Implementacija i analiza višeagentnog sustava kroz izradu igre u Unityju pomoću alata ML-Agents

Radotović, Emanuel

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:996329

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Emanuel Radotović

IMPLEMENTACIJA I ANALIZA VISEAGENTNOG SUSTAVA KROZ IZRADU IGRE U UNITYJU POMOCU ALATA ML-AGENTS

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Emanuel Radotović

Matični broj: 0069078505

Studij: Informacijsko i programsko inženjerstvo

IMPLEMENTACIJA I ANALIZA VISEAGENTNOG SUSTAVA KROZ IZRADU IGRE U UNITYJU POMOCU ALATA ML-AGENTS

DIPLOMSKI RAD

Mentor :

Dr. sc. Bogdan Okređa Đurić

Varaždin, veljača 2024.

Emanuel Radotović

Izjava o izvornosti

Izjavljujem da je moj 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Rad obrađuje višeagentne sustave, kao jedno od područja umjetne inteligencije, i njihove osnovne značajke i primjere korištenja, s posebnim naglaskom na korištenje u domeni videoigara. Nastavno, rad proučava mogućnosti i značajke primjene metoda umjetne inteligencije na domenu videoigara, posebno korištenjem programskog alata Unity. Praktični dio rada uključuje implementaciju i analizu odabranog sustava od više agenata pomoću alata ML-Agents u Unityju te analizu specifičnosti koje to razvojno okruženje donosi.

Ključne riječi: unity, višeagentni sustavi, multiagent system

Sadržaj

1. Uvod

Ovaj rad svoj fokus stavlja na implementaciju umjetne inteligencije u računalnu igru. Kako bi to bilo izvedeno, pomoću platforme *Unity* i jezika *C#*, razvijena je 3D igra u kojoj je cilj da igrač pronađe svoj put do mjesta na kojem se nalazi stroj za izlječenje zombija, te ih na taj način spasi. Na putu do ostvarivanja tog cilja sprječavaju ga horde zombija koje ga pokušavaju eliminirati. Kako bi umjetna inteligencija mogla biti implementirana, ideje je bilo potrebno pretočiti u stvarnost putem alata koje nudi *Unity*. Tako su kroz poglavlje *Unity platforma* prikazani svi alati korišteni prilikom izrade igre, kao što su objekti, scena, hijerarhija, animacije, događaji, zvuk i drugi. Sva tehnička i teorijska izvedba pojedinih aspekata igre popraćena je konkretnim primjerima.

Nakon izrade igre, slijedi istraživanje, razvoj i implementacija agenata strojnog učenja. Aspekt umjetne inteligencije omogućen je pomoću alata *ML-Agents* koji svoj princip rada dijeli na *Unity* i *Python*. *Unity* ima ulogu simulatora koji kreira podatke, a *Python virtualno okruženje* ulogu mozga koji prikupljene podatke obrađuje i šalje nazad simulatoru. Tako se nakon poglavlja *Unity platforma* nalazi poglavlje *Teorijska pozadina višeagentnih sustava* u kojem su teorijski objašnjeni svi pojmovi korišteni prilikom rada s višeagentnim sustavima. Pod time spadaju umjetna inteligencija kao krovni pojam koji pod sobom ima mnoge grane i načine primjene, potom njezine grane, kao i zasebna grana stojnog učenja te višeagentni sustavi kao takvi.

Razvoj i implementacija višeagentnih sustava provedeni su na način da je kreiran zaseban dio mape koji je procesom prilagodbe postao okruženje za učenje agenta. Svaki dio mape imao je neko značenje za agenta, bilo to dobro ili loše. Pod time se smatra da su simulacije vođene poticanim učenjem, gdje se agenta nagrađuje za dobre, a kažnjava za loše odluke, odnosno akcije. Sam ciklus učenja odvija se u fazama opažanja, odlučivanja, akcije i nagrađivanja. Tako se kroz niz simulacija agenta navodilo da sam savlada sve zadatke koje su stavljeni pred njega i time optimizira svoje iskustvo učenja. Svi navedeni postupci nalaze se u poglavlju *Implementacija i analiza višeagentnog sustava*, gdje je kao dodatak provedena analiza svih ključnih trenutaka u kojima je agent ostvario svoj napredak.

