HungerControll() bez nje pozvali u Update() funkciji vrijednost gladi bi se puno brže smanjilo ako računalo proizvodi više odnosno sporije ako računalo proizvodi manje slika u sekundi.

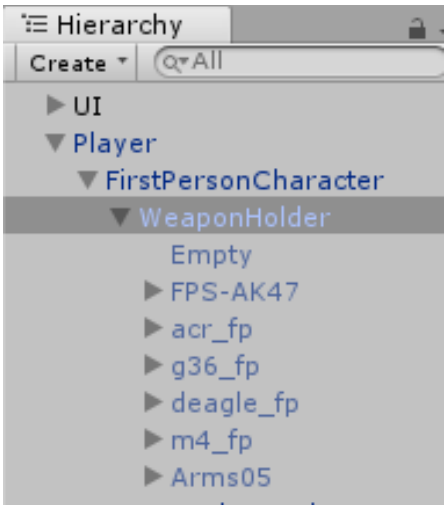
Nadalje, ako je vrijednost hungerSlider-a manje ili jednako nuli, počinje se vrijednost healthSlider-a smanjivat po istom principu. Ova metoda omogućava da igrač može umrijeti i od gladi, ako ne pronalazi hranu. Na slici 14 vidimo kako izgledaju Slider-i koji prikazuju zdravlje igrača i glad.



Slika 14. Prikaz zdravlja

4.2. Oružja

Oružja koja igrač može koristiti su dodana kao poddirektorij u kameri igrača, tako da se ona miču zajedno sa pokretima miša. Poddirektorij se zove WeaponHolder i kao što vidimo na slici 15., sva oružja na početku nisu aktivna. Oružja se aktiviraju tek kada ih igrač pronađe u igri i pokupi.



Slika 15. WeaponHolder

Pravilo u igri je da se može imati samo dvije vrste oružja odjednom, jednu pušku i jedan pištolj ili nož. Skriptom koja upravlja uzimanjem oružja s poda i bacanje neželjenih zove se WeaponManager i nalazi se u inspektoru navedenog poddirektorija. Kao što možemo vidjeti na slijedećoj slici inspektora, skripta sadrži nekoliko važnih elemenata, a to su: polje koje sadrži

dva prazna objekta Weapon In Use i polje veličine od sedam elemenata Weapon In Game. Polje od dva elemenata definira oružja koja je igrač pokupio tijekom igranja i između kojih može birati koje će koristiti. Polje od sedam elemenata Weapon In Game sprema sva oružja koja se nalaze u samoj igri, te se ona aktiviraju ako igrač pronađe ista u igri i želi ih pokupiti.

Kako bi Unity znao koji objekt su puške a koji pištolj ili nož, za svaki objekt smo definirali layer u inspektoru, tako imamo dvije vrste oružja te pomoću navedene skripte možemo raspoznati koliko puška odnosno pištolja imamo. Ako se u polju Weapon In Use nalazi već jedna puška odnosno pištolj pokrenut će se određena metoda za bacanje oružja, tj. zamjenu za novo. U skripti WeaponManager se nalazi nekoliko metoda, ali dvije od njih su posebno važne, a to su: TakeWeapon() i DropWeapon() koje vidimo na slijedećem dijelu koda.

```
private void TakeWeapon()
{
       RaycastHit hit;
       if (Physics.Raycast(shootPoint.position, shootPoint.transform.forward, out hit,
dis, layerMaskWeapon))
        {
           WeaponIndex pre = hit.transform.GetComponent<WeaponIndex>();

           int setElement = pre.setWeapon;

           if (weaponsInUse[1] != weaponInGame[setElement])
            {
                weaponSlot.sprite = pre.weaponImage;
                weaponsInUse[1].SetActive(false);

                if (weaponsInUse[1] != weaponInGame[0])
                {
                    DropWeapon(weaponToDrop);
                    pre.isTaked = false;
                }
                weaponsInUse[1] = weaponInGame[setElement];
                weaponsInUse[0].SetActive(false);
                weaponsInUse[1].SetActive(true);
                Weapon weapon = weaponsInUse[1].transform.GetComponent<Weapon>();
                weaponToDrop =
weaponsInUse[1].transform.GetComponent<WeaponIndex>().setWeapon;
                BulletsController.instance.activeWeapon = weaponToDrop;
                pre.isTaked = true;

                if (setElement == 1 || setElement == 2)
                {
                    weapon.bulletLeft = BulletsController.instance.akScar;
                }
                if (setElement == 3 || setElement == 5)
                {
                    weapon.bulletLeft = BulletsController.instance.b556;
                }
            }
            else
            {
                Weapon weapon = weaponsInUse[1].transform.GetComponent<Weapon>();
                if (setElement == 1 || setElement == 2)
                {
                    BulletsController.instance.akScar += weapon.bulletPerMag;
                }
                if (setElement == 3 || setElement == 5)
                {
                    BulletsController.instance.b556 += weapon.bulletPerMag;
                }
                BulletsController.instance.akScar += weapon.bulletPerMag;
                weapon.bulletLeft += weapon.bulletPerMag;
                pre.isTaked = true;
            }
            BulletsController.instance.slot1 = true;
            BulletsController.instance.slot2 = false;

        }
```

U igri je moguće imati šest oružja i jedan prazni objekt koji simulira same ruke, tj. stanje kada igrač nije pokupio niti jedno oružje. Tako ruke imaju indeks 0 a ostala oružja imaju indekse

od 1 do 5. Na slijedećem primjeru koda vidimo skriptu WeaponIndex koja svim oružjima definira navedeni indeks, ime i sliku oružja Skripta WeaponIndex sadrži još dvije metode koje se varijablu canTake koja se nalazi u skripti WeaponManager, postavlja na vrijednost true ako igrač uđe u radiju metode ili na vrijednost false ako izađe iz radijusa. To su metode OnTriggerEnter() i OnTriggerExit().

U igri postoje dvije vrste pušaka koje mogu koristiti samo određene metke koje igrač pronađe u igri. Isto je i sa pištoljem koji korisni samo njemu namijenjene metke. U navedenoj metodi se prepoznaje o kojoj je vrsti oružja riječ te se iz posebne skripte BulletsController preuzima svoj stanje metaka i osobina određene puške, o toj skriti nešto više kasnije. Uzimanje oružja sa poda radi na slijedeći način. Kao što je vidljivo u prvom if uvjetu, definira se Unity funkcija RaycastHit().

RaycastHit() funkcija radi tako da očita objekt te nam vraća neke informacije o njemu. U ovom slučaju funkcija od objekta koji se nalazi u njezinom fokusu čita prije navedeni Layer, ako on ima oznaku puška tada se iz objekta očitaju podaci iz skripte WeaponIndex koju svaki objekt oružja ima na sebi. Tako skripta WeaponManager automatski zna o kojoj se vrsti oružja radi i koje su osobine istih. U drugom if upitu se provjerava da li je polje Weapon In Use koje je rezervirano za pušku odnosno pištolj popunjeno. Ako je puno, oružje koje se nalazi na tom mjestu se baca na pod. Metoda koja je odgovorna za bacanje oružja zove se DropWeapon() i njezin kod je vidljiv u slijedećem primjeru.

```
void DropWeapon(int index)
{
        if (index == 0) return;

        for (int i = 0; i < worldModels.Length; i++)
        {
           if (i == index)
        {
Rigidbody drop = Instantiate(worldModels[i], dropPosition.transform.position,
dropPosition.transform.rotation) as Rigidbody;
drop.AddRelativeForce(0, 250, UnityEngine.Random.Range(100, 200));
drop.AddTorque(-transform.up * 40);
             }
         }
}
```

Kao što vidimo na primjeru koda, metoda aktivira Rigidboy određenog oružja koji fizički opisuje model i daje mu težinu te se on baca na neku udaljenost od same pozicije igrača. Model se aktivira na našoj poziciji te se baca u dalj, a model ruke s oružjem koje smo koristili

se deaktivira te postaje nevidljiv za igrača. Tako dobijemo prizor kao da igrač zamjenjuje staro oružje s novim. Na slici 16 vidimo kako je igrač pokupio novo oružje i kako se u donjem lijevom kutu učitava slika oružja iz WeaponIndex skripte.

