Izrada računalne igre temeljene na sustavu prikupljanja plijena

Marković, Marko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marko Marković

IZRADA RAČUNALNE IGRE TEMELJENE NA SUSTAVU PRIKUPLJANJA PLIJENA

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marko Marković

Matični broj: 46447/17–R

Studij: Poslovni sustavi

IZRADA RAČUNALNE IGRE TEMELJENE NA SUSTAVU PRIKUPLJANJA PLIJENA

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Mario Konecki

Varaždin, rujan 2022.

Marko Marković

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je izrada računalne igre na sustavu prikupljanja plijena(engl. *Loot*). Igra će se izrađivati u Unity razvojnom okruženju. Igrač će u igri moći prikupiti plijen u obliku različitih tipova oružja i oklopa koji se nalazi u kutijama. Zatim će ga moći iskoristiti u borbi koja se odvija u rundama i na početku svake runde igrač će moći ponovno prikupiti drugačiji plijen za sljedeću rundu. Izrađena igra će biti akcijska igra iz prvog lica. Kroz završni rad osmislit će se i prikazati kako je plijen strukturiran i kako se generiraju svojstva plijena. Svaka instanca plijena u igri imat će drugačije vrijednosti tih svojstava. Također će biti prikazano kako svojstva plijena utječu na tijek borbe. Uz to prikazat će se i ostali dijelovi igre kao što su: izrada i programiranje sučelja, programiranje kontrola za igrača, programiranje borbe i neprijatelja.

Ključne riječi: Plijen, Prikupljanje, Generiranje, Borba, Razvoj igre, Unity, C#.

Sadržaj

1. Uvod

U ovom radu izrađivat će se računalna igra s sustavom prikupljanja plijena pa bi najprije trebalo objasniti što je plijen. Plijen(engl. *Loot*) u kontekstu videoigara su svi objekti unutar igre koje lik igrača može pokupiti, kupiti, prodati, otključati, koristiti, modificirati, potrošiti i slično. Plijen obično ima svoja svojstva koja mogu privremeno ili trajno promijeniti igračeve sposobnosti, ali može ga koristiti samo u kozmetičke svrhe, naprimjer promjena odjeće lika kojeg igrač kontrolira.[1]

Računalna igra će se izrađivati u programskom alatu *Unity* pa će u radu će prvo biti obrađen osnovni rad u *Unityju*. Nakon toga bit će nabrojani neki primjeri igara koji su bili inspiracija za ovu igru i kako će ova računalna igra izgledati. Prvi korak u izradi bit će postavljanje scena, odnosno postavljanje početka, sredine i kraja igre, a nakon toga detaljno osmisliti i obraditi najvažnije komponente igre. Najbitniji dio je naravno osmišljavanje tipova plijena, kakva će plijen imati svojstva i razmisliti kakav bi učinak ta svojstva imala na samu igru. Sljedeći korak bit osmisliti kako se plijen generira. Nakon što je plijen generiran, trebat će se osmisliti kako će se plijen u igri prikazati i koristiti. Posljednji dio izrade igre bit će osmišljavanje i izrada borbe u kojoj će se moći isprobati različite kombinacije plijena .

2. Unity

Unity je višeplatformski softver za izradu 2D i 3D videoigara. Prva verzija softvera objavljena je 2005. godine, a do danas je softver nadograđen za razvoj desktop aplikacija, mobilnih aplikacija, konzolnih aplikacija i aplikacija namijenjenih za platforme s virtualnom stvarnošću. Programski jezik koji se u *Unityju* koristi za skriptiranje je C#.[2]

2.1. Unity Hub

Unity Hub je aplikacija koja korisniku omogućuje upravljanje projektima, prijavu na *Unity* korisnički račun i instalaciju raznih verzija *Unity Editor* softvera.[3] Da bi se instalirao *Unity* Editor, u *Unity Hubu* odaberemo *Installs* pa kliknemo na *Install Editor*. Pojavi nam se ekran s verzijama *Unity* Editora. Klikom na *Install* bilo koje verzije pojavi se prozor gdje možemo odabrati koje dodatne module želimo koristiti. Moduli uključuju podršku za razvoj aplikacija na različitim platformama, dokumentaciju i dodatne jezične pakete. Nudi se i mogućnost instalacije *Microsoft Visual Studio* za skriptiranje, no u izradi projekta koristit će se *Visual Studio Code* s proširenjem za rad u *Unityju*. *Unity Editor* započinje instalaciju klikom na gumb *Install*.

Rad u *Unityju* počinje stvaranjem novog projekta. U *Unity Hubu* odabiremo *Projects* pa kliknemo na gumb New Project. Na zaslonu stvaranja projekta možemo odabrati želimo li stvoriti 2D ili 3D projekt s različitim unaprijed izrađenim dijelovima igre kao što su kretnja u prvom ili trećem licu, igra utrkivanja i slično. Za ovaj projekt koristit će se prazni 3D projekt. Uz to se u kreiranju projekta može odabrati ime projekta i lokacija gdje će projekt biti spremljen. Klikom na *Create project* stvara se novi projekt i otvara se *Unity Editor* s učitanim novim projektom.

2.2. Unity Editor

Slika 1: Sučelje *Unity Editora* (Izvor: *Unity documentation*, 2021)

Sučelje *Unity Editora* sa zadanim postavkama prikazano je na slici 1. Slovom A označena je alatna traka koja služi za pristup *Unity* računu i uslugama povezanim s *Unity* računom. Slovom B označen je hijerarhijski prikaz svih objekata u sceni. Slovom C označen je pogled u igri. To je pogled koji igrač ima u igri kroz kameru u sceni. Pogled scene omogućava navigaciju scenom i uređivanje scene, a njen prikaz označen je slovom D. Svaki objekt u sceni pripada klasi *GameObject* i sastoji se od različitih komponenti koje možemo vidjeti u inspektoru koji je označen slovom F. Slovom E označeni su osnovni alati za manipulaciju scenom. Slovom H označena je statusna traka koja izvještava korisnika o različitim procesima u *Unityju*. Slovom G označen je prozor projekta. U prozoru Project može se vidjeti sva imovina(engl. *Assets*) uključena u projekt.[4]

Imovina uključuje sve resurse koji možemo uključiti u igru, kao naprimjer: unaprijed konfigurirane objekte igre(engl. *Prefabs*), teksture, materijale, modele, animacije i slično. Velik izbor imovine dostupan je za kupnju na online trgovini *Unity Asset Store*. Sva kupljena imovina može se preuzeti i uvesti u projekt preko prozora *Package Manager* u *Unity Editoru*.[5]

3. Ideja računalne igre s sustavom prikupljanja plijena

Najčešći žanrovi videoigara u kojima možemo pronaći sustav prikupljanja plijena su igre s igranjem uloga i takozvani *looter shooteri*. Igra s igranjem uloga je tip igre u kojoj igrač kontrolira svojeg lika koji ima određeni skup sposobnosti i tijekom igre igrač može utjecati na tijek igre donošenjem odluka. Žanr se razvio iz igara na ploči, a posebno iz igre *Dungeons & Dragons*. U igrama s igranjem uloga sustav plijena se koristi na način da igrač pokušava prikupiti i koristiti onaj plijen koji odgovara njegovom stilu igre i koji najbolje odgovara sposobnostima njegovog lika.[6] *Looter shooteri* su igre pucanja iz prvog ili trećeg lica, ali za razliku od klasičnih igra pucanja, imaju jako razvijen sustav plijena i plijen se generira nasumično te je cilj prikupiti najbolju kombinaciju plijena kroz tijek igre.[7]

Slika 2: Prikaz videoigre Diablo 2 (Izvor: Sličica iz videa [8])

Sustav prikupljana plijena je u nekoj svojoj verziji prisutan u velikom broju video igara. Popularizirala ga je serijal videoigara *Diablo*(slika 2), koji spada u već spomenuti žanr igara s igranjem uloga. U toj igri plijen se generira nasumično, a može se pokupiti nakon što je igrač porazio neprijatelja. Što je igrač porazio jačeg neprijatelja, veće su bile šanse da neprijatelj ispusti jači plijen.

Generirani plijen je imao svoju kvalitetu prema kojoj su se prilagođavala njegova svojstva. Kvaliteta plijena je bila označena bojom, siva i zelena za lošu kvalitetu, plava i ljubičasta za srednju kvalitetu, žuta i narančasta dobru kvalitetu plijena. Označavanje plijena tim bojama proširilo se i na druge serijale videoigara i vrlo često se može vidjeti u novim videoigrama.[1] Taj koncept kvalitete plijena će se primijeniti u izradi računalne igre koja je tema ovog rada.

Slika 3: Prikaz igre Borderlands 3 (Izvor: Sličica iz videa [9])

Jedan od najpoznatijih predstavnika *looter shooter* žanra je serijal videoigara *Borderlands*(slika 3). Igrač na početku svake igre bira lika s kojim će igrati. Tijekom igre pronalazi plijen koji ispuštaju pobijeđeni protivnici ili pronalazi plijen u kutijama. Plijen se nasumično generira slično kao i u *Diablo* igrama. Igrač napredovanjem kroz igru može koristiti sve jači plijen, odnosno korištenje plijena je ograničeno razinom igrača. Isto tako, protivnici su napredovanjem kroz igru sve jači pa je igrač prisiljen stalno mijenjati plijen koji već koristi.[10] *Borderlands* serijal je najveća inspiracija za izradu igre, jer će biti istog žanra, plijen će imati slična svojstva i imat će napredovanje kroz igru.

3.1. Opis računalne igre za izradu

Izrađena igra sa sustavom prikupljanja plijena će biti *looter shooter* u prvom licu. Tijek igre koja će biti izrađena je sljedeći: prilikom pokretanja igre, igrač(engl. *Player*) će vidjeti početni izbornik gdje će moći odabrati da igra igru ili da izađe iz igre. Odabiranjem igranja igre, igrač će se naći u sobi sa dvije kutije(engl. *Box*) u kojoj će biti plijen.

Plijen će biti podijeljen na dvije glavne kategorije, a to su oružje(engl. *Weapon*) i oklop(engl. *Armor*). Oružja i oklopi će imati po četiri vrste potkategorija. Potkategorije oružja će biti pištolj(engl. *Handgun*), sačmarica(engl. *Shotgun*), jurišna puška(engl. *Assault Rifle*) i snajper(engl. *Sniper Rifle*). Potkategorije oklopa će biti kaciga(engl. *Helmet*), gornji oklop(engl. *Upper Armor*), donji oklop(engl. *Lower Armor*) i čizme(engl. *Boots*). Sva oružja će imati sedam osnovnih svojstava(engl. *Stats*) a to su jačina napada (engl. *Attack Damage*), jačina kritičnog pogotka(engl. *Critical Hit Damage*), šansa za kritični pogodak(engl. *Critical Hit Chance*), brzina pucanja(engl. *Fire Rate*), brzina punjenja oružja(engl. *Reload Speed*), broj metaka(engl. *Magazine size*) i domet(engl. *Range*). Svaka potkategorija oružja imat će drugačije vrijednosti tih svojstava. Svi oklopi sadržavat će tri svojstva: životni bodovi(engl. *Health*), šansa za blokiranje protivničkog napada(engl. *Block Chance*) i brzina kretanja(engl. *Movement Speed*).

Plijen će imati i svoju kvalitetu(engl. *Quality*) koja će se za svaku instancu plijena generirati nasumično. Šansa za pojavljivanje instance plijena u kutiji bit će obrnuto proporcionalna kvaliteti tog plijena. Kvaliteta plijena odlučivat će o tome koliko dodatnih svojstava sadrži plijen, no neće odlučivati o tome koliko su dobre vrijednosti tih svojstava. Vrijednosti će se generirati nasumično unutar predodređene donje i gornje granice vrijednosti. Svaka instanca plijena će imati jednu od pet razina kvalitete koja je u sučelju naglašena prikladnom bojom i brojem u zagradi pored imena plijena.

Otvaranjem kutije pojavit će se sadržaj kutije i sadržaj igračeve opreme u kojoj se na početku igre ne nalazi ništa. Ispod igračeve opreme, igrač će imati popis svih svojstava i njihove vrijednosti koje igrač može promijeniti opremanjem, odnosno prikupljanjem plijena iz kutije. U jednoj kutiji bit će generirana oružja, a u drugoj oklopi. Igrač pozicioniranjem miša na plijen može vidjeti opis tog plijena, a klikom na plijen može ga pokupiti i automatski se opremiti tim plijenom. Igrač može pokupiti jedan pijen od svake potkategorije oklopa i samo jedan plijen iz kategorije oružje. Prilikom opremanja, zbrajaju se sva svojstva iz igračeve trenutne opreme i prikazuju u popisu svojstava.

Kad igrač smatra da je odabrao najbolju opremu, na izlazu iz sobe pritisnut će crveni gumb. Pri tome će kutije s plijenom nestati a u većoj prostoriji pojavit će se pet neprijatelja(engl. *Enemy*) koji će napadati igrača. Igrač mora u borbi pobijediti tih pet neprijatelja. Ako ih ne pobijedi, pojavljuje se izbornik koji pokazuje da je igra završena i ima izbor ponovno igrati igru iz početka ili izaći iz igre. Ukoliko igrač pobijedi sve neprijatelje, ulazi u novu rundu i može opet birati novi plijen ili igrati sa starim.

Igra se igra u rundama(engl. *Round*). Igrač počinje prvom rundom i svaki put kad pobijedi neprijatelje ulazi u novu rundu. Svaku novu rundu povećavaju se jačina napada i životni bodovi neprijatelja. Isto tako povećava se i jačina napada oružja i životni bodovi oklopa koji su generirani u toj novoj rundi. Razlog tome je da bi se igrača prisililo da kroz igru pokušava redovito mijenjati svoju opremu, no jačina neprijatelja se kroz runde povećava bolje nego jačina plijena što svaku novu rundu igre čini težom. Svaku novu rundu ili promjenu opreme životni bodovi će se resetirati na trenutni maksimalni broj životnih bodova.

Igrač će u donjem lijevom kutu ekrana moći pratiti koja je trenutna runda, koliko trenutno ima životnih bodova i koliko još ima metaka prije nego što treba ponovno napuniti oružje. Uz to, imat će i popis kontrola za igranje igre na tipkovnici i mišu. Približavanjem i fokusiranjem kamere na kutiju s plijenom ili na gumb za početak borbe pojavit će se indikator da se kutija može otvoriti ili da se gumb može pritisnuti. Fokusiranjem kamere na neprijatelja, na vrhu ekrana pojavit će se grafički i tekstualni prikaz neprijateljevih životnih bodova koji će se tijekom borbe mijenjati.

5. Plijen

Prilikom izrade klasa plijena i klasa svojstva plijena bit će izrađene skripte koje nasljeđuju klasu *ScriptableObject*. Općenito, kada u *Unityju* stvaramo novu C# skriptu ona automatski nasljeđuje klasu *MonoBehaviour*. To je klasa koja je standardna za sve skripte u *Unityju* i sadrži metode kao što su između ostalih metode *Start* i *Update*. Metoda *Start* se izvodi kada se skripta omogući, a metoda *Update* se izvodi svaku sličicu(engl. *frame*) kad je omogućena.[11]

Klasa *ScriptableObject* nema te metode. Ona služi primarno za pohranjivanje podataka i za razliku od *MonoBehaviour* klase onda se ne može koristiti kao komponenta *GameObjecta*, no klasa *ScriptableObject* se može referencirati na varijablu u klasi *MonoBehaviour*.[12] Klase *ScriptableObject* će se u ovom slučaju koristiti iz tog razloga da, uvijek kad instanciramo novi plijen, ne moramo instancirati novi *GameObject*, nego na iste *GameObjecte* koji su već instancirani u sceni, referenciramo instance klase *ScriptableObject* pa prema podacima u *ScriptableObjectu* modificiramo podatke u *GameObjectu*.

5.1. Skripta *Stat.cs*

Prije nego što se definira klasa *Stat*, u skripti *Stat.cs* definirat ćemo enumeraciju *StatType*. Kod definicije enumeracije je sljedeći:

```
public enum StatType
{
    AttackDamage, CritDamage, CritChance, FireRate, ReloadSpeed, ClipSize,
    Range, Health, BlockChance, MoveSpeed
}
```

Enumeracija *StatType* sadrži sva svojstva koja igrač može imati, a enumeracija će služiti tome da se u kodu lakše manipuliraju i preračunavaju vrijednosti svojstva koje igrač dobiva prikupljanjem plijena. Klasa *Stat* definirana je na sljedeći način:

```
using System;
using UnityEngine;

public class Stat : ScriptableObject
{
    public StatType statType;
    public float value;
```

Klasa Stat sadrži atribute *statType* tipa *StatType* u koju se sprema tip svojstva, i atribut *value* tipa *float* u koju se sprema vrijednost svojstva. Klasa sadrži 4 metode. Prva je metoda *CreateSpecificStat* i definirana je sljedećim kodom:

```
public static Stat CreateSpecificStat(StatType statType, float value)
{
    Stat newStat = ScriptableObject.CreateInstance<Stat>();
    newStat.statType = statType;
    newStat.value = value;
    return newStat;
}
```

Metoda *CreateSpecificStat* je statična metoda koja kreira novu instancu svojstva s točno određenim tipom svojstva i točno određenom vrijednošću tog svojstva te ga vraća. Metoda prima tip i vrijednost svojstva preko parametara, a budući da svojstvo nasljeđuje klasu *ScriptableObject*, instancira se metodom *ScriptableObject*.*CreateInstance*.[12]

Sljedeća metoda klase *Stat* je *CreateRandomStat* i definirana je na sljedeći način:

```
public static Stat CreateRandomStat()
{
   Stat newStat = ScriptableObject.CreateInstance<Stat>();
   newStat.statType = (StatType)(UnityEngine.Random.Range(0, 10));
   switch (newStat.statType)
   {
        case StatType.AttackDamage:
            newStat.value = UnityEngine.Random.Range(10f, 21f);
            return newStat;
        case StatType.CritDamage:
            newStat.value = UnityEngine.Random.Range(10f, 21f);
            return newStat;
        case StatType.CritChance:
            newStat.value = UnityEngine.Random.Range(5f, 10f);
            return newStat;
        case StatType.FireRate:
            newStat.value = UnityEngine.Random.Range(0.1f, 0.5f);
            return newStat;
        case StatType.ReloadSpeed:
            newStat.value = UnityEngine.Random.Range(0.1f, 0.5f);
            return newStat;
        case StatType.ClipSize:
            newStat.value = UnityEngine.Random.Range(1f, 5f);
            return newStat;
        case StatType.Range:
            newStat.value = UnityEngine.Random.Range(1f, 10f);
            return newStat;
        case StatType.Health:
            newStat.value = UnityEngine.Random.Range(5f, 10f);
            return newStat;
```

```
            case StatType.BlockChance:
                newStat.value = UnityEngine.Random.Range(1f, 5f);
                return newStat;
            default:
                newStat.value = UnityEngine.Random.Range(1f, 5f);
                return newStat;
        }
}
```

Metoda *GenerateRandomStat* kreira jednu instancu svojstva i vraća ju. Jedan od deset tipova svojstava se prvo generira, a zatim se prema tipu svojstva generira vrijednost svojstva. Minimalne i maksimalne vrijednosti svojstava odabrane su na taj način da imaju vidljiv utjecaj na igru, no ako plijen ima više svojstava istog tipa, igra ne postane prelagana za igrača.