Slika 16. Prikaz oružja

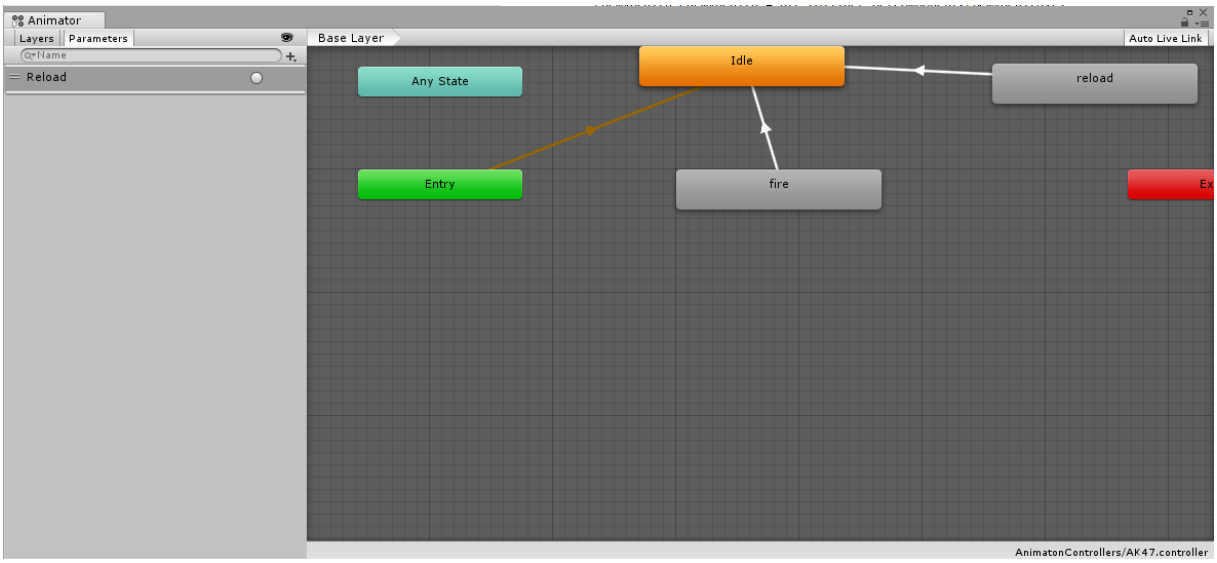
4.2.1 Pucanje

Pucanje unutar igre je realizirano na sličan način kao i uzimanje oružja sa poda, a to je preko RaycastHit() funkcije. Navedena funkcija iz jedne točke koja se nalazi na sredini ekrana čita na određenu duljinu Collider koji se nalazi na objektima. Svi zombiji imaju na sebi Collider koji ima efekt da ostale stvari ne mogu prolazit kroz njih. Također svaki neprijatelj sadrži na sebi skriptu EnemyHealth koja upravlja njegovim zdravljem. Kada RaycastHit() funkcija pogodi neki objekt i ako taj objekt sadrži navedenu skriptu, Weapon skripta koja upravlja svim oružjima zna da se radi o neprijatelju te iz dobivene skripte očita zdravlje neprijatelja. Ako je zdravlje veće od nula, pokreće se metoda TakeDemage() koja se nalazi na svakom Zombie-u u skripti koja upravlja njegovim zdravljem. Zdravlje se oduzima ovisno o kojem je oružju riječ, neka oružja uzimaju više a neka manje života neprijatelju. Pošto funkcija koja upravlja pucanjem oružja, Fire() uzima i poziciju pogođenog objekta, na toj poziciji gdje je RaycastHit() funkcija uzela uzorke sa objekta se aktiviraju objekti pod nazivom hitParticelsEffekt pa tako imamo dojam da je projektil udario u objekt. Ujedno se i u funkciji za pucanje pokreću i definirane metode za prikazivanje plamena na oružju, prikazivanje animacija pucanja i metoda za sviranje zvuka pucanja.

```
private void Fire()
{
       if (fireTimer < fireRate || currentBullets <= 0 || isReloding ||
LoadHolder.endGame)
       {
           return;
       }
       RaycastHit hit;
       //recoil
       Vector3 shootDirection = shootPoint.transform.forward;
       shootDirection.x += Random.Range(-spreadFactor, spreadFactor);
       shootDirection.y += Random.Range(-spreadFactor, spreadFactor);
       if (Physics.Raycast(shootPoint.position, shootDirection, out hit, range))
       {
           GameObject hitParticelsEffect = Instantiate(hitParicles, hit.point,
Quaternion.FromToRotation(Vector3.up, hit.normal));
EnemyHealth enemyHealth = hit.collider.GetComponent<EnemyHealth>();
       if (enemyHealth != null)
       {
       enemyHealth.TakeDemage(demage);
       }
       Destroy(hitParticelsEffect, 1f);
       }
       anim.CrossFadeInFixedTime("fire", 0.05f);
       muzzleFlash.Play();
       PlayShootSound();
       currentBullets--;
       ShowAmmo();
       fireTimer = 0.0f;
 }
```

4.2.2. Animacije oružja

Modeli ruke i oružja koji se koriste u samoj igri sadrže već gotove animacije, ma da ih je moguće i kreirati u samom Unity-u. Kako bi smo mogli koristiti animacije u igri prvo trebamo kreirati Animator Controller kojeg vidimo na slici 17.



Slika 17. Animator Controller

Animator Controller omogućuje da upravljamo našim animacijama. Kao što je vidljivo na slici 16, jednostavno Drag and Drop metodom povučemo animacije u contoller. Definiramo početnu animaciju koja je u ovom slučaju označena narančastom bojom i koja je podešena da se uvijek ispočetka izvršava ako druge animacije nisu aktivirane. U Animator Controller-u možemo također definirati i parametre koji nam pomažu tijekom izvođenja animacija također možemo i mijenjati sama svojstva animacije. U ovom slučaju nisu korištene dodatne mogućnosti animatora nego su animacije za pucanje i za punjenje oružja povećate povratnom vezom s početnom animacijom te ih mi u kodu aktiviramo po potrebi. Animacije se također pokreću u skripti koja upravlja oružjem Weapon, uz navedenu skriptu za pokretanje animacija koristimo i skriptu Reload koja sadrži dvije funkcije koje provjeravaju stanje tijekom osvježavanja igre, tj. ako želimo napuniti oružje novim metcima, skripta nam ne dopusta da pucamo ako animacija za punjenje nije skroz izvršena i ako metci nisu prikazani na ekranu. Nadalje su prikazane funkcije skripte Weapon koje upravljaju animacijama oružja.

```
public void Reload()
{
       if (bulletLeft <= 0) return;
       int bulletsToLoad = bulletPerMag - currentBullets;
       int bulletsToDeduct = (bulletLeft >= bulletsToLoad) ? bulletsToLoad : bulletLeft;
       bulletLeft -= bulletsToDeduct;
       BulletsALLControll(bulletsToDeduct);
       currentBullets += bulletsToDeduct;
       ShowAmmo();
 }
 private void DoReload()
 {
       AnimatorStateInfo info = anim.GetCurrentAnimatorStateInfo(0);
       if (isReloding) return;
       PlayReloadSound();
       anim.CrossFadeInFixedTime("reload", 0.01f);
 }
```

Kao što je prikazano na slijedećem isječku koda, funkcija Reload() provjerava da li imamo dovoljno metaka kod sebe kako bi smo napunili oružje. Ako je broj metaka koje imamo nula, izvršava se naredba return te se izlazi iz funkcije, a ako imamo metaka računa koliko se metaka treba oduzeti od ukupne količine da se napuni cijeli kapacitet oružja, koji je u većini slučaja trideset metaka. Naknadno se od ukupne količine oduzme broj metaka koji je bio potreban za cijeli kapacitet oružja.

Metoda DoReload() nam omogućuje da se pokrene animacija za punjenje oružja zajedno sa zadanim zvukom. Naredba za pokretanje animacija je anim.CrossFadeInFixedTime(„reload“, 0.01f) gdje anim predstavlja Animator Controller kojega u inspektoru modela oružja samo povučemo u skriptu. Funkcija CrossFadeInFixedTime() koja nam omogućava da pokrenemo

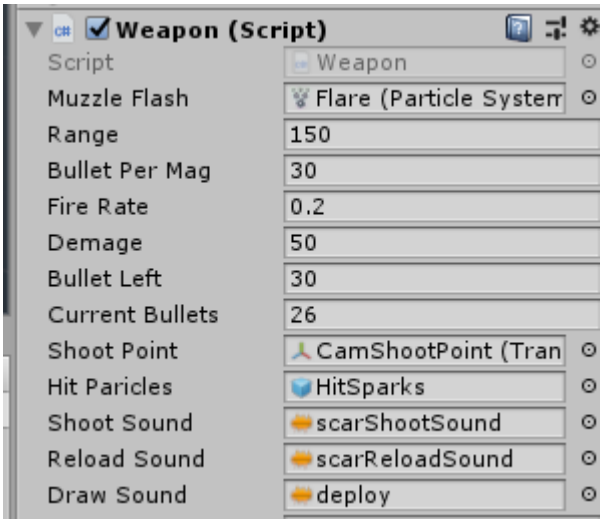
bilo koju animaciju nakon određenog vremena. Pomoću navedene funkcije se i kod svih modela oružja pokreće i animacija za pucanje u Fire () metodi.

4.2.3. Zvukovi oružja

Svi zvukovi koji se koriste u igri a nisu dobiveni sa modelima su besplatno preuzeti sa interneta i prerađeni u programu Audacity [15]. Kako bi smo zvukove pokretali u Unity-u trebamo dodati na model komponentu Audio Source koja nam omogućuje da objekt proizvodi zvuk. Tada u skripti trebamo definirati navedenu komponentu zajedno s zvukovima koji su tipa AudioClip koje pomoću funkcija za pokretanje zvuka pokrenemo. Slijedeći primjer koda koji se nalazi u skripti Weapon pokazuje pokretanje zvukova.

```
private AudioSource audioSource;
[SerializeField]
private AudioClip shootSound;
[SerializeField]
private AudioClip reloadSound;
public AudioClip drawSound;

private void PlayShootSound()
{
       audioSource.PlayOneShot(shootSound);

}
private void PlayReloadSound()
{
       audioSource.PlayOneShot(reloadSound);
}

private void PlayDrawSound()
{
       audioSource.PlayOneShot(drawSound);
}
```

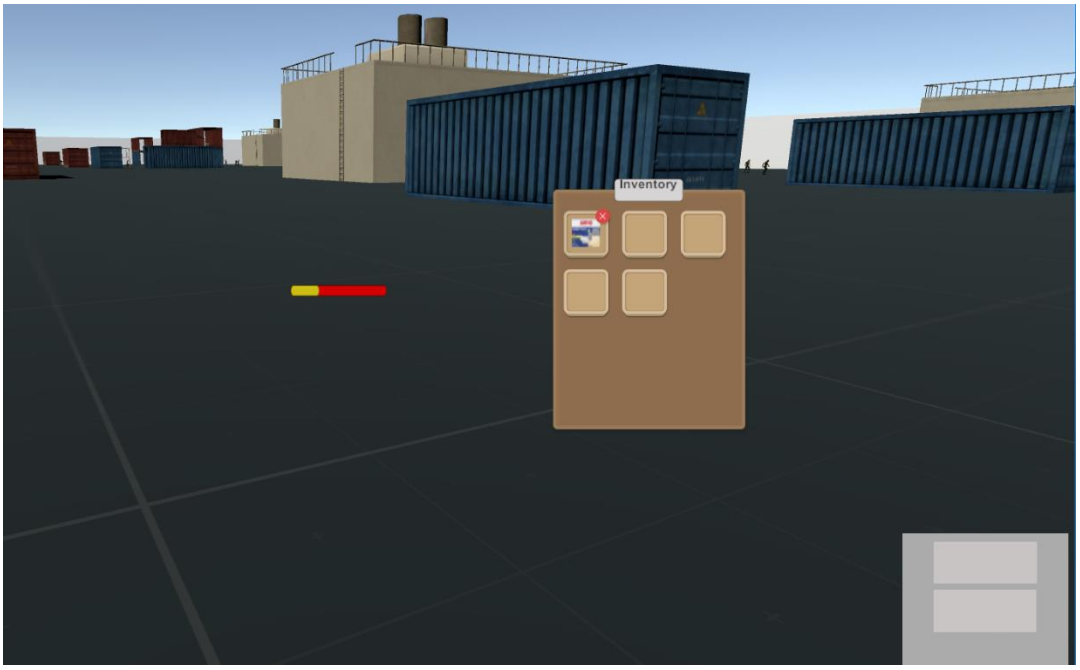
Na navedenom primjeru koda vidimo da pomoću PlayOneShot() funkcije koja je dio Audio Source klase, pokrećemo određene zvukove za pucanje, punjenje oružja i za zvuk koji se pokreće kad pokupimo oružje. Jedni što je još potrebno je da u inspektor na mjesta koja označavaju objekte zvuka dodamo naše zvukove koji će se pokretati, kao što je vidljivo na slici 18.



Slika 18. Weapon skripta

4.3.1. Inventory

Inventory ima ulogu torbe u koju možemo spremiti stvari koje smo pokupili tijekom igre. U Inventory možemo staviti metke i sve ostale objekte koji nam služe kako bi smo povećali zdravlje ili uklonili glad. Kao što možemo vidjeti na slici 19 u naš Inventory možemo pohraniti 5 stvari. Kada želimo određenu stvar iskoristiti jednostavno otvorimo Inventory s tipkom Tab ili nekom drugom koju namjestimo da obavlja tu zadaću te klikom miša iskoristimo objekt.



Slika 19. Inventory

Nakon što smo kliknuli na objekt kojeg želimo iskoristiti javlja se Slider koji se ovisno o vrsti objekta puni brže ili sporije. Kada se napuni izvrši se efekt pojedinog objekta.

Inventory radi tako da iz liste objekta čita sve objekte te njihove slike prikazuje u sučelju. Kako bi svaki objekt bio definiran kao objekt koji se može iskoristiti koristi se skripta Items. Kao što vidimo iz koda same skripte ona je vrlo jednostavna ali ispunjava važnu ulogu.

```
[CreateAssetMenu(fileName = "New Item", menuName = "Inventory/Item")]
public class Item : ScriptableObject {

    new public string name = "New Item";
    public Sprite icon = null;

    public virtual void Use()
    {

    }
    public void RemoveFromInventor()
    {

        Inventory.instance.Remove(this);
    }
}
```

Navedena skripta se koristi kao predložak, tj. izradimo komponentu koja ima osobine dobivene iz navedene skripte, a to su ime objekta i njegova ikona. To je vrlo bitno jer tako možemo napraviti više predloška sa specifičnim nazivima koji označavaju koja vrsta je koji objekt i sukladno tome će obavljati samo tom imenu definirane operacije. Tako u igri imamo objekte koji sadrže predloške različitih metaka, objekta koji smanjuju glad igrača kao što su: napitci, banane, kruh. Objekti koji nam služe kako bi smo povećali zdravlje igrača, bandaže i prve pomoći.

Skripta koja je usko povezana sa navedenom je skripta Equipment. Skripta Equipment nasljeđuje varijable i metode skripte Item te definira osobine koje se mijenjaju ako iskoristimo odabrani objekt. Na slijedećem primjeru koda vidimo da skripta ujedno sadrži i metodu Use koja iz skripte EquipmentManager uzima operacije koje će se izvesti ovisno o vrsti objekta.

Skripta još sadrži listu enum naziva EquipmentSlots. Enumeration ili enum nam omogućava da definiramo kolekciju povezanih konstanti [7]. Pomoću nje definiramo kojoj vrsti pojedini objekt pripada te u inspektoru odaberemo kojoj vrsti konstanti objekt pripada.

```
[CreateAssetMenu(fileName ="new Equpment", menuName ="Inventory/Equpment")]
public class Equipment : Item {
    public EquipmentSlots equipSlot;

    [Header("Equipment")]
    public int medModifier;
    public int hungerModifer;
    public int bulletsModifer;

    public override void Use()
    {
        base.Use();
        EquipmentManager.instance.Equip(this);
        RemoveFromInventor();

    }
}

public enum EquipmentSlots { Med, EnergyDrink, Bullets, DeagleBullets, Pepsi, Zavoji,
Banana, Kruh, AkScar,B556 }
```

Skripta koja pokreće iskorištavanje objekta iz Inventory-a i definira vrijednosti osobina objekta naziva se EquipmentManager. Skripta sadrži metodu Equip koja razlikuje koju vrstu objekta smo kliknuli. Uz navedenu metodu skripta sadrži još i metodu UseItem koja, kao što je vidljivo na slijedećem isječku koda, se konstanto vrti u Update() funkciji te ako se pokrene iskorištavanje objekta povećava vrijednosti Slider-a koji nam pokazuju glad i zdravlje.

```
private void Update()
    {
        move = player.GetComponent<FirstPersonController>().walk;
        if (klik && !move)
        {
            useObject.SetActive(true);
            useSlider.value += (Time.deltaTime + useChangeRateValue);
            if (useSlider.value == useMaxValue)
            {
                UseItem(newItem);
                useObject.SetActive(false);
                useSlider.value = useStartValue;
                klik = false;
            }
        }
        else
        {
            useObject.SetActive(false);
            klik = false;
            return;
        }
    }

public void Equip(Equipment _newItem)
    {
        newItem = new Equipment();
        if (_newItem.equipSlot == EquipmentSlots.EnergyDrink)
        {
            useChangeRateValue = 1;
            newItem = _newItem;
            klik = true;
            PlaySound(drinks);
        }
        if (_newItem.equipSlot == EquipmentSlots.Pepsi)
        {
            newItem = _newItem;
            useChangeRateValue = 1;
            klik = true;
            PlaySound(drinks);
        }
        if (_newItem.equipSlot == EquipmentSlots.Med)
        {
            useChangeRateValue = 0.1f;
            newItem = _newItem;
            klik = true;
            PlaySound(med);
        }

  private void UseItem(Equipment _newItem)
    {
       playerHealth.hungerSlider.value += _newItem.hungerModifer;
       playerHealth.healthSlider.value += _newItem.medModifier;
    }
    private void PlaySound(AudioClip _sound)
    {
        audioSource.PlayOneShot(_sound);
    }
```

U funkciji Update() se provjerava da li je uslijedio klik na objekt, tj. iskorištavanje. Ujedno se i provjerava da li igrač stoji na mjestu i da li je vrijeme prošlo koje je potrebno da se

aktiviraju osobine objekta. Ako se igrač u procesu iskorištavanja objekta pomakne, objekt se poništava i ne povećavaju se osobine gladi i zdravlja igrača. Uz navedeno, istovremeno se izvršava funkcija za pokretanje zvuka koji se razlikuju o vrsti objekta koji se iskorištava.

Jedna od bitnijih skripti koja također upravlja radom Inventory-a je skripta Inventory. Skripta sadrži metode koje su zadužene za dodavanje i brisanje objekata iz liste pomoću koje se objekti pokazuju u samom Inventory-u. Tako skripta sadrži metode Add() i Remove() koji su prikazani u slijedećem dijelu koda skripte.

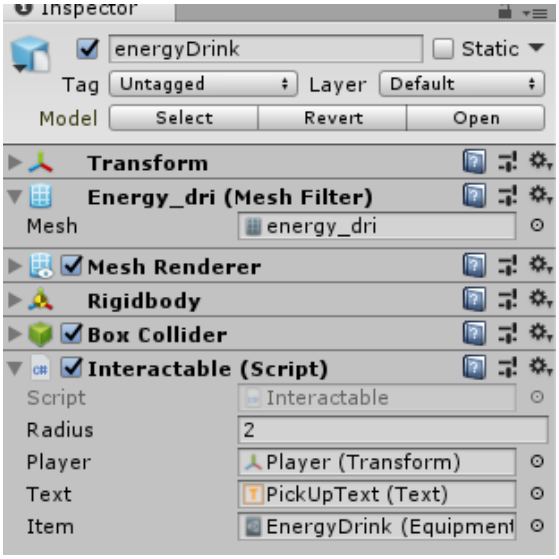
```
public bool Add(Item _item)
    {
        if (items.Count >= space)
        {
            return false;
        }
        items.Add(_item);
        if (onItemChangedCallBack != null)
            onItemChangedCallBack.Invoke();
        return true;

    }

    public void Remove(Item _item)
    {
        items.Remove(_item);
        if (onItemChangedCallBack != null)
            onItemChangedCallBack.Invoke();
    }
```

4.3. 3D objekti

U igri se nalazi nekoliko objekta uz oružja, koja korisnik može igrač kako bi što duže ostao živ.

4.3.1. Energetski napitci

Energetski napici su objekti koji se koriste kako bi se smanjila glad igrača i ujedno povećalo zdravlje za neku vrijednost. Tako imamo dva energetska pića koja se razlikuju po izgledu, ali imaju istu funkcionalnost. Kao što vidimo na slici 20. svaki objekt pa tako i ovaj sadrži skriptu Interactable.