Sljedeća metoda klase *Stat* je *PrintStat* koja prema tipu svojstva vraća opis svojstva tipa *string*, a definirana je na sljedeći način:

```
public string PrintStat()
{
    switch (statType)
    {
        case StatType.AttackDamage:
            return Math.Round(value) + " - Attack damage";
        case StatType.CritDamage:
            return Math.Round(value) + "% - Critical hit damage";
        case StatType.CritChance:
            return Math.Round(value) + "% - Critical hit chance";
        case StatType.FireRate:
            return Math.Round(value, 2) + " - Fire rate";
        case StatType.ReloadSpeed:
            return Math.Round(value, 2) + " - Reload speed";
        case StatType.ClipSize:
            return Math.Round(value) + " - Clip size";
        case StatType.Range:
           return Math.Round(value) + " - Range";
        case StatType.Health:
            return Math.Round(value) + " - Health";
        case StatType.BlockChance:
            return Math.Round(value) + "% - Block chance";
        default:
            return Math.Round(value) + " - Movement speed";
    }
}
```

Zadnja metoda klase *Stat* je *DeleteStat* koja pomoću metode *Object.Destroy* briše instancu svojstva iz igre.[13] Kod za tu metodu je sljedeći:

```
public void DeleteStat()
{
    Destroy(this);
}
```

5.2. Skripta *Loot.cs*

Prije izrade klase plijena, u skripti *Loot.cs* definirat ćemo dvije enumeracije zbog lakšeg snalaženja u kodu, a to su *LootType* i *EquipmentType*. Kod enumeracije *LootType* i *EquipmentType* je sljedeći:

```
public enum LootType
{
    Handgun, Shotgun, AssaultRifle, SniperRifle, Helmet, UpperArmor,
    LowerArmor, Boots
};
public enum EquipmentType
{
    Weapon, Head, Chest, Legs, Feet, Armor
};
```

Enumeracija *LootType* sadrži sve tipove plijena koji će se moći generirati. Enumeracija *EquipmentType* služit će nam za određivanje mjesta u opremi na koje možemo pokupiti određeni tip plijena i služit će nam da odredimo hoće li se u kutiji generirati plijen tipa oružje ili tipa oklop.

Klasa *Loot* definirana je u skripti *Loot.cs*. Sadrži sljedeće atribute:

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

public class Loot : ScriptableObject
{
    public LootType lootType;
    public EquipmentType equipmentType;
    public int quality;
    public string label;
    public Sprite icon;
    public List<Stat> basicStats = new List<Stat>();
    public List<Stat> bonusStats = new List<Stat>();
```

U atribut *lootType* spremat će se tip plijena i tipa je *LootType* koji je prethodno definiran. U atribut *equipmentType* spremat će se tip opreme plijena, a atribut je tipa *EquipmentType* koji smo također prethodno definirali. Atribut *quality* spremat će vrijednost kvalitete plijena i tipa je int. Kvaliteta će moći biti od 1 do 5. Atribut *label* je tipa *string* i sadržavat će ime plijena koje će se prikazivati u sučelju igre. Atribut *icon* sadržavat će referencu na ikonu koja će se koristiti za pojedini tip plijena. Tipa je *Sprite*, što je 2D grafički objekt u *Unityju*.[14]

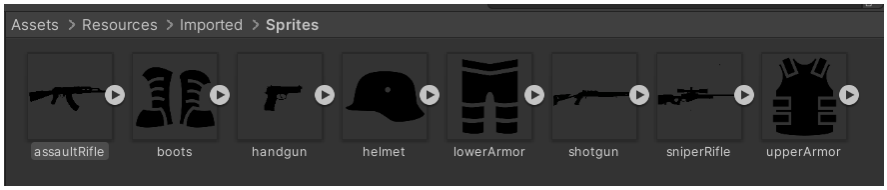
Definirat ćemo još dvije liste. Prva lista je *basicStats*. U nju će se spremati svojstva plijena, a ta svojstva će biti ista kod svake instance plijena istog tipa. Lista *bonusStats* sadržavat će još jedno do maksimalno pet dodatnih svojstava plijena, ovisno o tome koja će biti kvaliteta te instance plijena.

Klasa plijena imat će metodu *GenerateLootProperties* koja je prikazana sljedećim kodom:

```
public void GenerateLootProperties(LootType lootType, EquipmentType
equipmentType, string partialLabel, string iconName)
{
    this.lootType = lootType;
    this.equipmentType = equipmentType;
    this.quality = GenerateQuality();

    this.label = partialLabel;
    switch (quality)
    {
        case 1:
            this.label += " (1)";
            break;
        case 2:
            this.label += " (2)";
            break;
        case 3:
            this.label += " (3)";
            break;
        case 4:
            this.label += " (4)";
            break;
        default:
            this.label += " (5)";
            break;
    }
    icon = Resources.Load<Sprite>("Imported/Sprites/" + iconName);
}
```

Ta metoda prima vrijednosti svih parametra koji se pridodaju prethodno navedenim atributima, generira kvalitetu plijena pomoći metode *GenerateQuality* i pridodaje vrijednost kvalitete plijena nazivu plijena. Metoda i pomoću parametra *iconName* i metode *Resources.Load* učitava ikonu u igru. Metoda *Resources.Load* prima parametar putanje u mapi Resources i preko naziva ikone učitava resurs u scenu.[15]

Ikone oružja koje će se učitavati metodom *Resources.Load* uvezene su iz imovine „*FPS Icons Pack*“ iz *Asset Storea*, a ikone oklopa preuzete su na Web mjestu „game-icons.net“. Prije nego što se ikone mogu koristiti, svakoj ikoni se u prozoru *Inspector* mora promijeniti tip teksture u *Sprite*. Na sljedećoj slici je prikaz ikona u prozoru *Project* koje će se koristiti u izradi igre:



Slika 4 Ikone oružja i oklopa (Izvor: vlastita slika zaslona, 2022.)

Metoda *GenerateQuality* izrađena je na sljedeći način:

```
private int GenerateQuality()
{
    int quality;
    int chance = Random.Range(1, 101);
    if (chance <= 10)
    {
        quality = 5;
    }
    else if (chance > 10 && chance <= 25)
    {
        quality = 4;
    }
    else if (chance > 25 && chance <= 45)
    {
        quality = 3;
    }
    else if (chance > 45 && chance <= 70)
    {
            quality = 2;
    }
    else
    {
        quality = 1;
    }
    return quality;
}
```

U metodi su definirane lokalne varijable *quality* i *chance* tipa *int*. Varijabla *chance* prima vrijednost generiranog broja između jedan i sto. Brojevi se generiraju, kao i u ostatku koda, metodom *Random.Range* koja prima za prvi parametar donju uključenu granicu generiranog broja, a za drugi parametar gornju isključenu granicu generiranog broja.[16] Prema tome koji broj je generiran, strukturom *if* određuje se kvaliteta plijena metoda vraća taj podatak. Šanse za kvalitetu 5 su 10%, za kvalitetu 4 su 15%, za kvalitetu 3 su 20%, za kvalitetu 2 su 25% i za kvalitetu 1 su 30%.

Sljedeća metoda u klasi *Loot* bila bi metoda za generiranje svojstava, no ona se razlikuje između oružja i oklopa pa će se ta metoda definirati u potklasama klase *Loot*, ali u klasi Loot može se definirati klasa koja dodjeljuje dodatna svojstva prema kvaliteti. Za svaku razinu kvalitete, u listu *bonusStats*, instancirat će se jedna nasumična instanca tipa *Stat* pomoću statične metode *Stat.CreateRandomStat*, a to u kodu glasi:

```
public void GenerateBonusStats()

{
    for (int i = 0; i < quality; i++)
    {
        bonusStats.Add(Stat.CreateRandomStat());
    }
}
```

U klasi *Loot* potrebno je definirati još tri metode. One se u kodu definiraju na sljedeći način:

```
public static float AddRoundBonus(float statValue, float roundBonus)

{
    return statValue * (1f + roundBonus);
}

public string PrintLootDescription()
{
    string lootDescription = "";
    basicStats.ForEach(x => lootDescription += x.PrintStat() + "\n");
    lootDescription += "\nQuality bonus\n";
    bonusStats.ForEach(x => lootDescription += x.PrintStat() + "\n");
    return lootDescription;
}

public void DeleteStats()
{
        basicStats.ForEach(x => x.DeleteStat());
}
```

Statična metoda *AddRoundBonus* služi za preračunavanje vrijednosti svojstava plijena koje se povećavaju u svakoj novoj rundi i vraća tu vrijednost. Metoda prima parametre vrijednosti svojstva(*statValue*) i multiplikator za trenutnu rundu (*roundBonus*). Metoda *PrintLootDescription* vraća *string* svih opisa svojstva koje instanca plijena posjeduje pomoću metode *PrintStat* iz klase *Stat*. Metoda *DeleteStats* briše sva svojstva koje instanca posjeduje pomoću metode *DeleteStat* iz klase *Stat*.

5.3. Skripta *Weapon.cs*

Klasa *Weapon* nasljeđuje klasu *Loot* i uz sve što je definirano u klasi *Loot* sadrži i metodu *GenerateStats*. Metoda *GenerateStats* definirana je u skripti *Weapon.cs* na sljedeći način:

```
public class Weapon : Loot
{
    public void GenerateStats(float attackDamage, float criticalHit, float
    criticalChance,
    float fireRate, float reloadSpeed, float clipSize, float range)
    {
        basicStats.Add(Stat.CreateSpecificStat(StatType.AttackDamage,
        attackDamage));
        basicStats.Add(Stat.CreateSpecificStat(StatType.CritDamage,
        criticalHit));
        basicStats.Add(Stat.CreateSpecificStat(StatType.CritChance,
        criticalChance));
        basicStats.Add(Stat.CreateSpecificStat(StatType.FireRate,
        fireRate));
        basicStats.Add(Stat.CreateSpecificStat(StatType.ReloadSpeed,
        reloadSpeed));
        basicStats.Add(Stat.CreateSpecificStat(StatType.ClipSize,
        clipSize));
        basicStats.Add(Stat.CreateSpecificStat(StatType.Range, range));
        GenerateBonusStats();
    }
}
```

Metoda *GenerateStats* iz klase *Weapon* stvara sedam svojstava koje posjeduje svaki plijen koji je tipa oružje i dodaje ih u listu *basicStats*. Za parametre prima sedam vrijednosti tih svojstava. Na kraju metode generira dodatna svojstva pomoću metode *GenerateBonusStats*.

Klase koje nasljeđuju klasu *Weapon* definirane su u skripti *Weapon.cs*, a to su klase: *Handgun*, *Shotgun*, *AssaultRifle* i *SniperRifle*. Klasa *Handgun* definirana je na sljedeći način:

```
using UnityEngine;

public class Handgun : Weapon
{
    public static Handgun Generate(float roundBonus)
    {
        Handgun handgun = ScriptableObject.CreateInstance<Handgun>();
        handgun.GenerateLootProperties(LootType.Handgun,
        EquipmentType.Weapon, "Handgun", "handgun");

        float attackDamage = AddRoundBonus(200f, roundBonus);
        float criticalHit = 150f;
        float criticalChance = 20f;
        float fireRate = 2f;
        float reloadSpeed = 2f;
        float clipSize = 12f;
        float range = 40f;

        handgun.GenerateStats(attackDamage, criticalHit, criticalChance,
        fireRate, reloadSpeed, clipSize, range);

        return handgun;
    }
}
```

Sve četiri klase koje nasljeđuju klasu *Weapon* izrađene su na analogan način. Imaju statičnu metodu *Generate*. Metoda *Generate* za parametar prima vrijednost multiplikatora za trenutnu rundu. U metodi se stvara nova instanca plijena. Metodom *GenerateLootProperties* atributima se dodaju vrijednosti koje su specifične za tu potklasu oružja. Vrijednosti jačine napada statičnom metodom *AddRoundBonus* povećava se vrijednost ovisno o tome u kojoj rundi igre je plijen generiran. Zatim se definiraju vrijednosti sedam osnovnih svojstava oružja i dodaju se plijenu metodom *GenerateStats*. Metoda *Generate* na kraju vraća plijen.

5.4. Skripta *Armor.cs*

Klasa *Armor* nasljeđuje klasu *Loot* i uz sve što je definirano u klasi *Loot* sadrži i metodu *GenerateStats*. Metoda *GenerateStats* definirana je na sljedeći način:

```
using UnityEngine;

public class Armor : Loot
{
    public void GenerateStats(float health, float blockChance, float
    moveSpeed)
    {
        basicStats.Add(Stat.CreateSpecificStat(StatType.Health, health));
        basicStats.Add(Stat.CreateSpecificStat(StatType.BlockChance,
        blockChance));
        basicStats.Add(Stat.CreateSpecificStat(StatType.MoveSpeed,
        moveSpeed));
        GenerateBonusStats();
    }
}
```

Metoda *GenerateStats* iz klase *Armor* stvara tri svojstava koje posjeduje svaki plijen koji je tipa oklop i dodaje ih u listu *basicStats*. Za parametre prima tri vrijednosti tih svojstava. Na kraju metode generira dodatna svojstva pomoću metode *GenerateBonusStats*.

Klase koje nasljeđuju klasu *Armor* definirane su u skripti *Armor.cs*, a to su klase: *Helmet*, *UpperArmor*, *LowerArmor* i *Boots*. Klasa *Helmet* definirana je na sljedeći način:

```
public class Helmet : Armor
{
    public static Helmet Generate(float roundBonus)
    {
        Helmet helmet = ScriptableObject.CreateInstance<Helmet>();
        helmet.GenerateLootProperties(LootType.Helmet, EquipmentType.Head,
        "Helmet", "helmet");

        float health = AddRoundBonus(10f, roundBonus);
        float blockChance = 5f;
        float moveSpeed = 2.5f;

        helmet.GenerateStats(health, blockChance, moveSpeed);