Slika 20. Energy Drink

Navedena skripta nam omogućuje da objekte pokupimo s poda. Vidimo da je definirana varijabla Radius koja nam govori koliko igrač mora biti udaljen od objekta kako bi ga mogao pokupiti. Varijabla Player nam vraća poziciju igrača, kako bi skripta znala da li je igrač na dovoljno maloj udaljenosti kako bi pokupio objekt. Objekt Text je objekt dodan na sredinu ekrana koji je tipa text te ako je igrač unutar zadane udaljenosti, na ekran se ispiše poruka za pokupiti objekt. Varijabla Item nam označava predložak koji opisuje pojedine objekte kao što je bilo opisano u prijašnjem poglavlju. Na sljedećem dijelu koda vidimo kako se u Update() funkciji skripte Interactable provjerava udaljenost od igrača od objekta i da li ima mjesta u Inventory-u. Ako ima mjesta i ako je igrač u zadanoj udaljenosti od objekta, objekt se može pokupiti metodom PickUp(). Ako metoda uspješno spremi u listu objekt, 3D model koji leži na podu se uništava te tako imamo doživljaj kao da je pokupljen.

```
private void Update()
    {
      //  time += Time.deltaTime;
        float distance = Vector3.Distance(player.position, transform.position);

            if (distance <= radius)
            {
                if(!Inventory.instance.torbaPuna)
                       text.text = "Press key <color=#88FF6AFF> << F >> </color> to Use:
       " +      item.name;
                else
                text.text = "Torba je puna!!";

            if (inputManager.GetButtonDown("Use"))
                {
                    PickUp();
                }
            }
            else
            {
                text.text = item.name;
            }
    }

    private void PickUp()
    {
        wasPickedUp = Inventory.instance.Add(item);
        if (wasPickedUp)
        {
            Destroy(gameObject);
        }
```

Na slici 21 vidimo kako energetski napitak izgleda na terenu.

Slika 21. Model napitka

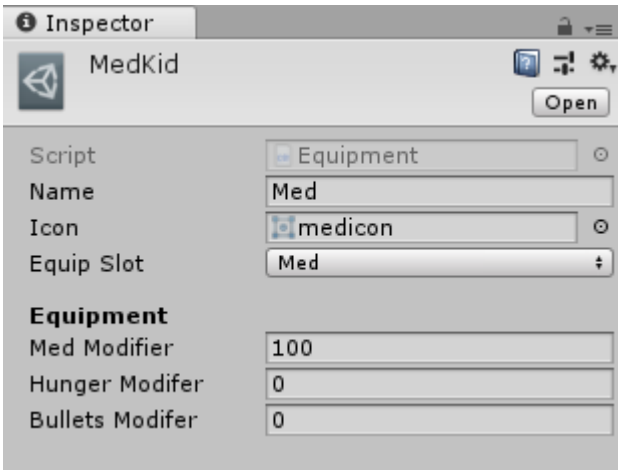
4.3.2. Hrana

U igri imamo dva objekta koji nam smanjuju glad, a to su: banana i kruh. Banana smanjuje glad za trideset i pet posto, a kruh za sto posto. Inspektor navedenih objekata izgleda identično kao i kod energetskog napitka samo je razlika u predlošku koji ga opisuje.

Slika 22. Modeli hrane

4.3.3. Prva pomoć i bandaže

Kao što u igri postoje objekt koji smanjuju glad, tako postoje i oni koji povećavaju zdravlje, ali su puno manje zastupljeni. Navedeni objekti su objekti prve pomoći koji povećavaju zdravlje za sto posto i objekti bandaža koji povećavaju zdravlje za 25 posto. Na slici 23 vidimo primjer predloška koji se stavi na objekte prve pomoći.



Slika 23. Predložak

Iz Inspektora predloška vidimo da smo definirali ime objekta, njegovu ikonu koja će se prikazati u Inventory-u i da je izabrana kolekcija iz liste tipa enum koja opisuje prvu pomoć.

Vidimo da smo u poljima Equipment definirani za koliko će se koja vrijednost kod igraća povećati. Vidimo da će se zdravlje povećati za sto, a pošto je najveća vrijednost koju Slider koji označava zdravlje isto sto znači da će igrač imati uvijek najveću vrijednost zdravlja. Svi navedeni objekti u igri podešavaju se na isti način. Tako da u predlošku definiramo koja svojstva igrača će za koju vrijednost promijeniti.

4.3.4. Metci

U igri imamo nekoliko vrsta oružja, oružja su tako definirana da mogu koristiti samo određene metke. U samoj igri se nalaze tri vrste metaka, jedni za pištolj, a ostali za puške. Nigdje u igri nije definirano koju vrstu metaka koriste puške. Time se otežava igranje same igre, ali se vrsta metaka i pušaka preklapaju kao i u stvarnom životu. Tako će igrači koji su bili u doticaju sa sličim tipovima igre brže shvatiti koje metke koristi koja puška, a oni koji nisu bili u doticaju će morati eksperimentirati. Na slici 24 vidimo 3D modele metaka koji se mogu pronaći u igri.

Slika 24. Modeli metaka

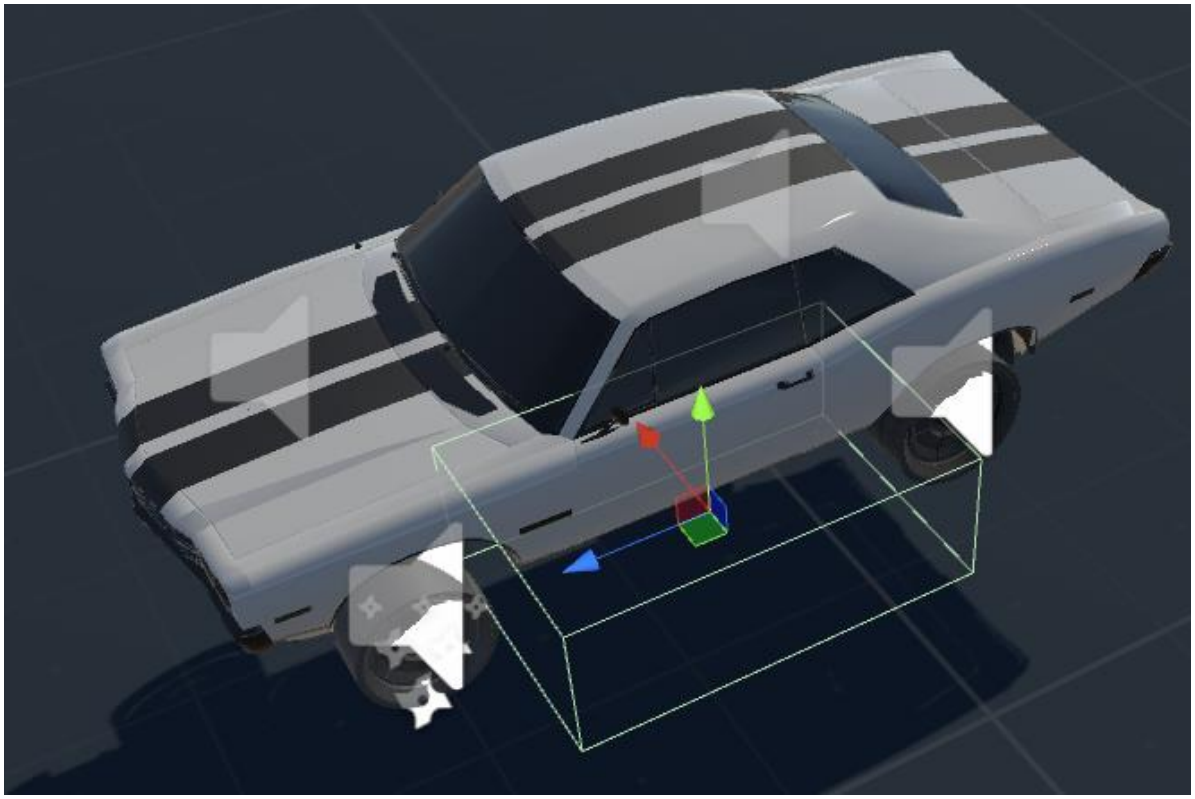
Kako bi određena vrsta oružja mogla koristiti samo njoj namijenjene metke, definirano je u skripti EquipmentManager koji metci pripadaju kojoj vrsti oružja. Na slijedećem kodu vidimo isječak iz metode Equip() koja se nalazi u EquipmentManager-u. Metoda upravlja ne samo vrstama metaka nego i prepoznale o kojoj se vrsti objekta radi, koji je igrač pokupio sa poda.

4.3.5. Auto

Kako bi se povećale mogućnosti da igrač pronađe mjesto gdje je siguran i pobjedi, predviđena je mogućnost korištenja auta. Automobil je samo jedan i ima ograničenu količinu goriva tako da se igrač ne može predugo njime voziti.

Kako Unity sadrži gotovi model automobila iskoristili smo njega za kretanje, ali uz neke bitne promjene. Na zadani model smo stavili 3D model drugog auta te tako omogućili da upravljamo svojim automobilom bez da moramo posebno pisati skripte koje će to omogućiti.

Kao što je na slici 25 automobila vidljivo, dodali smo jedan Box Collider koji nam označava mjesto gdje možemo ući u auto. Kako Box Collider ne bi imao svoju klasičnu funkciju, sprečavanje ostalih objekta da prolaze kroz njega. Označili smo u Inspektor-u da navedena komponenta bude Trigger. Navedena opcija nema klasične karakteristike, nego sada omogućava da objekti prolaze kroz Collider, ali se detektira kada je neki objekt ušao u njega.



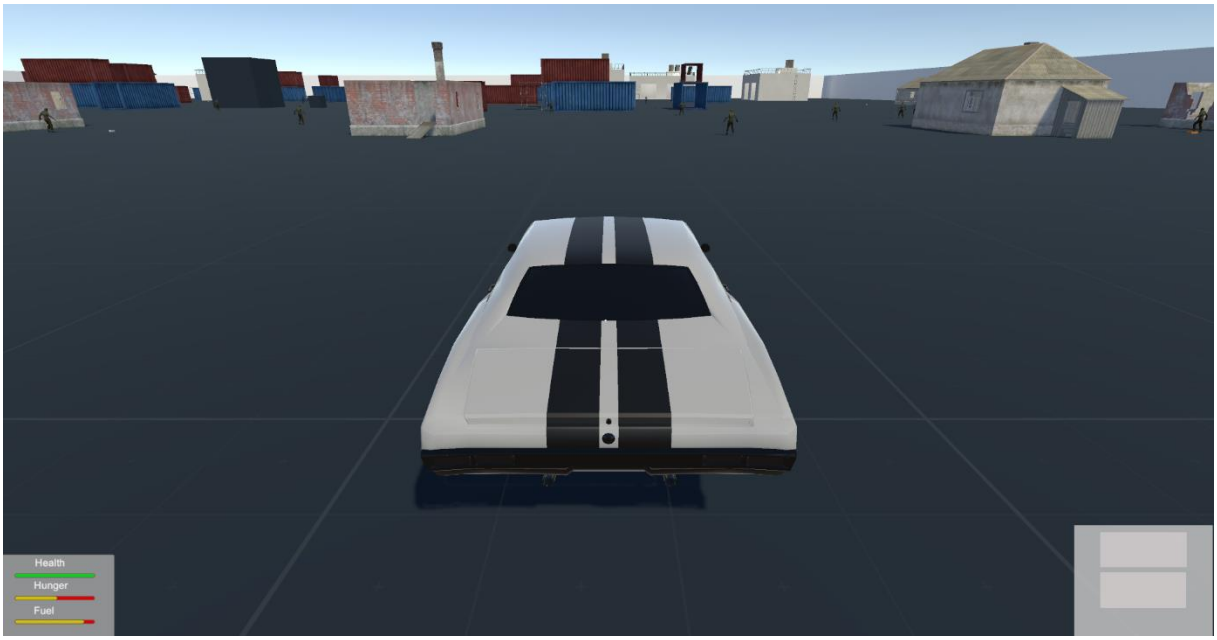
Slika 25. Model Auta

Kada se igra pokrene, skripte koje kreira Unity za upravljanje autom, su deaktivirane. Skripta koja upravlja aktiviranjem i ulaskom igrača u auto je skripta EnterVehcle. Navedena skripta sadrži dvije standardne funkcije Unity-a, a to su OnTriggerEnter() i OnTriggerExit(). Navedene funkcije mogu raditi samo ako su pozvane nad objektom koji ima Collider kojemu je aktivirana opcija Trigger. Pa tako funkcija OnTriggerEnter() provjerava da li je koji objekt ušao u Coliider-om odnosno u slučaju druge funkcije, da li je izašao iz Collider-a. U navedenom kodu vidimo primjer dviju funkcija. Također vidimo i metode CarActivate() i NoFuel().

```
private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == player)
            triggerCheck = true;
    }
    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject == player)
            triggerCheck = false;
    }
if (Input.GetKeyDown(KeyCode.F) && triggerCheck)
        {
            CarActivate();
        }

        if (fuelSlider.value >= 0 && triggerCheck)
        {
            fuelSlider.value -= Time.deltaTime / fuelFallRate * 10;
        }
        if (fuelSlider.value <= 0)
        {
            NoFuel();
        }

    }
    private void CarActivate()
    {
        LoadHolder.carAtive = false;
        fuelObject.SetActive(true);
        if (fuelSlider.value >= 0)
        {
            fuelSlider.value -= Time.deltaTime / fuelFallRate * 10;
        }
        rb = theCar.GetComponent<Rigidbody>();
        rb.constraints = RigidbodyConstraints.None;

        player.transform.tag = "Untagged";
        camera.SetActive(true);
        player.SetActive(false);
        audio = theCar.GetComponent<AudioSource>();
        if (audio != null)
            audio.enabled = true;

        theCar.GetComponent<CarController>().enabled = true;
        theCar.GetComponent<CarUserControl>().enabled = true;
        theCar.GetComponent<CarAudio>().enabled = true;
        exitTrigger.SetActive(true);
    }
    private void NoFuel()
    {
        theCar.GetComponent<CarController>().enabled = false;
        theCar.GetComponent<CarUserControl>().enabled = false;
        theCar.GetComponent<CarAudio>().enabled = false;
        audio = theCar.GetComponent<AudioSource>();
        if (audio != null)
            audio.enabled = false;
    }
```

Vidimo da standardne funkcije provjeravaju da li je Collider bilo kojeg objekta ušao u Collider koji je tipa Trigger. Ako je taj objekt igrač, tada se varijabla triggerCheck koja je tipa boolean postavlja na vrijednost true te omogućuje da se u Update() metodi ako je igrač pritisnuo tipku za ulaz u automobil aktivira metoda CarActivate().

Metoda CarActivate() nam služi kako bi smo aktivirali skripte koje služe za upravljanjem vozila, ujedno i deaktivira samog igrača te aktivira kameru koja se nalazi iznad vozila. Ako vozilo ima dovoljno goriva možemo upravljati njime. Gorivo je realizirano pomoću Slider-a Fuel koji se aktivira kada igrač sjedne u automobil te se tijekom vožnje smanjuje. Kao što je vidljivo na slici 26. Ako vrijednost Slider-a padne na nula, tada se pomoću metode NoFuel() skripte za upravljanje automobilom isključe i auto više nije upotrebljiv.



Slika 26. Razina goriva

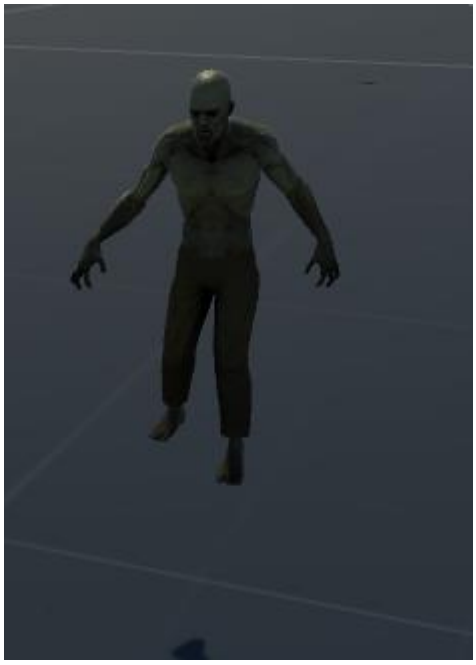
Kako bi smo izašli iz automobila potrebno je pritisnuti tipku F. Kao što je vidljivo iz navedenog koda, dešava se suprotna stvar nego kada ulazimo u automobil. Deaktiviraju se skripte koje upravljaju automobilom zajedno za kamerom koja se nalazi iznad automobila. Uzima se trenutna pozicija automobila kako bi se model igrača aktivirao na poziciji koja predstavlja izlaznu točku. Navedenu izlaznu točku smo definirali tako da smo u Hierarchy-u u direktorij automobila kreirali prazan objekt te ga pomaknuli malo lijevo od ulaznih vrata automobila. Pošto se prazan objekt nevidljiv i ujedno se nalazi u direktoriju automobila, on nije vidljiv te se kreče s automobilom.

S naredbom player.transform.position = exitPlace.transform.position; uzimamo poziciju navedene izlazne točke kako bi smo poziciju igrača izjednačili s pozicijom izlazne točke.

```
private void Update()
    {
      if (Input.GetKeyDown(KeyCode.F))
        {
            CarDeactivate();
        }
    }
private void CarDeactivate()
    {
        fuelObject.SetActive(false);
        rb = theCar.GetComponent<Rigidbody>();
        rb.constraints = RigidbodyConstraints.FreezePositionX;
        player.transform.tag = "Player";
        camera.SetActive(false);
        player.SetActive(true);
        player.transform.position = exitPlace.transform.position;
        theCar.GetComponent<CarController>().enabled = false;
        theCar.GetComponent<CarUserControl>().enabled = false;
        theCar.GetComponent<CarAudio>().enabled = false;
        theCar.GetComponent<AudioSource>().enabled = false;
        exitTrigger.SetActive(false);
        LoadHolder.carAtive = true;
    }
```

4.4. Neprijatelj (Zombie)

3D model Zombie-a je besplatno preuzet s Asset Store-a. Model kojega možemo vidjeti na slici 27, dolazi s tri animacije, a to su: Animacija stajanja na mjestu, animacija napadaja i animacije hodanja. Model na sebi sadrži nekoliko važnih komponenata kao što su: Animator pomoću kojeg upravljamo animacijama, Rigidbody koji dodjeljuje našem modelu osnovna fizikalna svojstva, Capsule Collider koji omogućuje da ostali Collider-i ne prolaze kroz njega. Komponente pomoću kojih se Zombie kreće su Nav Mesh Agent i Box Collider koji simulira oči Zombie-a. Zajedno sa navedenim komponentama, model sadrži još i tri skripte koji upravljaju njime.