        return helmet;
    }
}
```

Sve četiri klase koje nasljeđuju klasu *Armor* izrađene su na analogan način. Imaju statičnu metodu *Generate.* Metoda *Generate* za parametar prima vrijednost multiplikatora za trenutnu rundu. U metodi se stvara nova instanca plijena. Metodom *GenerateLootProperties* atributima se dodaju vrijednosti koje su specifične za tu potklasu oklopa. Vrijednosti životnih bodova statičnom metodom *AddRoundBonus* povećava se vrijednost ovisno o tome u kojoj rundi igre je plijen generiran. Zatim se definiraju vrijednosti tri osnovnih svojstava oklopa i dodaju se plijenu metodom *GenerateStats*. Metoda *Generate* na kraju vraća plijen.

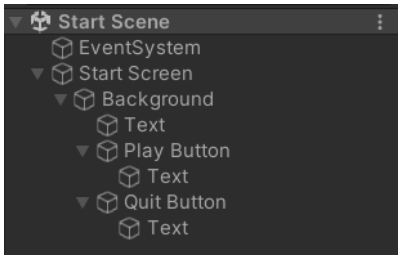
Razlog ovakve strukture plijena je da se sve vrijednosti jednog tipa plijena nalaze u na jednom mjestu i lako se mogu prilagoditi, ukoliko prve definirane vrijednosti plijena nisu bile izbalansirane za težinu igre ili se nekom atributu želi promijeniti naziv ili datoteka ikone.

6. Postavljanje scena

Scena(engl. *Scene*) u *Unityju* označava jedan dio igre u koji sadrži svoje pripadajuće objekte. Igre mogu sadržavati samo jednu scenu, a mogu i više, npr. jedna scena za svaku razinu igre.[17] U ovoj igri bit će izrađene tri scene. Prva scena će biti početna scena koja će se učitati na početku igre, u glavnoj sceni će biti smještena većina igre, a zadnja scena pojavit će se kad igrač izgubi igru. Početna scena prikazana je na sljedećoj slici.

Slika 5 Početni zaslon igre (Izvor: Vlastita slika zaslona, 2022.)

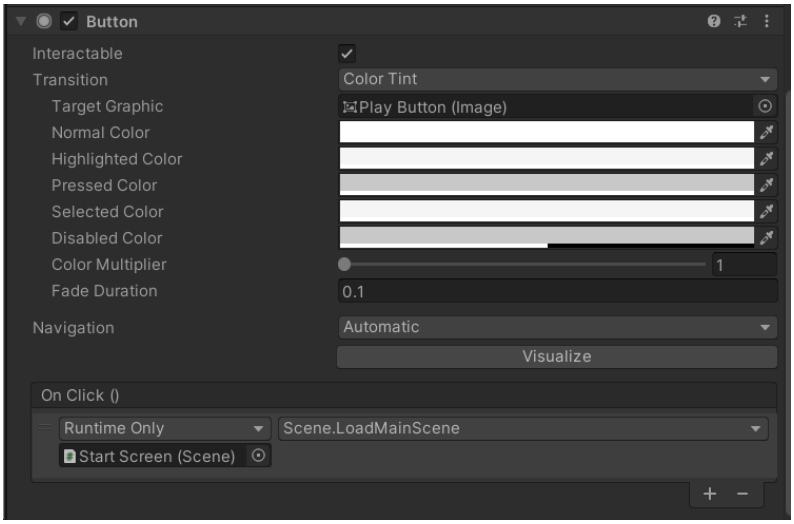
Na početnom zaslonu nalazi se tekst koji ukratko opisuje uputstva za igranje igre, gumb za početak igre i gumb za izlaz iz igre. Hijerarhiju objekata u sceni možemo vidjeti na sljedećoj slici:



Slika 6 Hijerarhija objekata početne scene (Izvor: Vlastita slika zaslona, 2022.)

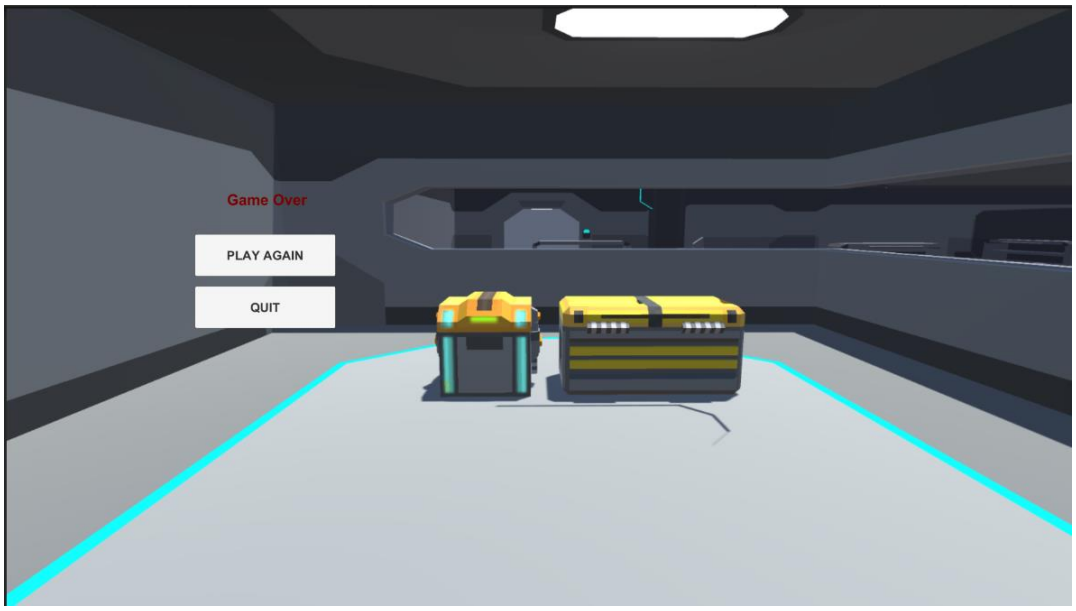
Početni zaslon sastoji se od objekta tipa *canvas* naziva *„Start Screen“*. *Canvas* je apstraktni prostor na kojem se prikazuju svi elementi sučelja.[18] Objektu *canvasa* dodana je komponenta *Scene.cs* gdje će se definirati metode za učitavanje scena i metoda za izlaz iz

igre. Objekt dijete *canvasu* će biti panel „*Background*“. Panel je *GameObject* koji u sebi sadrži komponentu slike i u ovom slučaju slika služi kao pozadina scene. Pozadina scene sadrži tekstualni objekt, gumb za početak igre i gumb za izlaz iz aplikacije. U tekstualnom objektu komponenta *Text* sadrži upute za igranje igre. Gumb za početak igre sadrži komponentu *Button* u kojoj je postavljen događaj koji klikom na gumb „*Play*“ iz skripte *Scene.cs* poziva metodu za učitavanje glavne scene. Gumb za izlaz iz igre postavljen je na jednaki način, jedino što događaj poziva metodu za izlaz iz igre. Komponenta *Button* od gumba za početak igre prikazana je na sljedećoj slici:



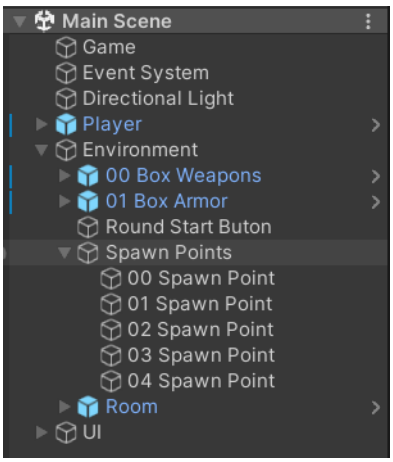
Slika 7 Prikaz komponente Button (Izvor: Vlastita slika zaslona, 2022.)

Na analogni način postavljeni je i završni zaslon igre. Razlika je samo u tekstu koji je prikazan na ekranu. Završni ekran možemo vidjeti na sljedećoj slici:



Slika 8 Završni zaslon igre (Izvor: Vlastita slika zaslona, 2022.)

U glavnoj sceni smještena je većina igre. Glavna scena sastoji se od objekata prikazanih na sljedećoj slici:



Slika 9 Hijerarhija objekata glavne scene (Izvor: Vlastita slika zaslona, 2022.)

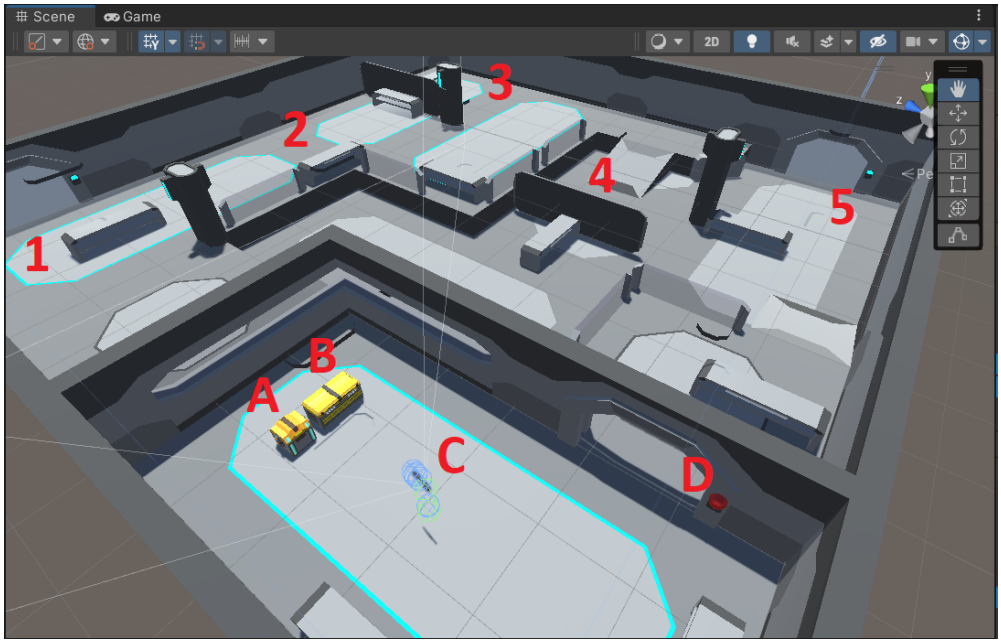
Objekt „*Game*“ je prazni objekt gdje smo dodali komponente *Scene.cs* i *Round.cs*. Objekt „*Player*“ je objekt igrača i on je na slici 9 označen slovom C, a objekt “UI“ je objekt u kojemu su smješteni svi objekti sučelja u glavnoj sceni. Ta dva objekta bit će razrađeni u narednim poglavljima. U praznom objektu „*Environment*“ nalaze se kutije s plijenom. Slovom A označena je kutija u kojoj se generiraju oružja, a slovom B označena je kutija u kojoj se generiraju oklopi. Za modele kutija korištena je imovina *„Pixel Boxes - low poly (Free)“* uvezena iz *Asset Storea*.

Slovom D označen je gumb koji igrač može pritisnuti za početak borbe. Gumb je 3D objekt u obliku sfere. Pritiskom na gumb pojavljuje se pet neprijatelja koje igrač mora pobijediti. Oni se pojavljuju na mjestima koji su na slici 9 označeni brojevima od 1 do 5, a u sceni su na tim mjestima dodani prazni objekti s tagom „*spawnPoint*“.

Svakom objektu igre se može dodati tag na taj način da se označi željeni objekt pa se u *Inspectoru* iz padajuće liste odabere već postojeći tag ili se na kraju padajuće liste odabere opcija „*Add tag*“. Klikom na tu opciju otvara se prozor za stvaranje novih tagova. Novostvoreni tagovi se prikazuju u padajućoj listi i mogu se odabrati.[19] Klasa *GameObject* sadrži metode *FindObjectWithTag* i *FindObjectsWithTag* koje za parametar primaju naziv taga i vraćaju tagirani objekt, odnosno niz s tagiranim objektima.[20]

Objekt „*Room*“ sadrži sve ostale objekte korištene za u izradi prostorije u kojoj se igrač može kretati. Kod kreiranja prostorije korištena je imovina *„Free LowPoly SciFi“* uvezena iz *Asset Storea* uz neke preinake. Obrisano je nekoliko objekata u prostoriji da se igrač i njegovi

neprijatelji lakše kreću po sobi i dodani su zidovi gdje su se prije nalazila otvorena vrata, zbog toga da igrač ne može izaći iz prostorije.



Slika 10 Objekti glavne scene (Izvor: Vlastita izrada, 2022.)

6.1. Skripta *Scene.cs*

Skripta *Scene.cs* sadrži klasu *Scene* koja sadrži metode za mijenjanje scena i kontrolu za izlaz iz igre. Metode za mijenjanje scene su metoda *LoadMainScene* koja učitava glavnu scenu i metoda *LoadEndScene* koja učitava zadnju scenu. Scene se učitavaju pomoću funkcije *SceneManager.LoadScene*.[21] Ta funkcija za parametar prima indeks scene. U slučaju ove igre početna scena ima redni indeks 0 ,glavna scena ima indeks 1, a zadnja scena ima indeks 2. Klasa *Scene* još sadrži metode *QuitGame* gdje se aplikacija gasi pritiskom na tipku *Esc* i *QuitGameOnButtonClick* koja samo poziva metodu za gašenje aplikacije. Kod klase *Scene* je sljedeći:

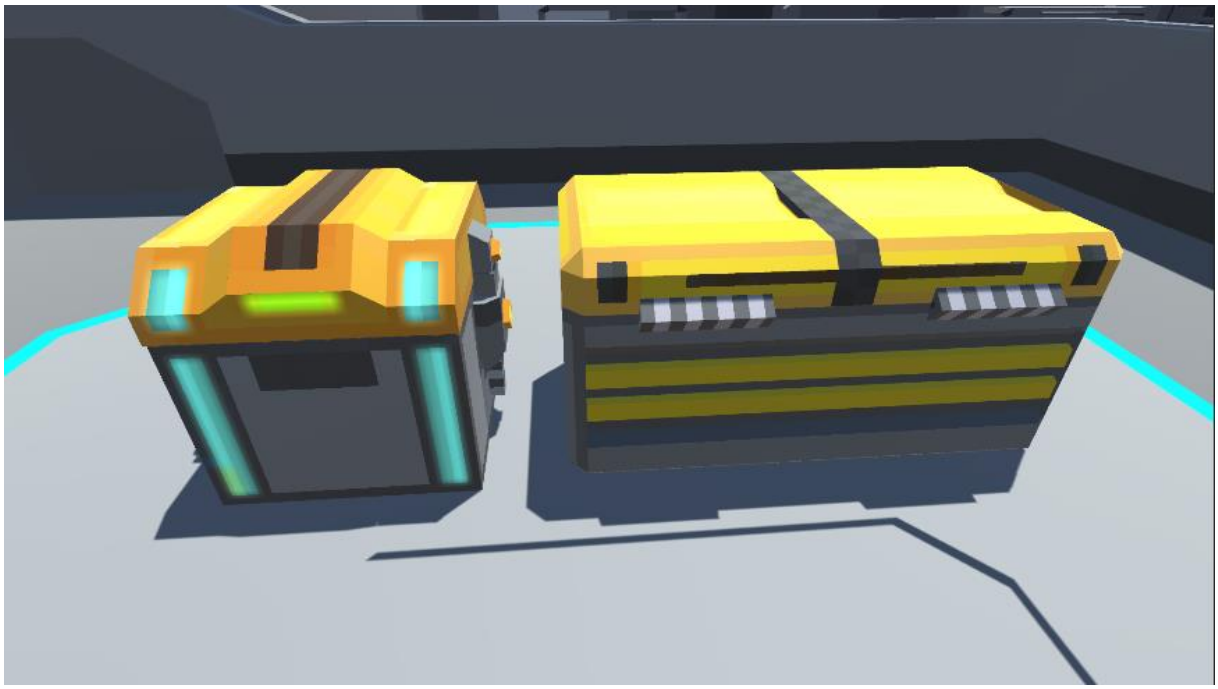
```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Scene : MonoBehaviour
{
    #region Singleton
    public static Scene instance;
    void Awake()
    {
```

```
        instance = this;
    }
    #endregion

    void Update()
    {
        QuitGame();
    }
    public void LoadMainScene() // Play Button, Play Again Button
    {
        SceneManager.LoadScene(1);
    }
    public void LoadEndScene()
    {
        PlayerControls.instance.UnlockMouse();
        SceneManager.LoadScene(2);
    }
    public void QuitGame()
    {
        if (Input.GetKey("escape"))
        {
            Application.Quit();
        }
    }
    public void QuitGameOnButtonClick() // Quit Buttons
    {
        Application.Quit();
    }
}
```

Klasa *Scene* definirana je kao *singleton* klasa. To znači da klasa uvijek ima samo jednu instancu i globalno je dostupna u kodu.[22]. Način na koji je definirana svaka *singleton* klasa u kodu igre prikazan je na primjeru klase. Svaka *singleton* klasa u kodu ove igre nasljeđuje klasu *MonoBehaviour*. U klasi je definiran statični atribut naziva instance koji je istog tipa kao i klasa te je u metodi *Awake* atributu instance pridružena sama klasa. Metoda *Awake* se izvodi jednom, prilikom učitavanja skripte, i izvodi se prije metode *Start*.[23]

7. *Kutije s plijenom*

U glavnoj sceni smo postavili dvije kutije kako je prikazano na slici 11. U lijevoj kutiji generirat će se plijen s oružjem, a u desnoj plijen s oklopima. U komponente svake kutije dodat ćemo skriptu *Box.cs*.



Slika 11: Kutije s plijenom (Izvor: Vlastita slika zaslona, 2022.)

7.1. Skripta *Box.cs*

Klasa *Box* definirana je u skripti *Box.cs*. U klasi *Box* definirani su atributi *generatedEquipmentType* koji će određivati tip plijena koji će se generirati u kutiji, *generatedEquipmentCount* koji će određivati koliko instanci plijena će se generirati, lista plijena loot u koju će se plijen generirati i atribut *isGenerated* koji će označavati je li u trenutnoj rundi već generiran plijen u kutiji ili se tek mora generirati.

Klasa *Box* sadrži metodu za generiranje plijena u kutiji. Metoda za generiranje plijena u kutiji je *GenerateLootInBox*. Kod za tu metodu je:

```
public void GenerateLootInBox()
{
    if (!isGenerated)
    {
        generatedLootCount = Mathf.Clamp(generatedLootCount, 0,
        12);

        loot.ForEach(x => x.DeleteStats());
        loot.ForEach(x => Destroy(x));
        loot.Clear();

        for (int i = 0; i < generatedLootCount; i++)
        {
            loot.Add(GenerateLootByEquipmentType(generatedEquipmentType));
        }
        isGenerated = true;
    }
```

Ova metoda generira plijen u kutiji samo ako je vrijednost varijable *isGenerated false*. Broj generiranih instanci plijena ograničava se na broj između 0 i 12 ukoliko se u inspektoru za atribut *generatedLootCount* unese manji ili veći broj. Nakon toga se sva svojstva plijena u kutiji brišu, pa se briše sav plijen u kutiji, pa se čisti lista plijena u kutiji. Zatim se petljom for u listu plijena generira novi plijen metodom *GenerateLootByEquipmentType* onoliko puta koliko smo to odredili atributom *generatedLootCount*. Na kraju metode atribut *isGenerated* postavlja se na *true*.

Budući da će se u sceni nalaziti dvije kutije, jedna sa oružjem i jedna sa oklopima, potrebna je jedna metoda koja će za svaku kutiju generirati plijen upravo za jedan od ta dva tipa opreme. Tome služi metoda *GenerateLootByEquipmentType*. Definirana je na sljedeći način:

```
    private Loot GenerateLootByEquipmentType(EquipmentType equipmentType)
    {
        int lootType;
        if (equipmentType == EquipmentType.Weapon)
        {
            lootType = Random.Range(0, 4);
        }
        else
        {
            lootType = Random.Range(4, 8);
        }
        return GenerateLootByLootType((LootType)lootType);
    }
```

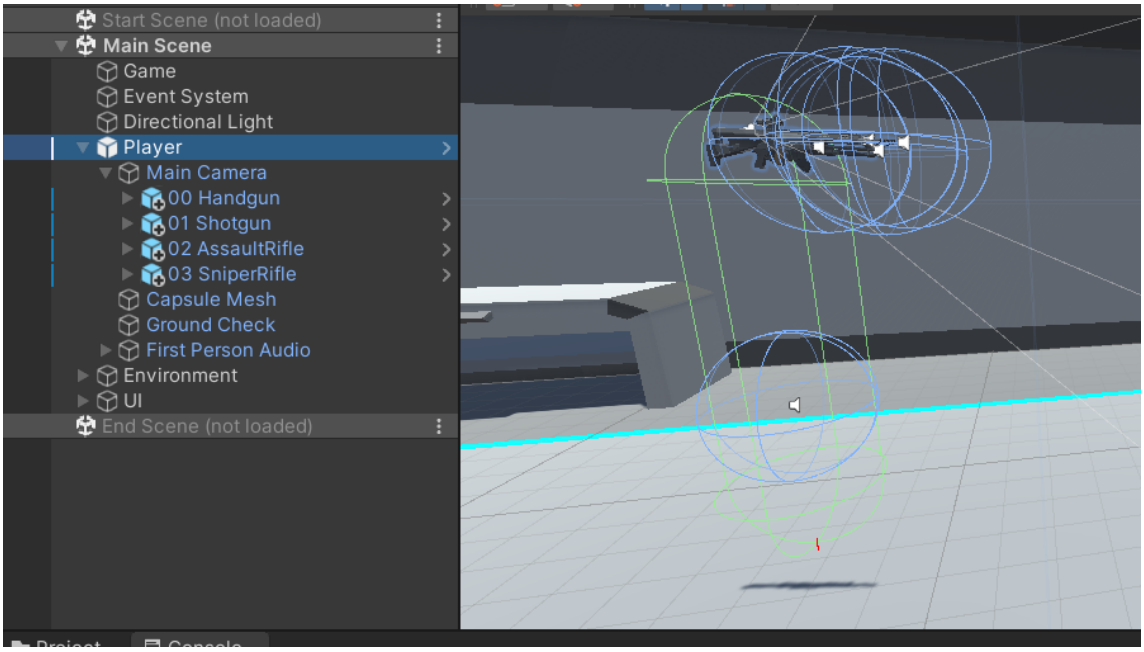
Ova metoda vraća podatak tipa *Loot*. Na početku metode definira se varijabla *lootType* u koji će se generirati nasumični broj. Ukoliko je ulazni parametar tip opreme *Weapon* generirat će se brojevi od 0 do 3, a ako nije, generirat će se brojevi od 4 do 7. Ti brojevi predstavljaju vrijednosti enumeracije *LootType* i prema tim parametrima, vratit će se generirani plijen metodom *GenerateLootByType*. Kod metode *GenerateLootByType* je sljedeći:

```
private Loot GenerateLootByLootType(LootType lootType)
{
    float roundBonus = RoundManager.instance.currentRound * 0.1f;
    switch (lootType)
    {
        case LootType.Handgun:
            return Handgun.Generate(roundBonus);
        case LootType.Shotgun:
            return Shotgun.Generate(roundBonus);
        case LootType.AssaultRifle:
            return AssaultRifle.Generate(roundBonus);
        case LootType.SniperRifle:
            return SniperRifle.Generate(roundBonus);
        case LootType.Helmet:
            return Helmet.Generate(roundBonus * 2f);
        case LootType.UpperArmor:
            return UpperArmor.Generate(roundBonus * 2f);
        case LootType.LowerArmor:
            return LowerArmor.Generate(roundBonus * 2f);
        default:
            return Boots.Generate(roundBonus * 2f);
    }
}
```

Ova metoda prima parametar tipa *LootType*, odnosno tip plijena. Na početku metode definirana je varijabla *roundBonus* koja pridružuje vrijednost trenutne runde iz klase *RoundManager* i množi ga sa 0,1 da se dobije multiplikator kojim množimo neka svojstva plijena koja se pojačavaju svake runde. To znači da svake runde jačina napada u osnovnim svojstvima plijena raste za 10% od originalnog iznosa, a životni bodovi se povećavaju za 20% svaku rundu jer se varijabla *roundBonus* množi s 2. Nakon što je određena varijabla *roundBonus*, u *switch* strukturi prema parametru se generira i vraća plijen određenog tipa.

8. Igrač

U ovom poglavlju objasnit ćemo objekt igrača(engl. *Player*) u sceni i skripte koje su pridružene kao komponente igrača. Na slici u nastavku vidimo kako igrač izgleda u sceni i njegove komponente.

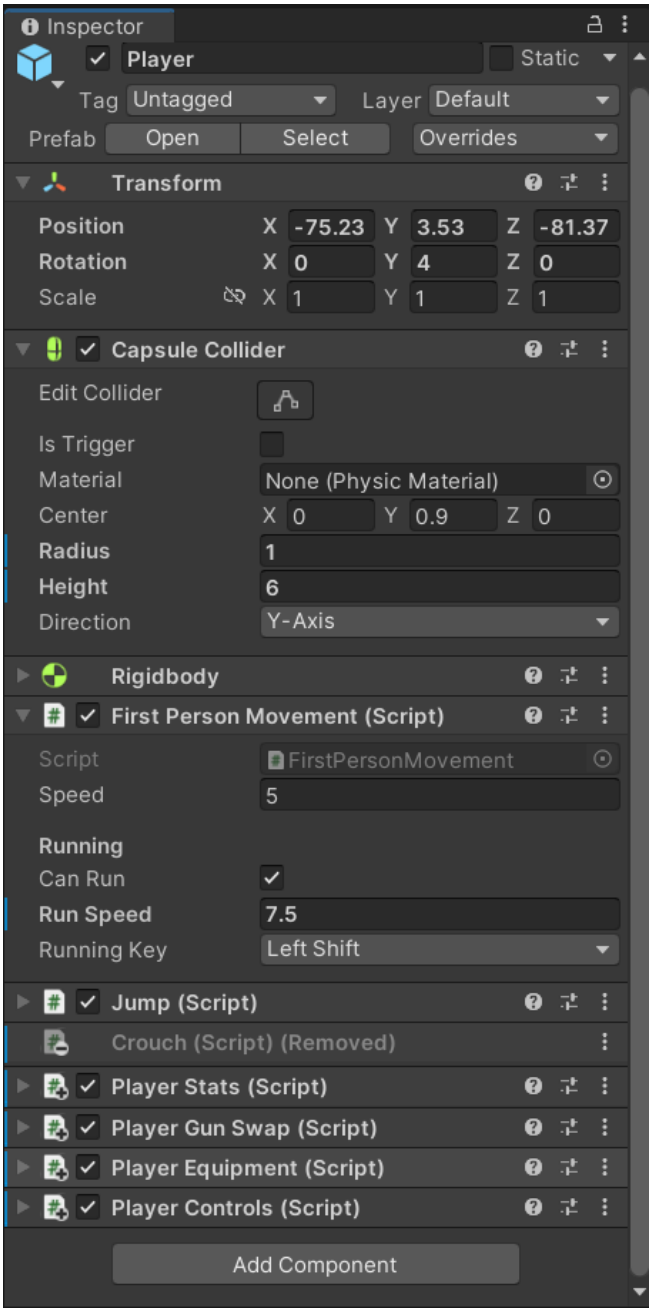


Slika 12: Igrač u sceni (Izvor: Vlastita slika zaslona, 2022.)

Objekt *Player* je prazni objekt s komponentom *Capsule Collider* i *Character Controller*. *Collideri* služe da se objekti mogu međusobno sudarati.[24] U ovom slučaju služi da igrač u igri ne prođe kroz zid, ili da ne propadne kroz pod. Komponenta *Character Controller* služi za upravljanje likom.[25] Objekti djeca igraču su *„Main Camera“* što je glavna kamera, *„Capsule Mesh“* što je 3D model igrača, no on će biti isključen jer nije potreban, jedan prazni objekt naziva *„Ground Check“*, i objekt *„First Person Audio“* koji sadrži sve zvukove kretanja igrača.

Za stvaranje objekta *Player*, korištena je imovina *„Mini First Person Controller“* uvezena iz *Asset Storea* koji dolazi sa već gotovim skriptama za kretanje igrača(*FirstPersonMovement.cs*), skakanje(*Jump.cs*), čučanje(*Crouch.cs*) koji su komponente objekta *Player*, skriptom za pogled i rotiranje igrača(*FirstPersonLook.cs*) koja je komponenta objekta kamere, i skriptom *GroundCheck.cs* koja služi za provjeru je li igrač na tlu ili u zraku, a komponenta je objekta *Ground Check*. Uz već naveden skripte, igraču su dodane još četiri skripte vlastite izrade, a to su *PlayerEquipment.cs*, *PlayerStats.cs*, *PlayerGunSwap.cs* i *PlayerControls.cs*. Uz to, promijenjeni su veličina i radijus *Capsule Collidera*, prilagođena je

početna brzina kretanja i trčanja igrača i uklonjena je skripta *Crouch.cs* jer u ovoj igri neće biti potrebna. Promjene u *Inspectoru* možemo vidjeti na slici 6.

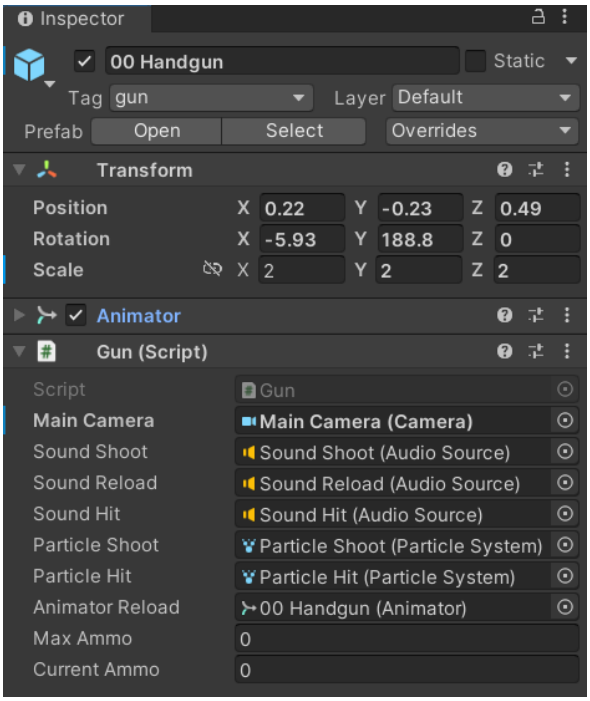


Slika 13: Komponente objekta igrača (Izvor: vlastita slika zaslona, 2022)

Objekti djeca kamere su četiri objekta oružja. Objekti oružja se aktiviraju ili deaktiviraju ovisno o tome koji tip oružja se nalazi u igračevoj opremi. Za modele oružja korištena je imovina *„Low Poly Weapons VOL.1“* uvezena iz *Asset Storea*. Objektima oružja su kao objekti djeca dodani objekti koji sadrže zvukove pucnja, punjenja oružja i pogotka, te objekti s čestičnim efektima blijeska pucnja i pogotka. Objekti sa zvukovima i čestičnim efektima su vlastite izrade no sami zvukovi su preuzeti sa Web mjesta *Pixabay*. Hijerarhiju objekta oružja možemo vidjeti na primjeru objekta pištolja (slika 14). Svi objekti oružja izrađeni su po jednakom principu, jedino su zvukovi pucnja drugačiji za svako oružje.

Slika 14: Primjer objekta oružja (Izvor: vlastita slika zaslona , 2022.)

Svakom objektu oružja dodana je komponenta Animator koji sadrži animaciju za punjenje oružja koja je vlastite izrade, i skriptu *Gun.cs*. Skripta *Gun.cs* detaljnije će biti objašnjena u poglavlju „Borba“. Komponente oružja možemo vidjeti na slici 15.



Slika 15: Komponente objekta pištolja (Izvor: vlastita slika zaslona, 2022.)

8.1. Skripta *PlayerEquipment.cs*

Klasa *PlayerEquipment* iz skripte *PlayerEquipment.cs* će biti definirana kao *singleton* klasa. U klasi *PlayerEquipment* definirana je lista plijena koja sadrži pet elemenata. Svaki element predstavlja jedno mjesto u opremi. Elementi opreme se u metodi *Start* postavljaju na prazne elemente. To je definirano sljedećim kodom:

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

public class PlayerEquipment : MonoBehaviour
{
    #region Singleton (…)

    public List<Loot> equipment;

    void Start()
    {
        equipment = new List<Loot>()
        {
            null, null, null, null, null
        };
    }
```

Klasa *PlayerEquipment* ima definirana još dvije metode, a to su *EquipLoot* i *UnequipLoot*. Prikazane su sljedećim kodom:

```
public void EquipLoot(Loot chosenLoot)
{
    UI.instance.boxInUI.loot.Remove(chosenLoot);
    if (equipment[(int)chosenLoot.equipmentType] != null)
    {
        UnequipLoot(equipment[(int)chosenLoot.equipmentType]);
    }
    equipment[(int)chosenLoot.equipmentType] = chosenLoot;
}

public void UnequipLoot(Loot chosenLoot)
{
    int boxLootCount = UI.instance.boxInUI.loot.FindAll(x => x !=
    null).Count;
    if (boxLootCount < 12)
    {
        UI.instance.boxInUI.loot.Add(chosenLoot);
        equipment[(int)chosenLoot.equipmentType] = null;
    }
 }
```

Metoda *EquipLoot* premješta odabrani plijen iz kutije na mjesto u opremi koje je za to predviđeno. Prvo se odabrani plijen uklanja iz kutije koja je otvorena u sučelju, zatim se plijen istog tipa iz opreme vraća u kutiju ako ga ima u opremi, pa se na to mjesto u opremi postavlja odabrani plijen.

Metoda *UnequipLoot* vraća odabrani plijen u opremi u kutiju ako u kutiji nema već 12 instanci plijena, a to radi tako da u kutiju koja je prikazana u sučelju dodaje odabrani plijen iz opreme, a na mjestu opreme gdje je bio odabrani plijen, postavlja plijen na *null*.

8.2. Skripta *PlayerStats.cs*

U skripti *PlayerStats.cs* definirana je klasa *PlayerStats*. Klasa *PlayerStats* sadrži funkcije za preračunavanje i dodavanje vrijednosti svojstava plijena igraču. Ta igračeva svojstva su zapravo ukupne vrijednosti svojstava plijena kojeg igrač trenutno koristi. Igračeva svojstva se preračunavaju svaki put kad igrač pokupi, odbaci ili zamijeni plijen. Klasa *PlayerStats* definirana je sljedećim kodom:

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

public class PlayerStats : MonoBehaviour
{
    #region Singleton (...)

    public float currentHealth, maxHealth;
    public List<Stat> stats;

    void Start()
    {
        stats = new List<Stat>()
        {
            Stat.CreateSpecificStat(StatType.AttackDamage, 0f),
            Stat.CreateSpecificStat(StatType.CritDamage, 0f),
            Stat.CreateSpecificStat(StatType.CritChance, 0f),
            Stat.CreateSpecificStat(StatType.FireRate, 0f),
            Stat.CreateSpecificStat(StatType.ReloadSpeed, 0f),
            Stat.CreateSpecificStat(StatType.ClipSize, 0f),
            Stat.CreateSpecificStat(StatType.Range, 0f),
            Stat.CreateSpecificStat(StatType.Health, 100f),
            Stat.CreateSpecificStat(StatType.BlockChance, 0f),
            Stat.CreateSpecificStat(StatType.MoveSpeed, 5f)
        };
    }
```

Klasa *PlayerStats* definirana je kao *singleton* klasa. Sadrži atribute *currentHealth* i *maxHealth* koji spremaju vrijednosti igračevih trenutnih i maksimalnih životnih bodova. Klasa sadrži i listu svih mogućih igračevih svojstava i vrijednosti tih svojstava postavljeni su na nulu, osim svojstva tipa Health koji je postavljen na vrijednost 100 jer igrač ima minimalno 100 maksimalnih životnih bodova i svojstva *MoveSpeed* jer najmanja brzina kretanja igrača je vrijednosti 5. U funkciji *Start* kreira se lista sa početnim svojstvima igrača.

U klasi *PlayerStats* definirana je funkcija *CalculateStats* i prikazana je sljedećim kodom:

```
public void CalculateStats()
{
    ResetStats();
    CalculateNewStats();
    maxHealth = stats[(int)StatType.Health].value;
    ResetCurrentHealth();
    this.gameObject.GetComponent<FirstPersonMovement>().speed =
    stats[(int)StatType.MoveSpeed].value;
    this.gameObject.GetComponent<FirstPersonMovement>().runSpeed =
    stats[(int)StatType.MoveSpeed].value * 1.5f;
}
```

Funkcija *CalculateStats* resetira sva svojstva igrača na početne vrijednosti metodom *ResetStats*, dodaje nove vrijednosti svojstava metodom *CalculateNewStats* i postavlja atribut *maxHealth* na novu vrijednost igračevih maksimalnih životnih bodova, izjednačava trenutne životne bodove s maksimalnim životnim bodovima u metodi *ResetCurrentHealth* i na kraju izjednačava brzinu kretanja igrača s novom vrijednosti svojstva tipa *MoveSpeed*. Navedene metode definirane su u klasi *PlayerStats* sljedećim kodom:

```
private void ResetStats()
{
    stats.ForEach(x => x.value = 0f);
    stats[(int)StatType.Health].value = 100f;
    stats[(int)StatType.MoveSpeed].value = 5f;
}

private void CalculateNewStats()
{
    foreach (Loot loot in PlayerEquipment.instance.equipment)
    {
        if (loot != null)
        {
            loot.basicStats.ForEach(x => AddStatValueToPlayer(x));
            loot.bonusStats.ForEach(x => AddStatValueToPlayer(x));
        }
    }
}
```

```
private void AddStatValueToPlayer(Stat lootStat)
{
    Stat playerStat = stats.Find(x => x.statType == lootStat.statType);
    playerStat.value += lootStat.value;
}
public void ResetCurrentHealth()
{
    currentHealth = maxHealth;
}
```

8.3. Skripta PlayerGunSwap.cs

U skripti *PlayerGunSwap.cs* definirana je *singleton* klasa *PlayerGunSwap*. Sadrži funkcije koje služe za aktivaciju i deaktivaciju objekata koji u igri predstavljaju oružje. ili Deaktivacijom objekta onemogućuju se sve komponente tog objekta i sve komponente objekata djece tog objekta. Stanje objekta se može promijeniti metodom *SetActive* koja prima parametar *true* za aktivaciju ili *false* za deaktivaciju objekta.[26] Funkcije klase *PlayerGunSwap* koriste se svaki put kada igrač iz kutije pokupi plijen koji je oružje ili kada igrač stavi oružje nazad u kutiju. Klasa *PlayerGunSwap* definirana je na sljedeći način:

```
public class PlayerGunSwap : MonoBehaviour
{
    #region Singleton (...)

    public Gun activeGun = null;
    private List<GameObject> guns;

    void Start()
    {
        guns = GameObject.FindGameObjectsWithTag("gun").ToList();
        guns = guns.OrderBy(x => x.transform.name).ToList();
        DeactivateWeapons();
    }
```

Ova klasa sadrži atribut *activeGun* koji sadrži referencu objekta oružja koji je trenutno aktivan, a na početku igre igrač ne posjeduje ni jedno oružje pa je vrijednost atributa null. Atribut *activeGun* je tipa *Gun* koji je definiran u skripti *Gun.cs* koja će biti objašnjena u poglavlju Borba. U klasi *PlayerGunSwap* definirana je lista objekata *guns*. U metodi *Start* u listu *guns* dodaju se svi objekti koji sadrže tag naziva „*gun*“, lista se sortira prema nazivu objekata i svi objekti u listi *guns* se deaktiviraju.

U igri koju ćemo izraditi postoje četiri tipa oružja i u igri su prikazani sa četiri objekta tagiranih sa tagom „*gun*“. Nazivi objekta su *„00 Handgun*“, *„01 Shotgun“*, *„02 AssaultRifle“* i *„03 SniperRifle“*. Ta četiri objekta tijekom igre referencirani su u listi *guns*.

Metode za aktivaciju i deaktivaciju oružja u klasi *PlayerGunSwap* definirane su prema sljedećem kodu:

```
public void ActivateWeapon(LootType lootType)
{
    DeactivateWeapons();
    activeGun = guns[(int)lootType].GetComponent<Gun>();
    guns[(int)lootType].SetActive(true);
    activeGun.ConfigureWeapon();
}
public void DeactivateWeapons()
{
    guns.ForEach(x => x.SetActive(false));
    activeGun = null;
}
```

Metoda *DeactivateWeapons* deaktivira sve *GameObjecte* u listi *guns* i postavlja vrijednost varijable *activeGun* na *null*. Metoda *ActivateWeapons* prima parametar *lootType*. Prvo deaktivira sve objekte u listi *guns*, zatim prema tipu plijena u atribut *activeGun* sprema komponentu klase *Gun* i aktivira *GameObject* koji je tog tipa plijena i na kraju poziva metodu *ConfigureWeapon* iz klase *Gun*.

8.4. Skripta *PlayerControls.cs*

U skripti *PlayerControls.cs* definirana je klasa *PlayerControls*. Definirana je kao *singleton* klasa. Klasa sadrži funkcije koje su potrebne za kontrolu pucanja i punjenja oružja, interakciju s objektima te otvaranje i zatvaranje sučelja. Klasa *PlayerControls* i njeni atributi definirani su sljedećim kodom:

```
using UnityEngine;

public class PlayerControls : MonoBehaviour
{
    #region Singleton (...)

    public bool uiActive = false;
    public Camera mainCamera;
```

Klasa *PlayerStats* sadrži metodu *Update* koja sadrži nekoliko metoda koje se tijekom igre konstantno izvršavaju. Ukoliko nije otvoreno sučelje koje prikazuje opremu igrača ili sučelje koje prikazuje sadržaj škrinje, izvršavaju se metode *Shoot* i *Reload*. Uvijek se izvršavaju metode *Interact*, *OpenEquipment* i *CloseEquipmentAndBox*. Atribut *uiActive* je na početku igre postavljen na *false* jer nije otvoreno nijedno od navedenih sučelja. Metoda je prikazana je sljedećim kodom:

```
void Update()
{
    if (!uiActive)
    {
        Shoot();
        Reload();
    }
    Interact();
    OpenEquipment();
    CloseEquipmentAndBox();
}
```

Metoda *Shoot* poziva drugu metodu *Shoot* iz klase *Gun* ako je pritisnuta kontrola *„Fire1“* što je u ovom slučaju lijevi klik miša i ako igrač u ruci drži oružje. Metoda *Shoot* definirana je sljedećim kodom:

```
private void Shoot()
{
    if (Input.GetButton("Fire1") && PlayerGunSwap.instance.activeGun !=
    null)
    {
        PlayerGunSwap.instance.activeGun.Shoot();
    }
}
```

Metoda *Reload* poziva drugu metodu *Reload* iz klase *Gun* ako je pritisnuta kontrola *„Fire2“* što je u ovom slučaju desni klik miša i ako igrač u ruci drži oružje. Metoda *Reload* definirana je sljedećim kodom:

```
private void Reload()
{
    if (Input.GetButton("Fire2") && PlayerGunSwap.instance.activeGun !=
    null)
    {
        PlayerGunSwap.instance.activeGun.Reload();
    }
}
```

Metoda *Interact*, pritiskom na tipku E na tipkovnici, stvara se *ray* ,tj. zraka koja počinje na objektu kamere i završava na trenutnoj poziciji kursora. U varijablu *hit* spremaju se

informacije o objektu pogođenom sa zrakom na udaljenosti 5 od kamere i to metodom *Physics.Raycast*.[27] Ukoliko je zraka pogodila objekt provjerava se sadržava li pogođeni objekt komponentu *Box* ili komponentu *RoundButton* metodom *TryGetComponent*.

Ukoliko pogođeni objekt sadržava komponentu *Box*, tj. ako je pogođena kutija s plijenom, poziva metoda za generiranje plijena u kutiji, prikazuje se sučelje te kutije i poziva se metoda *UnlockMouse*. Ukoliko je pogođeni objekt sadrži komponentu *RoundButton*, tj ako je pogođen gumb za početak runde, poziva se metoda *StartRound* u klasi *RoundButton*.

Metoda *Interact* definirana je na sljedeći način:

```
private void Interact()
{
    if (Input.GetKeyDown(KeyCode.E))
    {
        Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, 5))
        {
            Box box;
            if (hit.transform.TryGetComponent<Box>(out box))
            {
                box.GenerateLootInBox();
                UI.instance.ShowBox(box);
                UnlockMouse();
            }
            RoundButton roundButton;
            if (hit.transform.TryGetComponent<RoundButton>(out
            roundButton))
            {
                    roundButton.StartRound();
            }
        }
    }
}
```

Klasa *PlayerControls* sadrži i kontrole za otvaranje sučelja opreme i zatvaranje sučelja opreme i kutije s plijenom. Sučelje opreme prikazuje se pritiskom na tipku *Tab*. Ako je pritisnuta ta tipka poziva se metoda za prikaz sučelja opreme iz klase UI i otključava se miš. Sučelje opreme i sučelje kutije s plijenom sakriva se pritiskom na tipku Q. Ako je pritisnuta ta tipka poziva se metoda za skrivanje sučelja opreme i kutije s plijenom iz klase UI i zaključava se miš. Kod za ove dvije metode je sljedeći:

```
private void OpenEquipment()
{
    if (Input.GetKeyDown(KeyCode.Tab))
```

```
    {
        UI.instance.ShowEquipment();
        UnlockMouse();
    }
}
private void CloseEquipmentAndBox()
{
    if (Input.GetKeyDown(KeyCode.Q))
    {
        UI.instance.HideEquipmentAndBox();
        LockMouse();
    }
}
```

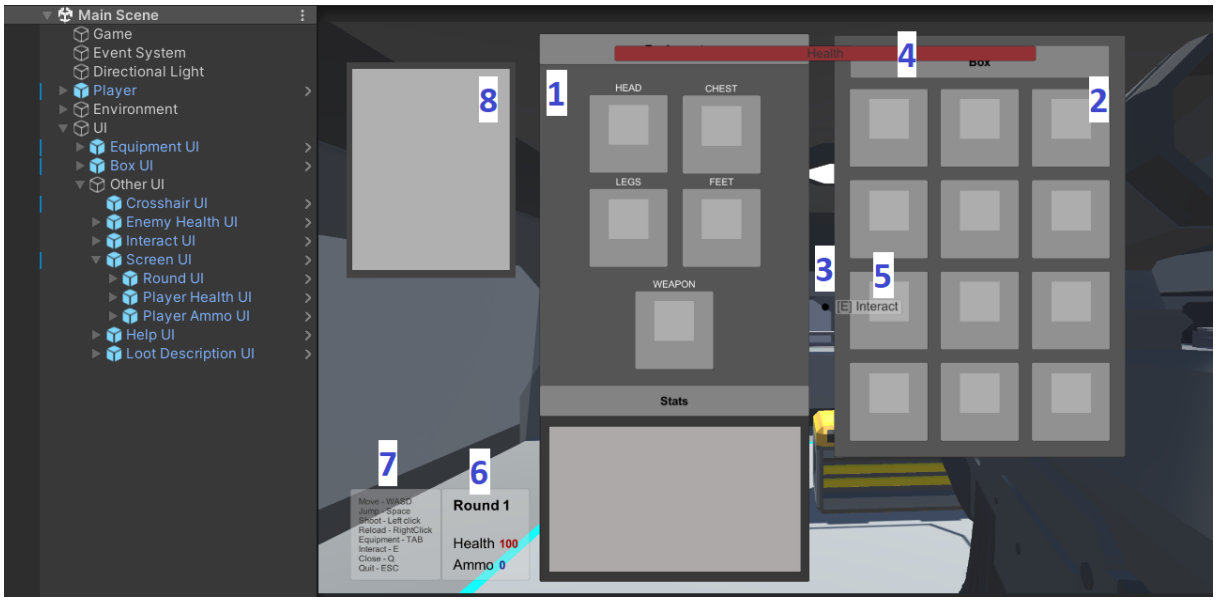
U klasi *PlayerControls* za definirati još su ostale metode za otključavanje i zaključavanje miša. Kod za ove metode je sljedeći:

```
public void UnlockMouse()
{
    GameObject.FindGameObjectWithTag("MainCamera").GetComponent<FirstPerson
    Look>().sensitivity = 0f;
    this.gameObject.GetComponent<FirstPersonMovement>().enabled = false;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    uiActive = true;
}
public void LockMouse()
{
    GameObject.FindGameObjectWithTag("MainCamera").GetComponent<FirstPerson
    Look>().sensitivity = 2f;
    this.gameObject.GetComponent<FirstPersonMovement>().enabled = true;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
    uiActive = false;
}
```

Metoda *UnlockMouse* omogućuje korištenje kursora kad su otvorena sučelje opreme i sučelje kutije s plijenom, bez da se igrač pomiče u prostoru ili da se rotira kamera. Pomicanje kamere spriječeno je na način da se osjetljivost miša postavi na nulu. Kretanje igrača spriječeno je onemogućavanjem komponente *FirstPersonMovement* u kojoj su definirane metode za kretanje igrača. Uz to stanje kursora postavljeno je na None, tj. kursor nije zaključan na centar zaslona, vidljivost kursora se postavlja na *true* i varijabla *uiActive* koja nam označuje je li sučelje otvoreno, postavlja se na *true*. Metoda *LockMouse* sve varijable promijenjene u metodi *UnlockMouse* vraća na početne vrijednosti.

9. Sučelje glavne scene

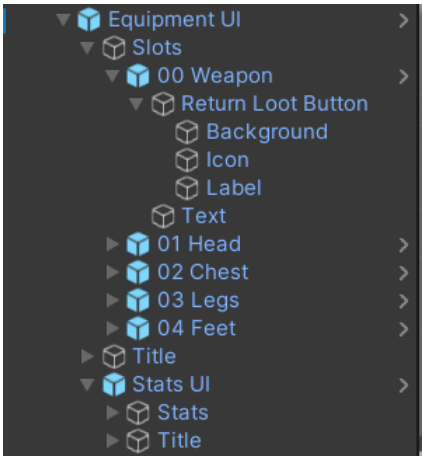
Sučelje glavne scene prikazano je na sljedećoj slici:



Slika 16 Sučelje glavne scene (Izvor: vlastita izrada, 2022.)

Sučelje opreme(*Equipment UI*) je označeno brojem 1. Sučelje kutije s plijenom(*Box UI*) je označeno brojem 2. Brojem 3 označeno je sučelje nišana(*Crosshair UI*) koji se sastoji od jednog panela crne boje i izvorna slika ima oblik kruga. Brojem 4 označeno je sučelje protivničkih životnih bodova(*Enemy Health UI*). Sastoji se od panela crvene boje teksta koji prikazuje trenutne protivničke životne bodove. Taj objekt je vidljiv samo kad igrač drži nišan direktno na protivniku. Širina panela sa životnim bodovima ovisi o postotku preostalih životnih bodova protivnika. Brojem 5 je označen objekt naziva *„Interact UI“* koji obavještava igrača da može stupiti u interakciju s objektom u igri. Brojem 6 označeni su objekti sučelja koji igrača obavještavaju o broju trenutne runde(*Round UI*), trenutnim životnim bodovima igrača(*Player Health UI*) i broju preostalih metaka prije punjenja oružja (*Ammo UI*). Brojem 7 označen je panel s uputama za igranje (*Help UI*). Brojem 8 označen je panel koji pokazuje tekst svojstava plijena(*Loot Description UI*).On se uvijek prikazuje direktno ispod kursora i vidljiv je samo kada igrač kursorom prođe iznad plijena.

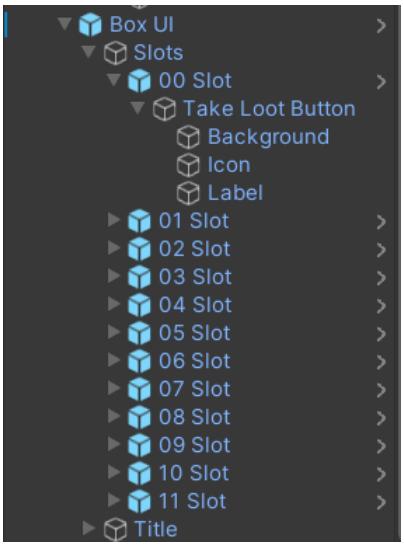
Hijerarhijski prikaz objekata sučelja opreme prikazan je na sljedećoj slici:



Slika 17 Hijerarhija sučelja opreme (Izvor: vlastita slika zaslona, 2022.)

Djeca objekti sučelja opreme su sljedeći paneli: naslov, 5 pozicija(engl. *Slot*) na kojima prikazuje plijen u opremi za svaki tip opreme, i panel koji pokazuje vrijednosti igračevih ukupnih svojstva kada se pribroje vrijednosti svojstva s svakog plijena u opremi.

Sličnu strukturu ima i sučelje kutije. Razlika je u tome što kutija ima 12 pozicija na kojima može biti plijen i nema igračeva ukupna svojstva. Strukturu sučelja kutije vidimo na sljedećoj slici:

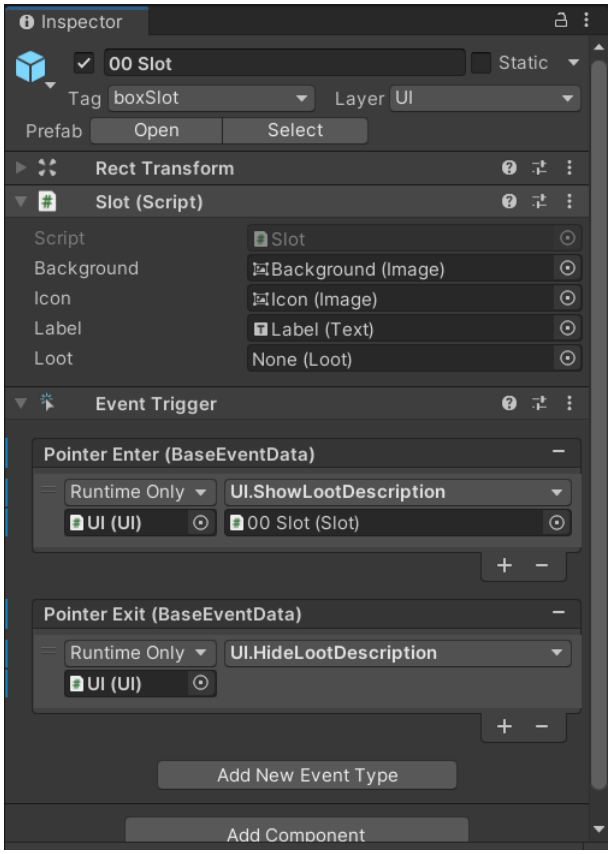


Slika 18 Hijerarhija sučelja kutije (Izvor: vlastita slika zaslona, 2022.)

Iz prethodne dvije slike možemo primijetiti da pozicija u sučelju opreme ima gumb za vraćanje plijena u kutiju(*Return Loot Button*), a pozicija u sučelju kutije ima gumb za pokupljanje plijena iz kutije u opremu(*Take Loot Button*). Na gumbe za vraćanje plijena postavljen je *onClick* događaj koji klikom na taj gumb poziva metodu *ReturnLoot* iz klase *Slot*,

a na gumbe za pokupljanje plijena postavljen je *onClick* događaj koji klikom na taj gumb poziva metodu *TakeLoot* iz klase *Slot*. Da bi događaji mogli raditi, u sceni mora biti točno jedan objekt *EventSystem* koji procesuira i rukuje događajima.[28]

Objekt „*Background*“ mijenja boju prema tome kakve je kvalitete plijen na toj poziciji. U objekt „*Icon*“ postavlja se odgovarajuća ikona za taj plijen. U objekt „*Label*“ upisuje se naziv plijena na toj poziciji. Ta tri objekta referenciraju se na skriptu *Slot.cs* koja je komponenta pozicije. Svaka pozicija ima i dva događaja, *onPointerEnter*, koji poziva metodu za prikaz opisa plijena kada kursor prođe iznad sučelja pozicije, i *onPointerExit*, koji poziva metodu za skrivanje opisa plijena kada kursor više nije iznad pozicije. Komponente pozicije možemo vidjeti na primjeru sljedeće slike:



Slika 19 Komponente sučelja pozicije (Izvor: vlastita slika zaslona, 2022.)

9.1. Skripta *Slot.cs*

Skripta *Slot.cs* sadrži klasu *Slot*. Pored već spomenutih atributa, ova klasa sadrži i atribut *loot* u na koji se referencira plijen koji je na toj poziciji kada je otvoreno sučelje te kutije. Klasa *Slot* u kodu je definirana na sljedeći način:

```
using UnityEngine;
using UnityEngine.UI;

public class Slot : MonoBehaviour
{
    public Image background;
    public Image icon;
    public Text label;

    public Loot loot;
```

Klasa *Slot* sadrži još tri metode. Prva metoda je *PrintLootDescription*. Metoda vraća opis svojstava plijena koji je na toj poziciji. Druga metoda je *TakeLoot* služi za pokupljanje plijena na toj poziciji u sučelju kutije, ako plijen postoji. To radi u nekoliko koraka ovim redoslijedom:

1. Opremanjem igrača plijenom
2. Pribrajanjem svih svojstava plijena u opremi igraču
3. Ako je pokupljeni plijen oružje, aktiviranjem objekta oružja istog tipa, a ako nije, samo se poziva metoda za konfiguriranje već aktivnog oružja na nova svojstva
4. Ažuriranjem sučelja opreme, kutije i igračevih svojstava
5. Ažuriranjem sučelja za prikaz preostalih metaka ako postoji oružje u opremi i ažuriranjem sučelja za prikaz trenutnih životnih bodova

Treća metoda je *ReturnLoot* služi za vraćanje plijena na toj poziciji u sučelju opreme, ako plijen postoji. To radi u nekoliko koraka ovim redoslijedom:

1. Skidanjem opreme igrača
2. Pribrajanjem svih svojstava plijena u opremi igraču
3. Ako je vraćeni plijen oružje, deaktiviranjem objekata oružja, a ako nije, samo se poziva metoda za konfiguriranje već aktivnog oružja na nova svojstva
4. Ažuriranjem sučelja opreme, kutije i igračevih svojstava
5. Ažuriranjem sučelja za prikaz preostalih metaka ako postoji oružje u opremi i ažuriranjem sučelja za prikaz trenutnih životnih bodova

Kod za ove tri navedene metode je sljedeći:

```
public string PrintLootDescription()
{
    return loot.PrintLootDescription();
}

public void TakeLoot() // Take Loot Button
{
    if (loot != null)
    {
        PlayerEquipment.instance.EquipLoot(loot);
        PlayerStats.instance.CalculateStats();
        if (loot.equipmentType == EquipmentType.Weapon)
            PlayerGunSwap.instance.ActivateWeapon(loot.lootType);
        else
            PlayerGunSwap.instance.activeGun?.ConfigureWeapon();
        UI.instance.UpdateEquipmentUI();
        UI.instance.UpdateBoxUI();
        UI.instance.UpdateStatsUI();
        if (PlayerGunSwap.instance.activeGun != null)
            UI.instance.UpdateAmmoUI();
        UI.instance.UpdateHealthUI();
    }
}

public void ReturnLoot() // Return Loot Button
{
    if (loot != null)
    {
        PlayerEquipment.instance.UnequipLoot(loot);
        PlayerStats.instance.CalculateStats();
        if (loot.equipmentType == EquipmentType.Weapon)
            PlayerGunSwap.instance.DeactivateWeapons();
        else
            PlayerGunSwap.instance.activeGun?.ConfigureWeapon();
        UI.instance.UpdateEquipmentUI();
        UI.instance.UpdateBoxUI();
        UI.instance.UpdateStatsUI();
        if (PlayerGunSwap.instance.activeGun != null)
            UI.instance.UpdateAmmoUI();
        UI.instance.UpdateHealthUI();
    }
}
```

9.2. Skripta *UI.cs*

Skripta *UI.cs* sadrži *singleton* klasu UI u kojoj se nalaze sve metode za prikaz i ažuriranje sučelja. U ovoj klasi prvo su definirani javni atributi na koje su referencirani svi elementi koji se tijekom igre mijenjaju. Definiran je i javni atribut *boxInUI* u koji se dodaje referenca na kutiju s plijenom koju otvaramo. Kod ovog dijela skripte je sljedeći:

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

public class UI : MonoBehaviour
{
    #region Singleton (…)

    public Camera mainCamera;
    public Text playerHealthUI;
    public Text playerAmmoUI;
    public GameObject enemyHealthUI;
    public Text enemyHealthText;
    public GameObject interactUI;
    public GameObject boxUI;
    public GameObject equipmentUI;
    public GameObject lootDescriptionUI;
    public Text lootDescriptionText;
    public Text statsText;
    public Text roundText;
    public GameObject floatingDamageUI;

    public Box boxInUI;
```

Zatim su definirane: lista pozicija u opremi, lista pozicija kutiji i lista gumbova za vraćanje plijena u kutiju. U metodi *Start* ove tri liste se pune referencama na elemente sučelja po njihovim odgovarajućim tagovima koji su im prethodno bili dodijeljeni. Prije nego što igra počne sakrivaju se ovi elementi sučelja: protivnički životni bodovi, znak za interakciju i sučelje opreme i kutije. Ovaj dio koda je sljedeći:

```
    private List<GameObject> boxSlots;
    private List<GameObject> equipmentSlots;
    private List<Button> returnLootButtons;
```

```
    void Start()
    {
        boxSlots = GameObject.FindGameObjectsWithTag("boxSlot").ToList();
        boxSlots = boxSlots.OrderBy(x => x.transform.name).ToList();
        equipmentSlots =
        GameObject.FindGameObjectsWithTag("equipmentSlot").ToList();
        equipmentSlots = equipmentSlots.OrderBy(x =>
        x.transform.name).ToList();

        List<GameObject> buttons =
        GameObject.FindGameObjectsWithTag("returnLootButton").ToList();
        buttons = buttons.OrderBy(x => x.transform.name).ToList();
        returnLootButtons = new List<Button>();
        buttons.ForEach(x =>
        returnLootButtons.Add(x.GetComponent<Button>()));

        enemyHealthUI.SetActive(false);
        interactUI.SetActive(false);
        HideEquipmentAndBox();
    }
```

Sljedeća metoda u klasi UI je metoda *Update* u kojoj se svaku sličicu igre izvršavaju metode *ShowEnemyHealthUI* i *ShowInteractUI*. Metoda *ShowEnemyHealthUI* prikazuje protivničke životne bodove ako igrač nišani protivnika. Tada se i veličina panela sa životnim bodovima prilagođava trenutnim protivničkim životnim bodovima na način da se omjer trenutnih i maksimalnih životnih bodova protivnika pomnoži sa maksimalnom širinom panela. Uz to u tekstualni objekt se upisuju trenutni životni bodovi protivnika. Ako igrač ne nišani protivnika, panel sa protivničkim životnim bodovima se sakrije. Metoda *ShowInteractUI* pokazuje obavijest za moguću interakciju ako igrač nišani objekt s kojim može stupiti u interakciju. U igri su to kutije s plijenom ili gumb za novu rundu. Ako igrač ne nišani ništa od navedenog, obavijest za moguću interakciju se sakriva. Kod ovih triju metoda je sljedeći:

```
void Update()
{
    ShowEnemyHealthUI();
    ShowInteractUI();
}

private void ShowEnemyHealthUI()
{
    Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit))
    {
        Enemy enemy;
        if (hit.transform.TryGetComponent<Enemy>(out enemy))
        {
```

```
            Vector2 healthSize = new Vector2((float)enemy.currentHealth /
            enemy.maxHealth * 800, 30);
            enemyHealthUI.GetComponent<RectTransform>().sizeDelta =
            healthSize;
            enemyHealthText.text = enemy.currentHealth.ToString();
            enemyHealthUI.SetActive(true);
        }
        else
        {
            enemyHealthUI.SetActive(false);
        }
    }
}

private void ShowInteractUI()
{
    Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit, 5))
    {
        Box box;
        if (hit.transform.TryGetComponent<Box>(out box))
        {
            interactUI.SetActive(true);
        }
        RoundButton roundButton;
        if (hit.transform.TryGetComponent<RoundButton>(out roundButton))
        {
            interactUI.SetActive(true);
        }
    }
    else
    {
        interactUI.SetActive(false);
    }
}
```

Zatim slijede metoda za otvaranje sučelja kutije(*ShowBox*) i metoda za otvaranje opreme(*ShowEquipment*). Metoda *ShowBox* prima parametar tipa *Box*. U tom parametru nalazi se popis plijena koji se nalazi u kutiji koja je otvorena. Ova metoda postavlja odabranu kutiju u sučelje, poziva metodu za ažuriranje sučelja kutije, prikazuje sučelje kutije i poziva metodu *ShowEquipment*. Metoda *ShowEquipment* prikazuje sučelje opreme i otključava kursor. Ako sučelje kutije nije otvoreno onda se metodom *DisableButtons* onemogućuju svi gumbi na pozicijama u opremi, a ako je otvoreno onda se omogućavaju metodom *EnableButons*. Metoda *HideEquipmentAndBox* skriva sučelje opreme i kutije te zaključava kursor. Kod ovih pet metoda je sljedeći:

```
public void ShowBox(Box box)
{
    boxInUI = box;
    UpdateBoxUI();
    boxUI.SetActive(true);
    ShowEquipment();
}

public void ShowEquipment()
{
    equipmentUI.SetActive(true);
    if (boxUI.activeSelf == true)
    {
        EnableButtons();
    }
    else
    {
        DisableButtons();
    }
    PlayerControls.instance.UnlockMouse();
}

private void EnableButtons()
{
    returnLootButtons.ForEach(x => x.interactable = true);
}

private void DisableButtons()
{
    returnLootButtons.ForEach(x => x.interactable = false);
}

public void HideEquipmentAndBox()
{
    boxUI.SetActive(false);
    equipmentUI.SetActive(false);
    lootDescriptionUI.SetActive(false);
    PlayerControls.instance.LockMouse();
}
```

Sljedeće tri metode služe za ažuriranje sučelja kutije, a to su *RemoveBoxUILoot* koja na svim pozicijama u sučelju kutije uklanja plijen koji je na poziciji i *UpdateBoxInUI* koja prvo poziva metodu *RemoveBoxUILoot*, zatim na pozicije u sučelju kutije stavlja nove instance plijena na pozicije i za svaku poziciju poziva metodu *UpdateSlotUI*. Metoda *UpdateSlotUI* za parametar prima instancu plijena. Ako nema instance plijena, izgled pozicije vraća na početno

stanje, a ako ima instance plijena poziciju mijenja prema svojstvima plijena na toj poziciji. Kod za ove tri metode je sljedeći:

```
public void UpdateBoxUI()
{
    RemoveBoxUILoot();

    List<Loot> loot = boxInUI.loot;
    List<GameObject> slots = boxSlots;

    for (int i = 0; i < loot.Count; i++)
    {
        slots[i].GetComponent<Slot>().loot = loot[i];
    }

    foreach (GameObject boxSlot in boxSlots)
    {
        Slot slot = boxSlot.GetComponent<Slot>();
        UpdateSlotUI(slot);
    }
}

private void RemoveBoxUILoot()
{
    boxSlots.ForEach(x => x.GetComponent<Slot>().loot = null);
}

private void UpdateSlotUI(Slot slot)
{
    if (slot.loot == null)
    {
        slot.label.text = "";
        slot.icon.sprite = null;
        slot.background.color = Color.clear;
    }
    else
    {
        slot.icon.sprite = slot.loot.icon;
        slot.label.text = slot.loot.label;

        if (slot.loot.quality == 1) slot.background.color = Color.grey;
        if (slot.loot.quality == 2) slot.background.color = Color.green;
        if (slot.loot.quality == 3) slot.background.color = Color.blue;
        if (slot.loot.quality == 4) slot.background.color = Color.magenta;
        if (slot.loot.quality == 5) slot.background.color = Color.yellow;
    }
}
```

Metoda *UpdateStatsUI* ažurira opis svih igračevih svojstava, a kod za ovu metodu je sljedeći:

```
public void UpdateStatsUI()
{
    statsText.text = "";
    foreach (Stat stat in PlayerStats.instance.stats)
    {
            statsText.text += stat.PrintStat() + "\n";
    }
 }
```

Metoda *ShowLootDescription* ispisuje kompletni opis plijena na poziciji na koji pokazuje kursor miša i prikazuje ga na lokaciji od 150 jedinica ispod kursora miša ako plijen na toj poziciji postoji. Metoda *HideLootDescription* sakriva opis plijena kada se kursor miša domakne od pozicije na kojoj je plijen. Kod za ove dvije metode je sljedeći:

```
public void ShowLootDescription(Slot slot) // OnPointerEnter Event
{
    if (slot.loot != null)
    {
        Vector3 lootDescriptionPosition = Input.mousePosition;
        lootDescriptionPosition.y -= 150;
        lootDescriptionUI.transform.position = lootDescriptionPosition;
        lootDescriptionText.text = slot.loot.label.ToUpper() + "\n";
        lootDescriptionText.text += slot.PrintLootDescription();
        lootDescriptionUI.SetActive(true);
    }
}
public void HideLootDescription() // OnPointerExit Event
{
     lootDescriptionUI.SetActive(false);
}
```

Sljedeća metoda u klasi UI je *UpdateRoundUI*. Ova metoda osvježava prikaz runde i pokreće kratku animaciju naziva „*NewRound*“ vlastite izrade. Animacija nakratko podiže tekstualni objekt runde da obavijesti igrača da je pobijedio sve protivnike pa vraća tekstualni objekt na staru poziciju. Metoda *UpdateHealthUI* ažurira trenutne životne bodove igrača, a metoda *UpdateAmmoUi* ažurira koliko je igraču preostalo metaka prije nego što mora ponovo puniti oružje. Kod ovih triju metoda je sljedeći:

```
public void UpdateRoundUI()
{
    roundText.text = Round.instance.currentRound.ToString();
```

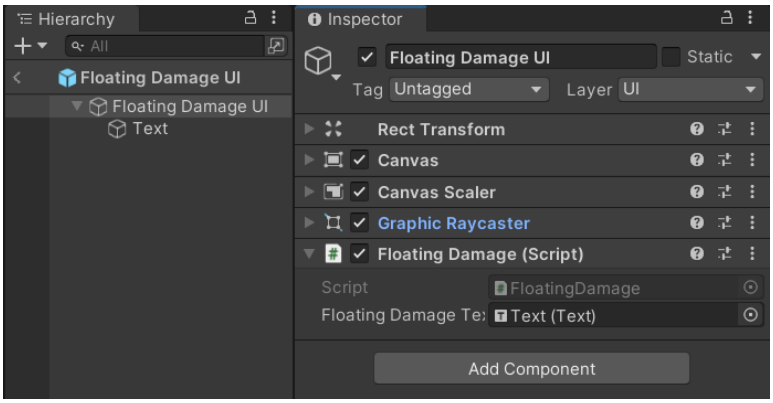
```
    roundText.GetComponentInParent<Animator>().Play("NewRound");
}

public void UpdateHealthUI()
{
    playerHealthUI.text =
    Mathf.Round(PlayerStats.instance.currentHealth).ToString();
}

public void UpdateAmmoUI()
{
    playerAmmoUI.text =
    PlayerGunSwap.instance.activeGun.currentAmmo.ToString();
}
```

Zadnja metoda u klasi UI je metoda *ShowFloatingDamage*. Ova metoda prima tri parametra: šteta koju je zadobio protivnik, je li igrač napravio kritični pogodak i referencu protivnika kojemu je načinjena šteta. Metoda instancira *prefab* „*floatingDamageUI*“ na protivnikovoj lokaciji metodom *Instantiate* i poziva metodu *ShowDamage* koja se nalazi u skripti *FloatingDamage.cs*. Kod za metodu *ShowFloatingDamage* je sljedeći:

```
public void ShowFloatingDamage(float damageRecieved, bool isCritical,
Enemy enemy)
{
    GameObject floatingDamage = Instantiate(floatingDamageUI,
    enemy.transform.position, Quaternion.identity);
    floatingDamage.GetComponent<FloatingDamage>().ShowDamage(damageRecieved
    , isCritical);
}
```

9.3. Skripta *FloatingDamage.cs*

Skripta *FloatingDamage.cs* ima klasu *FloatingDamage*. Skripta je dodana kao komponenta *prefabu* naziva *„Floating Damage UI“* koji se sastoji od *canvasa* i tekstualnog objekta kao što možemo vidjeti na slici 20:



Slika 20 Floating Damage UI (Izvor: vlastita slika zaslona, 2022.)