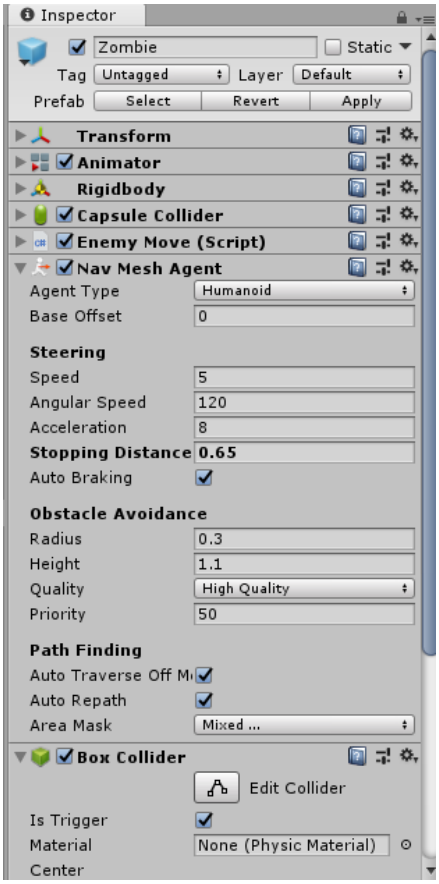


Slika 27. Model Neprijatelja

4.4.1. Kretanje neprijatelja

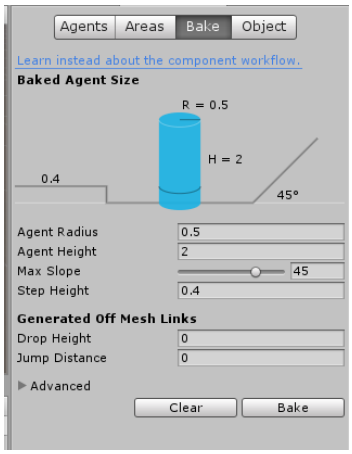
Kretanje neprijatelja je zamišljeno na način da se modeli nalaze na nekim određenim pozicijama na terenu, pokretanjem igre oni se samostalno kreću po slučajnim putanjama. Ako u tom svojem kretanju pomoću Box Collider-a koji je Trigger, uoče igrača, oni se više ne kreću slučajnom putanjom nego uzimaju poziciju igrača te se ubrzanim hodom kreću prema njemu. Neprijatelji prate igrača sve do onog trenutka kad on izađe iz Box Collider-a koji označava radijus do kojeg neprijatelji uočavaju igrača. Kad igrač izađe iz tog radijusa, neprijatelji se dalje kreću slučajnom rutom.

Kako bi smo navedeno realizirali potrebno je modelu Zombie-a dodati Unity komponentu Nav Mesh Agent. Navedena komponenta omogućava objektu na kojem se nalazi da prati definirane objekte ili pozicije na mapi. Kao što vidimo na slici 28 komponente u inspektoru, komponenta nam omogućuje da mijenjamo neke osobine kao što su brzina kretanja, udaljenost na kojoj objekt stane kad se približi definiranom objektu, radijus, veličina komponente, koja se bi trebala biti slične veličine kao i sam model.



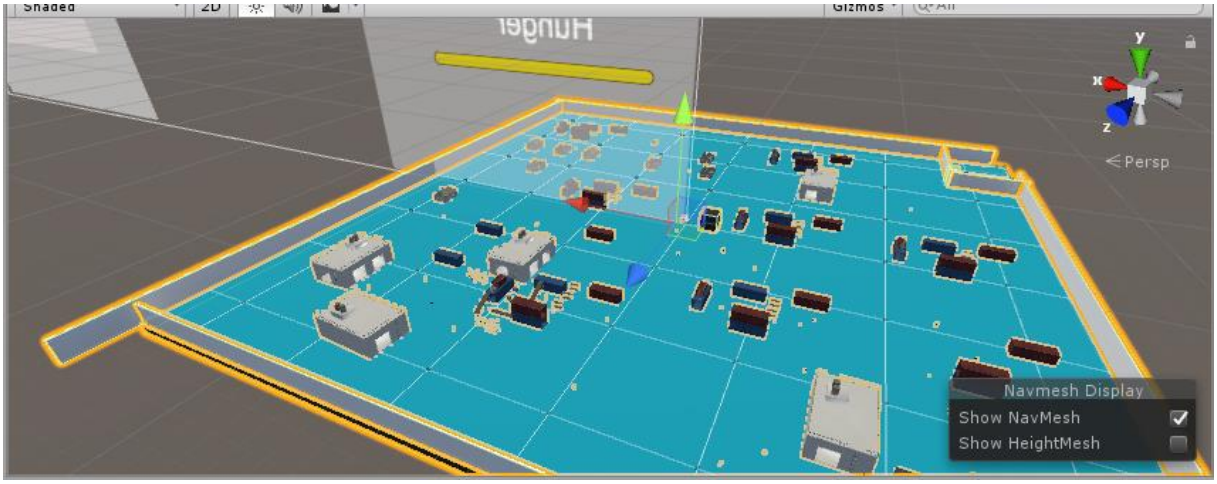
Slika 28. Nav Mesh Agent

Kada smo podesili osobne karakteristike komponente potrebno je označi po kojem dijelu terena će se objekt moći kretati, po kojem ne. Nav Mesh Agent komponenta nam omogućuje da označimo teren po kojem se objekt može kretati. U slučaju ove igre, označena je ravna ploha koja predstavlja pod. Kako objekt ne bi mogao hodati na neke modele koji su povišeni, definiramo na koji je najveći nagib na koji se objekt koji sadrži komponentu može popeti. Na slici 29 vidimo primjer opcija za nagib.



Slika 29. Opcija nagiba

Kada smo podesili sve željene opcije, komponenta nam na terenu plavom bojom označi kojim dijelom će se neprijatelji moći kretati. Na slici 30 vidimo primjer za našu igru.



Slika 30. Mjesta kretanja [Izradio autor rada]

Skripta koja upravlja kretanjem neprijatelja zove se EnemyMove. Navedena skripta sadrži metoda za kretanje, pokretanje animacija i pokretanje zvuka. Kao što vidimo na primjeru koda, skripta sadrži metodu NewTarget(). Metoda definira brzinu kojom će Nav Mash Agent micati objekt Zombie-a i brzinu kojom će se animacija hodanja vrtjeti. Metoda definira varijablu random koja uzima slučajnu vrijednost od sredine poda umnožene sa petsto kako bi se pokrila cijela površina terena. Ta varijabla sa koordinatama se zatim prenosi u Nav Mash Agent funkciju SetDestination koja pokreće objekt prema toj slučajno generiranoj destinaciji.

Kao što je već navedeno, neprijatelji sadrže Box Collider koji je Trigger te označava na koju udaljenost neprijatelj može uočiti igrača. Kada igrač uđe u Collider, neprijatelj ga uoči te se pokreće metoda MoveToPlayer(). Navedena metoda povećava brzinu kretanja i brzinu animacije što daje efekt trčanja. Nav Mesh Agent u funkciju SetDestination() ne prima slučajnu destinaciju prema kojoj će se kretati nego poziciju igrača. Metoda koja još upravlja kretnjom neprijatelja je metoda FacePlayer(). Navedena metoda služi kako bi se neprijatelj okretao očima prema igraču da ne može doći do situacije da Zombie stoji leđima okrenut prema igraču i napada ga.

Metoda koja upravlja animacijama neprijatelja možemo također vidjeti u primjeru koda, a to je CheckMovmend(). Metoda provjerava da li je neprijatelj promijenio poziciju, ako nije izvršava se animacija stajanja na mjestu, ako je neprijatelj promijenio poziciju aktivira se animacija za hodanje.

```
private void NewTarget()
    {
        nav.speed = 2;
        anim.speed = 1.5f;
        Vector3 random = Random.insideUnitSphere * 500;
        nav.SetDestination(random);

    }
private void MoveToPlayer()
    {
        nav.speed = 5;
        anim.speed = 2;
        nav.SetDestination(playerPosition.position);
    }

private void CheckMovmend()
    {
        if (zombiePosition.transform.hasChanged)
        {
            anim.SetBool("isIdle", false);
            anim.SetBool("isWalking", true);
        }
        else
        {
            anim.SetBool("isIdle", true);
            anim.SetBool("isWalking", false);
        }
    }
 private void FacePlayer()
    {
        Vector3 direction = (playerPosition.position - transform.position).normalized;
        Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0,
direction.z));
        transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation,
Time.deltaTime);
    }
    private void PlaySound()
    {
        int randomClips = Random.Range(0, zombieSounds.Length);
        audioSource.PlayOneShot(zombieSounds[randomClips]);
    }
```

Kada Zombie uoči igrača metoda PlaySound pokreće slučajnim odabirom jednu od tri zvuka koje neprijatelj može proizvoditi.

4.4.2. Napadanje igrača

Skripta koja upravlja napadajima neprijatelja je skripta EnemyAttack. Navedena skripta sadrži metodu Attack() koja provjera da li je Zombie na određenoj udaljenosti od igrača koja je namijenjena za napadanje. Ako je Zombie unutar zadane udaljenosti pokreće se animacija napadanja te se iz skripte koja upravlja zdravljem igrača playerHealth pokreće metodu TakeDemage() koja igraču oduzima zdravlje. Kao što vidimo iz koda skripte EnemyAttack u metodi koja upravlja napadanjem, bitno se smanjuje brzina kretanje neprijatelja i brzina

animacije. U Update() funkciji se ujedno provjerava da li je igra prekinuta i da li je igrač ušao u auto. Uz navedene uvjete još se provjerava da li je prošlo vrijeme između napadanja. To vrijeme iznosi pola sekunde i služi da se akcija napadanja vrši svakih pola sekundi.

```
private void Update()
    {
        timer += Time.deltaTime;
        if(timer >= timebetweenAttacks && !LoadHolder.endGame && LoadHolder.carAtive)
        {
            Attack();
        }
    }

 private void Attack()
    {
        timer = 0f;
        if (Vector3.Distance(player.position, this.transform.position) < 1.5f)
        {
            nav.speed = 0.5f;
            anim.speed = 1f;
            anim.SetBool("isWalking", false);
            anim.SetBool("isAttacking", true);
            playerHealth.TakeDemage(attackDemage);
        }
        else
        {
            anim.SetBool("isWalking", true);
            anim.SetBool("isAttacking", false);
        }
    }
```

4.4.3. Zdravlje neprijatelja

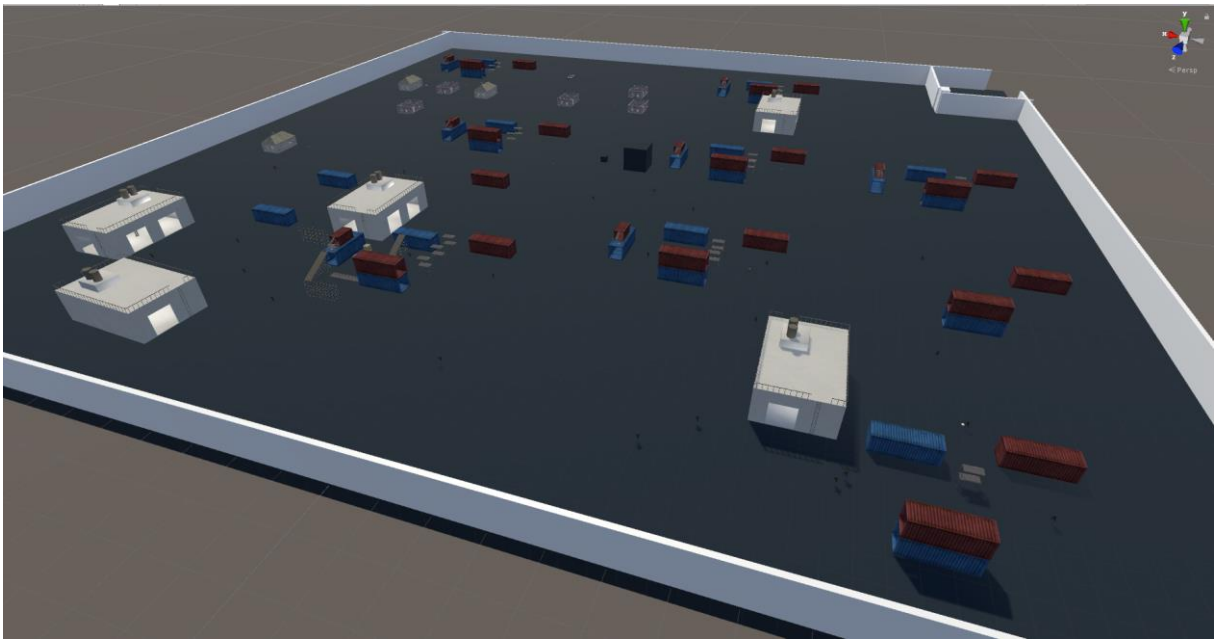
Skripta koja upravlja zdravljem neprijatelja je EnemyHealth. Skripta sadrži dvije važne metode, a to su TakeDemage() i Death(). Metoda TakeDemage() je javna te se ona pokreće kada igrač zapuca na Zombie-a. Metoda provjerava da li je zdravlje neprijatelja veće od nule, ako je taj uvijek ispunjen oduzima se zdravlje neprijatelju. Ako uvjet nije ispunjen znači da igrač nema zdravlja te se pokreće metoda Death() koja pokreće animaciju umiranja i uništava objekt. Navedene metode vidimo u slijedećem isječku koda skripte.

```
public void TakeDemage(int _amount)
    {
        if (isDead)
            return;
        currentHealth -= _amount;

        if (currentHealth <= 0)
        {
            Death();
        }

    }

private void Death()
    {
        isDead = true;
        anim.SetTrigger("isDying");
        nav.enabled = false;
        rb.isKinematic = true;
        Destroy(gameObject, 4f);
        nav.enabled = false;
    }
```

4.5. Teren

Unity omogućava da unesemo gotov teren ili da kreiramo svoj vlastiti. Unity omogućava kreiranje objekta tipa Terrain koji se na razne načine može podesiti. Kod ove igre nije korištena navedena opcija jer je u igri predviđen veliki teren za koji bi bilo potrebno puno resursa računala. Umjesto toga kao teren se koristi ravna ploha koja sadrži Collider i nema specifične efekte okoliša kao što to omogućuje Terrain objekt.. Na terenu se uz modele neprijatelja i objekata koje igrač može koristiti nalaze i objekti raznih kontejnera i kuća.

Na terenu se nalazi poseban dio zida koji simulira vrata u spas. Vrata sadrže Collider koji je tipa Trigger i aktivira se kada igrač uđe u njega. Ulaskom igrača u Collider igra se završava. Na slici 31 vidimo teren.



Slika 31. Teren

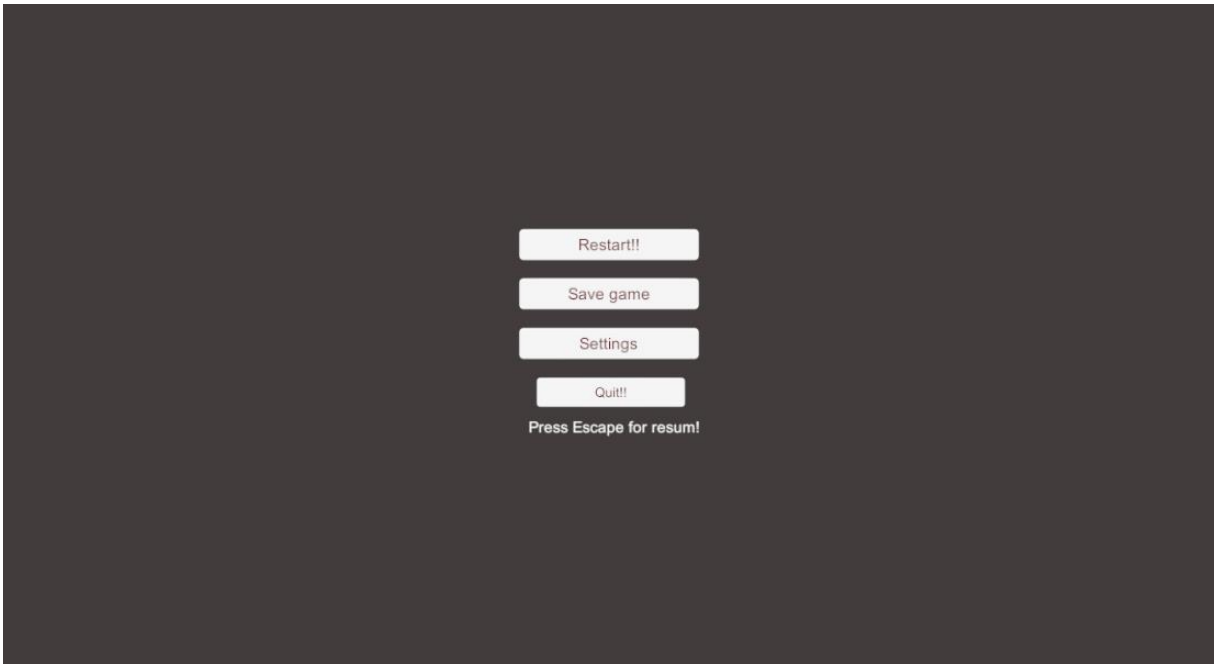
4.6. Sučelje igre

Prilikom pokretanja igre prvo se pojavljuje sučelje koje vidimo na slici 32. Sučelje nam omogućuje da pokrenemo igru ispočetka ili da nastavimo ako smo prije spremili stanje igre.

Slika 32. Početno sučelje

Tijekom igranja, igraču se nudi opcija da pauzira igru pritiskom na tipku Escape. Kada se igra pauzira pokrene se sučelje koje možemo vidjeti na slici 33. Sučelje nam omogućuje da resetiramo igru, spremimo igru kako bi smo neki drugi put mogli nastaviti tamo gdje smo stali.

Opciju Settings koja nema omogućuje da definiramo koja tipka na tipkovnici će izvršavati određene radnje i opciju Quit koja gasi scenu same igre i pokreće početno sučelje.



Slika 33. Sučelje u igri

4.6.1. Spremanje stanja igre

Spremanje stanja igre izvršava se pomoću Unity funkcije PlayerPrefs koja omogućuje da se vrijednosti varijabli spremaju u Windows Registry. „Windows Registry (registar) je baza podataka koja je sastavni dio operacijskog sustava Microsoft Windows i u kojoj su zabilježena podešavanja operativnog sistema i korisničkog okruženja, podaci o instaliranim programima, računarskoj opremi i svi drugi podaci vezani za pravilno funkcioniranje računalnog sustava“ [8]. Navedena funkcija može raditi gotovo na svim platformama. Stanja igre koja se spremaju su pozicija igrača, njegova razina zdravlja i gladi, oružja i količina metaka koja igrač posjeduje za određeno oružje. U slijedećem primjeru koda vidimo metode iz skripti koje se koriste kako bi se navedena stanja spremila.

```
public void SaveBullets()
{
    PlayerPrefs.SetInt("deagleBullets", deagleBullets);
    PlayerPrefs.SetInt("b556", b556);
    PlayerPrefs.SetInt("akScar", akScar);
    PlayerPrefs.SetInt("activeWeapon", activeWeapon);
    PlayerPrefs.SetInt("activePistole", activePistole);
    PlayerPrefs.SetInt("currentRifle", currentRifle);
    PlayerPrefs.SetInt("currentPistole", currentPistole);
    if (slot1)
    {
        activeSlot = 1;
    }
    else
    {
        activeSlot = 0;
    }
    PlayerPrefs.SetInt("slot1", activeSlot);
}

public void SavePlayerStats()
{
    PlayerPrefs.SetFloat("playerHealth", playerHealth);
    PlayerPrefs.SetFloat("playerHunger", playerHunger);
    PlayerPrefs.SetFloat("X", player.transform.position.x);
    PlayerPrefs.SetFloat("Y", 6);
    PlayerPrefs.SetFloat("Z", player.transform.position.z);
}
```

Iz navedenih primjera vidimo da u funkciji PlayerPrefs() odredimo koji tip će varijabla koji želimo spremiti, odredimo joj ime preko kojeg ćemo pristupati varijabli u Load() metodi i nakraju varijablu čiju vrijednost želimo spremiti.