Na komponentu skripte referenciran je tekstualni objekt naziva „*Text*“. Kod Skripte *FloatingDamage.cs* je sljedeći:

```
using System;
using UnityEngine;
using UnityEngine.UI;

public class FloatingDamage : MonoBehaviour
{
    public Text floatingDamageText;

    void Start()
    {
        Destroy(gameObject, 0.5f);
    }
    void Update()
    {
        floatingDamageText.transform.position += new Vector3(0f, 1f, 0f);
    }

    public void ShowDamage(float damageRecieved, bool isCritical)
    {
        floatingDamageText.text = Math.Round(damageRecieved).ToString();
        if (isCritical)
        {
            floatingDamageText.color = Color.red;
            floatingDamageText.fontStyle = FontStyle.Bold;
        }
        else
        {
            floatingDamageText.color = Color.black;
            floatingDamageText.fontStyle = FontStyle.Normal;
        }
    }
}
```

U metodi *Start* instancirani objekt se uništava metodom *Destroy*, ali sa odgodom od pola sekunde. Dok se instancirani objekt ne uništi, objekt se pomiče po y-osi svaku sličicu po sekundi, što je određeno metodom *Update*. Metoda *ShowDamage* prima podatke o šteti protivnikovih životnih bodova i o tome je li igrač postigao kritični pogodak. U metodi se prvo vrijednost štete zaokružuje i zapisuje u tekstualni objekt. Ako je pogodak kritičan boja teksta je mijenja se u crvenu i tekst se podeblja, a ako nema kritičnog pogotka boja teksta mijenja se u crnu i tekst je normalan.

10. Borba

U ovom poglavlju obradit će svi procesi koji se događaju u igri kad je u tijeku borba. To uključuje procese pucanja, punjenja metaka, izračunavanja i nanošenja štete protivnicima, kao i pucanje i kretanje protivnika, nanošenje štete igraču. Obradit će se i način na koji se odvijaju runde u igri.

10.1. Skripta *Round.cs*

Skripta *Round.cs* pridružena je kao komponenta praznom objektu „Game“. Na skriptu je na javni atribut *enemyPrefab* referenciran *prefab Enemy*. Skripta *Round.cs* sadrži *singleton* klasu koja je definirana na sljedeći način:

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

public class Round : MonoBehaviour
{
    #region Singleton (…)

    public GameObject enemyPrefab;
    public int currentRound = 1;
    private int enemiesLeft;
    private List<GameObject> boxes;
    private List<GameObject> spawnPoints;
```

U metodi *Start* u listu kutija se prema prethodno postavljenim tagovima spremaju reference na kutije s plijenom, a u listu pozicija gdje se pojavljuju protivnici, isto se prema tagovima spremaju objekti pozicija. Kod metode *Start* je sljedeći:

```
void Start()
{
    boxes = GameObject.FindGameObjectsWithTag("box").ToList();
    spawnPoints =
    GameObject.FindGameObjectsWithTag("spawnPoint").ToList();
}
```

Metoda *StartRound* započinje borbu ukoliko je broj protivnika jednak nuli jer ako još ima protivnika to znači da prošla runda nije završila. U metodi se zatim na svakoj od pozicija instancira jedan protivnik i poziva se metoda za deaktivaciju kutija. Metoda *StartRound* definirana je sljedećim kodom:

```
public void StartRound()
{
    if (enemiesLeft == 0)
    {
        enemiesLeft = spawnPoints.Count;
        spawnPoints.ForEach(x => Instantiate(enemyPrefab,
        x.transform.position, Quaternion.identity));
        DeactivateBoxes();
    }
}
```

Sljedeća metoda je *MarkEnemyDeath*. Svaki put kad igrač ubije protivnika broj preostalih protivnika se smanjuje za jedan. Ako broj protivnika dođe do nule, ova metoda poziva funkcije *ChangeToNextRound*, *ResetBoxes* i *ActivateBoxes*. Kod ove metode je sljedeći:

```
public void MarkEnemyDeath()
{
    enemiesLeft--;
    if (enemiesLeft == 0)
    {
        ChangeToNextRound();
        ResetBoxes();
        ActivateBoxes();
    }
}
```

Metoda *ChangeToNextRound* povećava broj runde za jedan, resetira trenutne životne bodove igrača i poziva metodu za ažuriranje sučelja runde i metodu za ažuriranje sučelja za prikaz životnih bodova igrača. Kod ove metode je sljedeći:

```
private void ChangeToNextRound()
{
    currentRound++;
    PlayerStats.instance.ResetCurrentHealth();
    UI.instance.UpdateHealthUI();
    UI.instance.UpdateRoundUI();
}
```

Metoda *ResetBoxes* za svaku kutiju s plijenom mijenja atribut *isGenerated* iz klase *Box* u *false* da bi se prilikom sljedećeg otvaranja kutije generirao novi plijen u kutijama. Kod ove metode je sljedeći:

```
public void ResetBoxes()
{
    boxes.ForEach(x => x.GetComponent<Box>().isGenerated = false);
}
```

Zadnje metode u klasi *Round* su *ActivateBoxes* i *DeactivateBoxes*. Metoda *ActivateBoxes* aktivira sve kutije, tj. kutije se pojavljuju u sceni, a metoda *DeactivateBoxes* deaktivira sve kutije u sceni, tj. kutije se ne pojavljuju u sceni. Kutije se deaktiviraju zato što igrač ne bi mogao mijenjati svoju opremu tijekom borbe. Kod ovih metoda je sljedeći:

```
private void ActivateBoxes()
{
    boxes.ForEach(x => x.SetActive(true));
}

private void DeactivateBoxes()
{
    boxes.ForEach(x => x.SetActive(false));
}
```

10.2. Skripta *RoundButton.cs*

Skripta *RoundButton.cs* komponenta je objekta u igri koji predstavlja gumb za početak borbe. Njena klasa, *RoundButton*, ima samo jednu metodu, a to je metoda koja poziva drugu metodu za početak runde iz skripte *Round.cs.* Kod klase *RoundButton* je sljedeći:

```
using UnityEngine;

public class RoundButton : MonoBehaviour
{
    public void StartRound()
    {
        Round.instance.StartRound();
    }
}
```

10.3. Skripta *Gun.cs*

Skripta Gun.cs sadrži klasu *Gun* i komponenta je svakog od četiri objekta oružja. Sadrži metode za konfiguraciju oružja, pucanje i punjenje oružja, i za preračunavanje štete načinjene protivniku. Klasa *Gun* definirana je na sljedeći način:

```
using UnityEngine;

public class Gun : MonoBehaviour
{
    public Camera mainCamera;
    public AudioSource soundShoot, soundReload, soundHit;
    public ParticleSystem particleShoot, particleHit;
    public Animator animatorReload;

    private float attackDamage, critDamage, critChance, timeBetweenShots,
    range;
    public int maxAmmo = 0, currentAmmo = 0;
    private bool isReadyToShoot = true, isReloading = false, isCritical =
    false;
```

Prva metoda klase *Gun* je *ConfigureWeapon*. U toj metodi se sva igračeva svojstva koja utječu na učinak oružja primjenjuju na oružje. Kod za ovu metodu je sljedeći:

```
public void ConfigureWeapon()
{
    attackDamage =
    PlayerStats.instance.stats[(int)StatType.AttackDamage].value;
    critDamage = attackDamage *
    PlayerStats.instance.stats[(int)StatType.CritDamage].value / 100f;
    critChance =
    PlayerStats.instance.stats[(int)StatType.CritChance].value;
    timeBetweenShots = 1f /
    PlayerStats.instance.stats[(int)StatType.FireRate].value;
    animatorReload.speed =
    PlayerStats.instance.stats[(int)StatType.ReloadSpeed].value;
    maxAmmo =
   (int)PlayerStats.instance.stats[(int)StatType.ClipSize].value;
    currentAmmo = maxAmmo;
    range = PlayerStats.instance.stats[(int)StatType.Range].value;
    }
```

Jačini napada oružja(*attackDamage*) se samo pridružuje vrijednost igračeve jačine napada. Jačina kritičnog pogotka oružja(*critDamage*) dobije se množenjem jačine napada oružja i igračeve jačine kritičnog pogotka pa dijeljenjem sa 100. Šansa za kritični napad oružja(*critChance*) se samo pridružuje vrijednost igračeve šanse za kritični napad. Vrijeme između dva pucnja(*timeBetweenShots*) je recipročna vrijednost igračeve brzine pucanja.

Da bi se prilagodila brzina punjenja oružja treba se promijeniti atribut *speed* u animatoru oružja. Taj atribut predstavlja brzinu reprodukcije animacije, a početno je postavljena na vrijednost 1. To znači da atributu *speed* treba pridružiti vrijednost igračeve brzine punjenja oružja. Maksimalni broj metaka (*maxAmmo*) oružja je cijeli broj pa se mora igračev broj metaka promijeniti u tip *int*. Maksimalni i trenutni broj metaka je na početku svake runde jednak pa se vrijednosti ta dva atributa trebaju izjednačiti. Dometu oružja pridružuje se vrijednost igračevog dometa.

Sljedeća metoda klase *Gun* je *Shoot*. Ako je oružje spremno za ponovno pucanje i ako igrač trenutno ne puni oružje, varijabla *isReadyToShoot* postavlja se na *false* da oružje ne bi moglo pucati brže neko to dozvoljava brzina pucanja oružja, smanjuje se trenutni broj metaka za 1, ažurira se sučelje za prikaz broja metaka i reproduciraju se zvuk pucnja i čestični efekt blijeska prilikom pucnja. Ako igrač nišani na protivnika i protivnik je u dometu oružja, na mjestu pogotka reproducira se čestični efekt pogotka i zvuk pogotka. Nakon toga se poziva metoda *CalculateDamage* koja računa štetu protivniku. Nakon toga poziva se metoda *TakeDamage* iz klase *Enemy* i šteta se prikazuje pozivanjem metode *ShowFloatingDamage* iz klase UI. Zatim se provjerava ima li oružje još metaka. Ako nema, atribut *isReloading* se postavlja na *false*, i poziva se metoda *Reload* kojom se puni oružje. Na kraju metode *Shoot*, metodom *Invoke* priziva se Metoda *ResetShot*. Metoda *Invoke* prima dva parametra. Prvi parametar je ime metode koja će se izvršiti, a drugi parametar je vrijeme za koje će se navedena metoda početi izvršavati.[29] Metoda *ResetShot* izvršava se nakon što prođe vrijeme između dva pucnja. Metoda *ResetShot* vraća vrijednost varijable *isReadyToShot* na *true*. Kod za metode *Shoot* i *ResetShot* je sljedeći:

```
public void Shoot()
{
    if (isReadyToShoot && !isReloading)
    {
        isReadyToShoot = false;
        currentAmmo--;
        UI.instance.UpdateAmmoUI();
        particleShoot.Play();
        soundShoot.Play();

        Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, range))
        {
            Enemy enemy;
            if (hit.transform.TryGetComponent<Enemy>(out enemy))
            {
                particleHit.transform.position = hit.point;
                particleHit.Play();
```

```
                soundHit.transform.position = hit.point;
                soundHit.Play();
                float calculatedDamage = CalculateDamage();
                enemy.TakeDamage((int)calculatedDamage);
                UI.instance.ShowFloatingDamage(calculatedDamage,
                isCritical, enemy);
            }
        }
        if (currentAmmo == 0)
        {
            isReloading = true;
            Reload();
        }
        Invoke("ResetShot", timeBetweenShots);
    }
}

private void ResetShot()
{
   isReadyToShoot = true;
}
```

Metoda *CalculateDamage* generira nasumični broj od 0 do 100. Ako je šansa za kritični pogodak veća ili jednaka od nasumičnog broja, metoda vraća vrijednost jačine kritičnog pogotka, ako je šansa manja onda metoda vraća jačinu običnog napada oružja. Kod ove metode je sljedeći:

```
private float CalculateDamage()
{
    if (critChance >= Random.Range(0, 101))
    {
        isCritical = true;
        return critDamage;
    }
    else
    {
        isCritical = false;
        return attackDamage;
    }
}
```

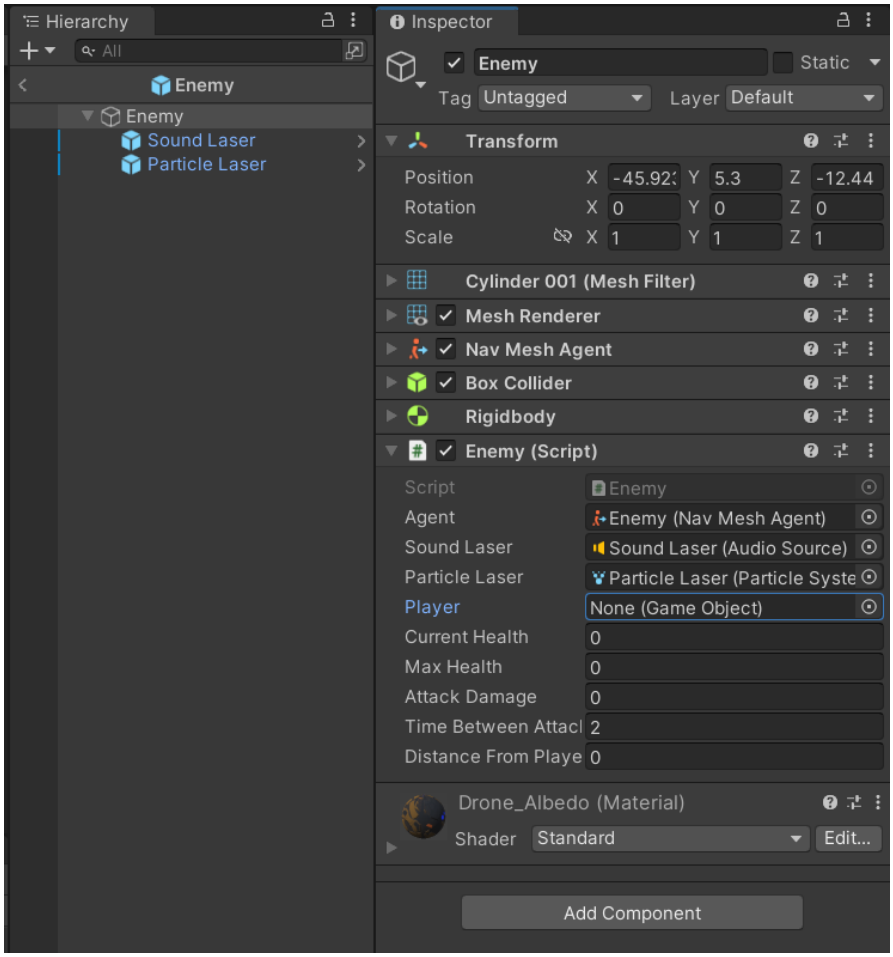
Metoda *Reload* postavlja vrijednost atributa *isReloading* na *true* zato da oružje ne bi moglo pucati kad se oružje puni. Zatim se reproduciraju zvuk i animacija punjenja oružja. Na kraju metode se metodom *Invoke* priziva metoda *ResetAmmo* s odgodom izvršavanja u vremenu trajanja animacije punjenja. Početno vrijeme trajanja animacije punjenja svakog tipa oružja je 3 sekunde pa se to vrijeme dijeljenjem sa brzinom punjenja oružja prilagođava na stvarno trajanje punjenja oružja. Metoda *ResetAmmo* postavlja trenutni broj metaka na

maksimalni broj metaka, poziva metodu za ažuriranje sučelja za prikaz broja metaka i postavlja atribut *isReloading* na *false*. Kod ovih dviju metoda je sljedeći:

```
public void Reload()
{
    isReloading = true;
    soundReload.Play();
    animatorReload.Play("Reload");
    Invoke("ResetAmmo", 3f / animatorReload.speed);
}

private void ResetAmmo()
{
    currentAmmo = maxAmmo;
    UI.instance.UpdateAmmoUI();
    isReloading = false;
}
```

10.4. Skripta *Enemy.cs*

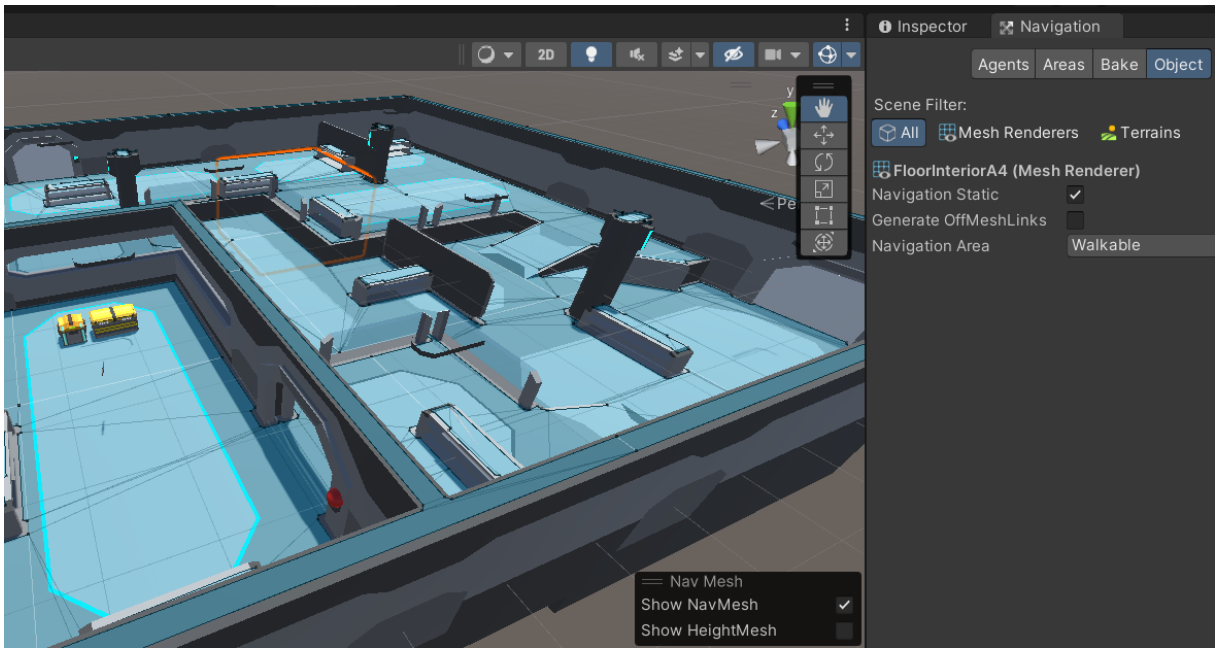
Skripta *Enemy.cs* sadrži klasu *Enemy* i komponenta je *prefaba Enemy*, tj. objekta protivnika. Objekt protivnika sastoji se od modela drona, za koji je korištena imovina *„Low poly combat drone*“ uvezena iz *Asset Storea*. Objekti djeca su mu objekti zvuka lasera i čestični efekt lasera koji su vlastite izrade. Hijerarhijski prikaz objekata u objektu protivnika i komponente objekta protivnika možemo vidjeti na sljedećoj slici:



Slika 21 Objekt protivnika (Izvor: vlastita slika zaslona, 2022.)

Objekt protivnika sadrži komponentu *„Nav Mesh Agent*“. Ova komponenta služi za stvaranje likova u igri koji se mogu kretati prema cilju i izbjegavati prepreke na putu kao i jedan drugoga. U toj komponenti mogu se prilagoditi postavke kao što su naprimjer: brzina kretanja lika, brzina okretanja lika, visina i radijus lika i slično.[30] Da bi se likovi s komponentom *„Nav Mesh Agent“* mogli kretati po sceni, mora se odrediti koje površine u sceni su prohodne. To se može odrediti u prozoru *„Navigation“*. U hijerarhiji objekata označimo objekte po kojima bi se likovi trebali moći kretati kao i objekte koji bi trebali blokirati kretnju agenata, pa ih u prozoru *„Navigation“* u predjelu *„Object“* označimo kao *„Navigation Static“*. Uz to, u prozoru

„Navigation“ u predjelu *„Agent“* mogu se prilagođavati i postavke kao što su naprimjer: radijus do koliko se može agent približiti zidu ili drugom agentu, visina prostora kroz koji agent može proći, visina prepreke koji agent može proći, nagib površine po kojoj se agent može popeti. Kad se poslože željene postavke, u predjelu „*Bake*“ trebamo kliknuti na gumb „*Bake*“ da se odrede sve površine po kojima se agent može kretati prema prije određenim postavkama, tj. da se izgradi „*NavMesh*“.[31] Te površine su u sceni označene plavom bojom. Na sljedećoj slici vidimo primjer prohodnih površina u igri koju izrađujemo:



Slika 22 Površine za kretanje agenta (Izvor: vlastita slika zaslona, 2022.)

Klasa *Enemy* u skripti *Enemy.cs* definirana je na sljedeći način:

```
using UnityEngine;
using UnityEngine.AI;

public class Enemy : MonoBehaviour
{
    public NavMeshAgent agent;
    public AudioSource soundLaser;
    public ParticleSystem particleLaser;

    public GameObject player;
    public float currentHealth, maxHealth, attackDamage, timeBetweenAttacks
    = 2f, distanceFromPlayer;
    private bool isReadyToShoot = true, inRange = false;
```

U metodi *Start*, koja se izvodi kad se objekt protivnika instancira, u varijablu se dodaje referenca na objekt igrača. U varijablu *maxHealth* zapisuju se maksimalni životni bodovi

protivnika koji iznose 1000 i na tu vrijednost se još dodaje 200 životnih bodova po svakoj rundi. U varijablu *attackDamage* zapisuje se jačina napada protivnika koja iznosi 10 i na tu vrijednost se još dodaje jačina napada 2 po svakoj rundi. Vrijednost varijable *maxHealth* pridružuje se varijabli *currentHealth*, što predstavlja trenutnu jačinu napada. Kod za metodu *Start* je sljedeći:

```
void Start()
{
    player = GameObject.Find("Player");
    maxHealth = 1000f + (Round.instance.currentRound * 200f);
    attackDamage = 10f + (Round.instance.currentRound * 2f);
    currentHealth = maxHealth;
}
```

Metoda *Update* poziva funkcije *MoveToPlayer* i *ShootPlayer*. Metoda *MoveToPlayer* izvršava se ako igrač nije u dometu protivnika, na način da se varijabli *distanceFromPlayer* pridružuje udaljenost protivnika od igrača, a odredište agenta, tj. protivnika, postavlja se na poziciju igrača. Kod za ove dvije metode je sljedeći:

```
void Update()
{
    MoveToPlayer();
    ShootPlayer();
}

public void MoveToPlayer()
{
    if (!inRange)
    {
        distanceFromPlayer = Vector3.Distance(player.transform.position,
        transform.position);
        agent.destination = player.transform.position;
    }
}
```

Sljedeća metoda klase *Enemy* je *ShootPlayer*. Postupak izvršavanja ove metode je sljedeći: ako je protivnik u dometu igrača i spreman je pucati, varijable *inRange* postavlja se na *true*, a *isReadyToShoot* postavlja se na *false* da protivnik ne bi odmah pucao više puta. Nakon toga se protivnik okreće prema igraču i reproduciraju se zvuk i efekt pucnja. Ako je igračeva šansa za blokiranje manja od nasumično generiranog broja između 1 i 100, igraču se smanjuju životni bodovi za napad protivnika i ažurira se sučelje koje pokazuje životne bodove igrača. Ako su igračevi životni bodovi manji ili jednaki nuli, igrač umire i igra je gotova pa se učitava posljednja scena u igri. Na kraju se priziva metoda *ResetAttack* s odgodom od 2 sekunde. Metoda *ResetAttack* postavlja atribut *isReadyToShoot* na *true* da bi protivnik opet

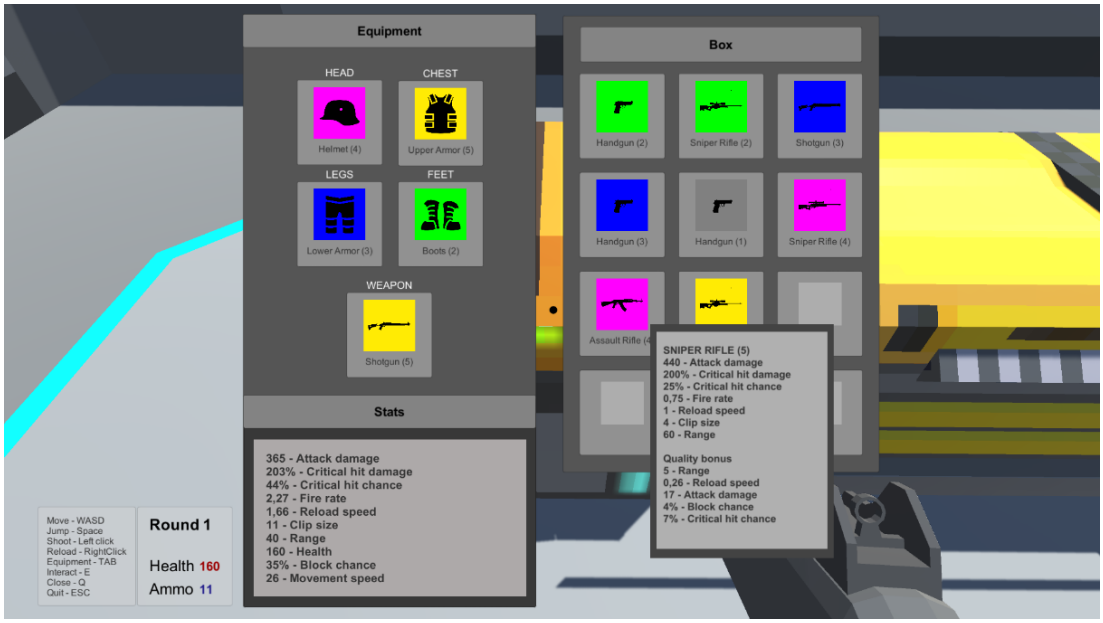
mogao napasti. Ako igrač nije u dometu, varijabla *inRange* postavlja se na *false* da bi se protivnik mogao kretati do igrača. Kod za metode *ShootPlayer* i *ResetAttack* su sljedeće:

```
public void ShootPlayer()
{
    if (distanceFromPlayer <= 20f)
    {
        if (isReadyToShoot)
        {
            inRange = true;
            isReadyToShoot = false;
            transform.LookAt(player.transform);
            soundLaser.Play();
            particleLaser.Play();

            if (PlayerStats.instance.stats[(int)StatType.BlockChance].value
            <= Random.Range(0, 101))
            {
                PlayerStats.instance.currentHealth -= attackDamage;
                UI.instance.UpdateHealthUI();
                if (PlayerStats.instance.currentHealth <= 0f)
                {
                    Scene.instance.LoadEndScene();
                }
            }
            Invoke("ResetAttack", timeBetweenAttacks);
        }
    }
    else
    {
        inRange = false;
    }
}

private void ResetAttack()
{
    isReadyToShoot = true;
}
```

Posljednja metoda klase *Enemy* je metoda *TakeDamage*. Ova metoda prima parametar *damageRecieved*, čija vrijednost pokazuje koliko protivnik gubi životnih bodova. Ta vrijednost se oduzima od trenutnih životnih bodova protivnika. Ako životni bodovi protivnika padnu ispod nule, protivnik umire i objekt protivnika briše se metodom *Destroy*. Nakon toga se još poziva naredba *MarkEnemyDeath* iz klase *Round*.

11. Rezultat izrade igre

Konačna igra izrađena je na način da su prilikom igranja igre vrlo istaknute dvije faze u jednoj rundi igre. Prva faza je opremanje igrača s plijenom gdje igrač kombinira različita svojstva plijena da ukupna kombinacija svojstava plijena što je moguće učinkovitija u borbi. Ova faza prikazana je na sljedećoj slici:



Slika 23 Pokupljanje plijena (Izvor: slika vlastitog zaslona, 2022.)

Druga faza igre je borba protiv dronova gdje igrač mora pobijediti dronove prije nego oni pobjede igrača. U svakoj rundi sve je teže pobijediti dronove jer imaju više životnih bodova i rade više štete igraču. Jačina plijena kojim se igrač oprema je isto sve jača, ali ne u toj mjeri koliko jačina dronova. Faza borbe je prikazana na sljedećoj slici:

Slika 24 Borba s dronovima (Izvor: slika vlastitog zaslona, 2022.)

12. Zaključak

Cilj izrade igre s prikupljanjem plijena je bio osmisliti plijen i način na koji se generira plijen pa oko te mehanike napraviti smislenu igru. Budući da sustav plijena dobro funkcionira u igrama gdje postoji progresija kroz igru, tj. igre u kojima se plijen pojačava progresijom u igri, najjednostavnije je bilo napraviti igru koja ima jednu mapu, igra se odvija u rundama, a plijen i protivnici su svaku rundu sve jači. Taj dio igre dovoljno dobro funkcionira prvih deset do dvadeset rundi igre pa je taj dio igre djelomično uspio, a mogao bi se poboljšati dodatnim balansiranjima i testiranjima igre.

Sam plijen je dovoljno raznolik da se razlike primijete u igri kada se koriste i dvije različite instance istog plijena, a puno bolje se vidi utjecaj svojstava plijena kad se različiti tipovi plijena dobro iskombiniraju. Nešto što bi još poboljšalo kako svojstva plijena utječu na igru bilo bi dodavanje svojstava specifičnih za određeni tip plijena što bi se po mom mišljenju moglo dodati u igru bez puno mijenjanja onoga što je već izrađeno.

Kad sam počeo izrađivati ovu igru imao sam vrlo drukčiju ideju kako će igra na kraju izgledati ali, u konačnici, zadovoljan sam s izrađenom igrom, jer mi je dobar osjećaj zaigrati ovu igru, a to je glavni cilj svake igre, da zabavi igrača.

Popis literature

[1] „Loot (video games)“, *Wikipedia*. 28. kolovoz 2021. Pristupljeno: 27. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Loot_(video_games)&oldid=1041074489

[2] „Unity (game engine)“, *Wikipedia*. 23. lipanj 2022. Pristupljeno: 29. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1094621448

[3] U. Technologies, „Unity - Manual: Installing Unity“. https://docs.unity3d.com/Manual/GettingStartedInstallingUnity.html (pristupljeno 29. lipanj 2022.).

[4] U. Technologies, „Unity - Manual: Unity’s interface“. https://docs.unity3d.com/Manual/UsingTheEditor.html (pristupljeno 29. lipanj 2022.).

[5] U. Technologies, „Unity - Manual: Unity’s Asset Store“. https://docs.unity3d.com/Manual/AssetStore.html (pristupljeno 30. lipanj 2022.).

[6] „Role-playing video game“, *Wikipedia*. 07. lipanj 2022. Pristupljeno: 27. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Role-playing_video_game&oldid=1092053729

[7] „Shooter game“, *Wikipedia*. 10. lipanj 2022. Pristupljeno: 27. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Shooter_game&oldid=1092437604

[8] *Diablo 2: Resurrected Review*, (01. listopad 2021.). Pristupljeno: 27. lipanj 2022. [Na internetu Video]. Dostupno na: https://www.youtube.com/watch?v=hHm4J_lMW_A

[9] *Borderlands 3 - Official E3 Gameplay Demo*, (11. lipanj 2019.). Pristupljeno: 27. lipanj 2022. [Na internetu Video]. Dostupno na: https://www.youtube.com/watch?v=eFOqG1-T9w8

[10] „*Borderlands* (series)“, *Wikipedia*. 11. lipanj 2022. Pristupljeno: 27. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Borderlands_(series)&oldid=1092558882

[11] U. Technologies, „Unity - Scripting API: MonoBehaviour“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (pristupljeno 27. lipanj 2022.).

[12] U. Technologies, „Unity - Manual: ScriptableObject“. https://docs.unity3d.com/Manual/class-ScriptableObject.html (pristupljeno 27. lipanj 2022.).

[13] U. Technologies, „Unity - Scripting API: Object.Destroy“. https://docs.unity3d.com/ScriptReference/Object.Destroy.html (pristupljeno 28. lipanj 2022.).

[14] U. Technologies, „Unity - Manual: Sprites“. https://docs.unity3d.com/Manual/Sprites.html (pristupljeno 27. lipanj 2022.).

[15] U. Technologies, „Unity - Scripting API: Resources.Load“. https://docs.unity3d.com/ScriptReference/Resources.Load.html (pristupljeno 28. lipanj 2022.).

[16] U. Technologies, „Unity - Scripting API: Random.Range“. https://docs.unity3d.com/ScriptReference/Random.Range.html (pristupljeno 28. lipanj 2022.).

[17] U. Technologies, „Unity - Manual: Scenes“. https://docs.unity3d.com/Manual/CreatingScenes.html (pristupljeno 16. rujan 2022.).

[18] „Canvas | Unity UI | 1.0.0“. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-Canvas.html (pristupljeno 16. rujan 2022.).

[19] U. Technologies, „Unity - Manual: Tags“. https://docs.unity3d.com/Manual/Tags.html (pristupljeno 14. rujan 2022.).

[20] U. Technologies, „Unity - Scripting API: GameObject.FindGameObjectsWithTag“. https://docs.unity3d.com/ScriptReference/GameObject.FindGameObjectsWithTag.html (pristupljeno 14. rujan 2022.).
[21] U. Technologies, „Unity - Scripting API: SceneManagement.SceneManager.LoadScene“. https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html (pristupljeno 16. rujan 2022.).
[22] J. French, „Singletons in Unity (done right)“, *Game Dev Beginner*, 23. prosinac 2021. https://gamedevbeginner.com/singletons-in-unity-the-right-way/ (pristupljeno 28. lipanj 2022.).
[23] U. Technologies, „Unity - Scripting API: MonoBehaviour.Awake()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html (pristupljeno 27. lipanj 2022.).
[24] U. Technologies, „Unity - Manual: Collision“. https://docs.unity3d.com/Manual/collision-section.html (pristupljeno 30. lipanj 2022.).
[25] U. Technologies, „Unity - Manual: Character control“. https://docs.unity3d.com/Manual/character-control-section.html (pristupljeno 30. lipanj 2022.).
[26] U. Technologies, „Unity - Scripting API: GameObject.SetActive“. https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html (pristupljeno 14. rujan 2022.).
[27] U. Technologies, „Unity - Scripting API: Physics.Raycast“. https://docs.unity3d.com/ScriptReference/Physics.Raycast.html (pristupljeno 17. rujan 2022.).
[28] U. Technologies, „Unity - Scripting API: EventSystem“. https://docs.unity3d.com/2018.2/Documentation/ScriptReference/EventSystems.EventSystem.html (pristupljeno 17. rujan 2022.).
[29] U. Technologies, „Unity - Scripting API: MonoBehaviour.Invoke“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Invoke.html (pristupljeno 17. rujan 2022.).
[30] „Unity - Manual: NavMesh Agent“. https://docs.unity3d.com/2017.1/Documentation/Manual/class-NavMeshAgent.html (pristupljeno 17. rujan 2022.).
[31] U. Technologies, „Unity - Manual: Building a NavMesh“. https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html (pristupljeno 17. rujan 2022.).

Popis slika