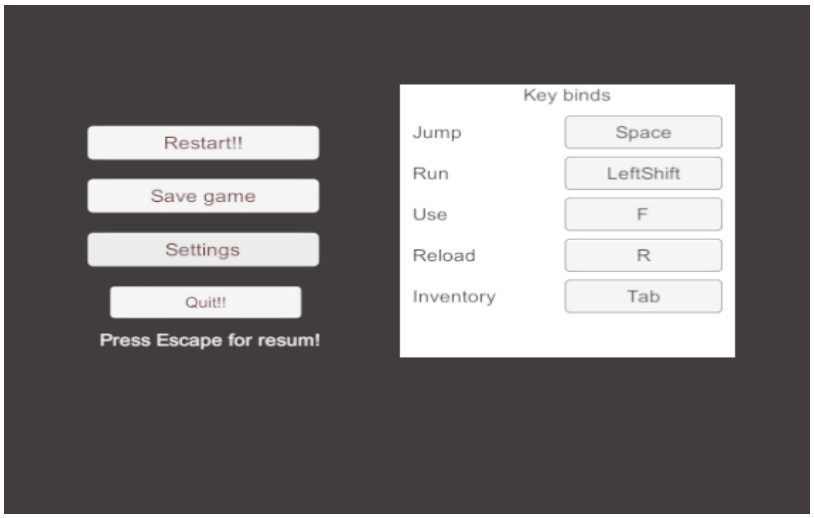
4.6.2. Učitavanje spremljenog stanja

Kako bi smo prethodno spremljene podatke učitali, koristimo također Unity funkciju PlayerPrefs(), ali u ovom slučaju koristimo opciju Get koja iz Windows Registry-a uzima vrijednosti varijabli te ih mi moramo samo izjednačiti sa varijablama u skriptama. Na slijedećem primjeru koda vidimo metode koje se koriste za učitavanje svih spremljenih podataka.

```
public void LoadBullets()
    {
        deagleBullets = PlayerPrefs.GetInt("deagleBullets");
        b556 = PlayerPrefs.GetInt("b556");
        akScar = PlayerPrefs.GetInt("akScar");
        activeWeapon = PlayerPrefs.GetInt("activeWeapon");
        activePistole = PlayerPrefs.GetInt("activePistole");
        currentRifle = PlayerPrefs.GetInt("currentRifle");
        currentPistole = PlayerPrefs.GetInt("currentPistole");
        activeSlot = PlayerPrefs.GetInt("slot1");
        if (activeSlot == 0)
        {
            slot2 = true;
        }
        if (activeSlot == 1)
        {
            slot1 = true;
        }
    }
public void LoadPlayer()
    {
        playerHealth = PlayerPrefs.GetFloat("playerHealth");
        playerHunger = PlayerPrefs.GetFloat("playerHunger");
        x = PlayerPrefs.GetFloat("X");
        y = PlayerPrefs.GetFloat("Y");
        z = PlayerPrefs.GetFloat("Z");
        player.transform.position = new Vector3(x, y, z);
        playerHP.LoadPlayerHealth();
    }
```

4.6.2. Promjena kontroli za igranje.

Kada u sučelju kada je igra zaustavljena kliknemo gumb Settings otvorit će nam se dodatni prozor u kojem možemo promijeniti neke od osnovnih kontroli za upravljanje. Na slici 34 vidimo prozor u kojem možemo napraviti promjene.



Slika 34. Mijenjanje kontrola

Prozor je realiziran na način da su ubačeni objekti tipa Button na koje se ispisuje ujedno i ime tipke na tipkovnici koja će izvršavati određene operacija. Promjena kontroli je realizirana tako da u Input Manager dodamo određene opcije te im dodijelimo tipku na tipkovnici koja tu opciju izvršava. Input Manager je mjesto gdje definiramo unose koji će upravljati akcijama u našoj igri [9]. Pošto nam Unity ne omogućava da tijekom igranja mijenjamo vrijednosti Input Manager-a kreirana je skripta InputManager u kojoj je definirana rječnik sa akcijama, čija su imena jednaka onima u Input Manager-u Unity-a. Kao što je vidljivo na dijelu koda skripte, akcijama su dodijeljene početne komande. U metodi GetButtonDown() provjera se da li je tipka tipkovnice koja je definirana u opcijama pritisnuta. Tako se preko Unity Input.GetKey() funkcije izvršava pokretanje akcije u Input Menager-u, a da se direktno ne mijenja vrijednost te akcije nego se samo u igri simulira pritisak tipke koja je tamo definirana preko neke druge koju smo mi sami odabrali.

```
Dictionary<string, KeyCode> buttonKeys;
private void OnEnable()
    {
        buttonKeys = new Dictionary<string, KeyCode>();
        buttonKeys["Jump"] = KeyCode.Space;
        buttonKeys["Run"] = KeyCode.LeftShift;
        buttonKeys["Use"] = KeyCode.F;
        buttonKeys["Reload"] = KeyCode.R;
        buttonKeys["Inventory"] = KeyCode.Tab;
    }
 public bool GetButtonDown(string buttonName)
    {
        if (buttonKeys.ContainsKey(buttonName) == false)
        {
            Debug.LogError("No button named: " + buttonName);
            return false;
        }
        return Input.GetKeyDown(buttonKeys[buttonName]);
    }
```

Dalje je samo potrebno u skriptama gdje se provjerava da li je aktiviran koji gumb na tipkovnici ili mišu, staviti da se poziva InputManager skripta koju smo mi sami napisali, umjesto one koju Unity koristi i koja se ne može mijenjati. Primjer pozivanja navedene skripte vidimo u slijedećem isječku koda koji upravlja s aktivacijom Inventory-a.

```
if (inputManager.GetButtonDown("Inventory"))
```

5. Zaključak

Kreiranje igra pomoću Uniry-a je dosta jednostavno i lako shvatljivo. Unity nam omogućuje da radimo igre namijenjene gotovo za sve platforme i igrače konzole koje postoje na tržištu, što nam uvelike olakšava razvoj jer ne moramo razmišljati kako podesiti kod igrice kako bi on bio kompatibilan s raznim platformama. Uz navedene prednosti, uvelike se ističe i brzina razvoja. Unity sadrži već neke gotove komponente kao što su komponente za upravljanje igračem ili vozilima, gotove efekte eksplozija, itd. Navedene komponente se bez previše problema mogu prilagoditi vlastitim potrebama te tako ubrzavaju razvoj jer oslobađaju programere da rade neke specifične stvari koje su u većini slučaja svugdje iste.

Najzahtjevniji dio kod razvoja igre Zombie apokalipsa predstavio je razvoj sučelja koji bi omogućio promjenu kontroli. Kao što je navedeno u radu, ne postoji neki direktan način da se u Unity-u mijenja Input Manager nego moramo sami pisati novog ispočetka, što je dosta zahtjevan proces ili pokušati manipulirati njime tako da ne mijenjamo direktno definirana svojstva Managera nego svojstva unosa, kao što je prikazano u radu.

Proces spremanja stanja igre se isto ispostavio kao dosta zahtjevan. Ne zbog samog kodiranja, nego zbog logike povezanosti svih komponenata. Potrebno je bilo izraditi za svaki segment spremanja, tj. za svaku skupinu kao što su oružja i zdravlje igrača, posebnu klasu koja upravlja vrijednostima varijabli. Kako bi se omogućilo pokretanje spremljenih stanja na računalu.

LITERATURA

1. Wikipedija, Unity(game engine), dostupno 23.8.2018. na: https://en.wikipedia.org/wiki/Unity_(game_engine)
2. Unity Documentation, The Inspector Window, dostupno 23.8.2018. na: https://docs.unity3d.com/Manual/UsingTheInspector.html
3. Unity, Rigidbody, dostupno 23.8.2018. na: https://docs.unity3d.com/ScriptReference/Rigidbody.html
4. Unity, The Hierarchy window, dostupno 23.8.2018. na: https://docs.unity3d.com/Manual/Hierarchy.html
5. Unity, Scene View navigation, dostupno 23.8.2018. na: https://docs.unity3d.com/Manual/SceneViewNavigation.html
6. Unity, The Game view, dostupno 23.8.2018. na: https://docs.unity3d.com/Manual/GameView.html
7. Unity, Enumerations, dostupno 23.8.2018. na: https://unity3d.com/learn/tutorials/topics/scripting/enumerations
8. Wikipedija, Windows Registry, dostupno 23.8.2018. na: https://bs.wikipedia.org/wiki/Windows_Registry
9. Unity, Input Manager, dostupno 23.8.2018. na: https://docs.unity3d.com/Manual/class-InputManager.html
10. Unity Asset Store, Zombie, dostupno 23.8.2018. na: https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232
11. Unity Asset Store, #004 Muscle Car, dostupno 23.8.2018. na: https://assetstore.unity.com/packages/3d/vehicles/land/004-muscle-car-100307
12. Unity Asset Store, Abandoned buildings, dostupno 23.8.2018. na: https://assetstore.unity.com/packages/3d/environments/abandoned-buildings-62875
13. Unity Store, Unity Engine, dostupno 3.9.2018. na: https://store.unity.com/
14. Gimp, Gimp GNU Image Manipulation Program, dostupno 3.9.2018. na: https://www.gimp.org/
15. Audacity Team, Audacit, dostupno 3.9.2018. na: https://www.audacityteam.org/download/
16. Blender, Blender, dostupno 3.9.2018. na: https://www.blender.org/
17. PUBG Corporation, PUBG, dostupno 3.9.2018. na: https://store.steampowered.com/app/578080/PLAYERUNKNOWNS_BATTLEGROUNDS/
18. AMC, The Walking Dead, dostupno 3.9.2018. na: https://www.amc.com/shows/the-walking-dead