Motivacija za odabir ove teme je zainteresiranost računalnim igrama, ali i umjetnom inteligencijom. Svakim danom se nalazi sve više programera koji sami izrađuju igre, što je prije bilo nezamislivo. Cilj je bio učiniti imaginarne ideje stvarnima, ali i iskoristiti alat *Unity* do njegovog potencijala i vidjeti može li se ostvariti stvarno sve što se zamisli. Također igra sa zombijima se činila kao idealna prilika za upoznavanje sa umjetnom inteligencijom, odnosno višegantnim sustavima. Pogotovo kako svakim danom sve više različitih aplikacija na jedan ili drugi način iskorištava snagu umjetne inteligencije.

2. Metode i tehnike rada

Način i pristup izrade ovog rada podjeljen je na 3 dijela. Programski dio, grafički dio te dio koji se odnosi na izradu i implemenataciju višeagentnog sustava. Programski dio izrade računale igre odrađen je pomoću progamskog jezika *C#* unutar programskog okruženja *Visual Studio*. Skripte koje su tamo kreirane i uređivane, kasnije su primjenjene u alatu *Unity* nad raznim objektima, kao njihov sastavni ili općeniti dio. Grafički dio odrađen je pomoću alata *Unity* koji uz svoje osnovne geometrijske oblike (kocka, krug, valjak itd.) nudi, putem funkcionalnosti *Package Manager*, besplatne i plaćene funkcionalnosti u smislu samih objekata u igri kao što su na primjer kuće, oružja, likovi itd., ali i razni vizualni efekti, fontovi, zvučni efetki, glazba i ostali alati koji olakšavaju tehnički razvoj. Kao jedan od paketa, dostupan putem stranice *GitHub*, razvijen kao službeni alat od *Unity-a* je *ML-Agents*. Tim alatom napravljene su razne konfiguracije i simulacije nad agentima koji su kasnije implementirani u igri.

ML-Agents sam po sebi, kao alat, ima mnoge funkcionalnosti koje su korištene na više strana. Putem programskog jezika *C#* korištene su biblioteke s metodama namjenjenim za simulacije i konfiguracije agenata. Specifično za alat *Unity* postoje skripte koje se automatski dodaju nakon dodavanja agenata te znatno olakšavaju njihov razvoj. Uz *C#* i *Unity*, alat dolazi sa skriptama razvijenim u programskom jeziku *Python*, gdje se kroz virtualno okruženje vrši njihovo izvođenje, a služe kao direktna poveznica između skupa podataka koje agent prikupi kroz simulacije i samih simulacija koje se izvode unutar *Unity-a*.

Uz tehnički dio bilo je potrebno razraditi konceptualni dio, koji se odnosi na izgled mape na kojoj igrač igra, tok igre te sve ostale apspekte kao što su sučelja, zvukovi, animacije i slično. Na finalni izgled mape utjecalo je više faktora. Primarni fokus bio je na tome da je mapa zapravo poligon za razvoj agenata. Sekundarni fokus bio je na tome da sama po sebi bude zabavna, korisna i da postepeno uvodi igrača u igru. Prilikom dizajna mape i izrade toka igre, idejno su razmotrene razne poznatije igre, videi i članci povezani s višeagentnim sustavima te ostali izvori.

3. Unity platforma

U ovom dijelu objašnjene su pojedinosti o samom razvojnom alatu *Unity*, kao i postupak razvoja tehničke strane igre čiji glavni cilj je implementacija i analiza višeagentnih sustava. Kroz prikaze i objašnjenja važnijih dijelova igre objašnjen je način funkcionairanja *Unity-a*, odnosno većina funkcionalnosti koje nudi. Pod to spadaju rad s objektima, grafika, vizualni i zvučni efekti, animacije te ostali alati koji nisu nužno prikazani u korisničkom sučelju nego kroz biblioteke korištene prilikom programiranja u jeziku C#.

3.0.1. Karakteristike Unity-a

Unity je više platformski alat za razvoj igara. Kao takav podržava razvoj igara za osobna računala, mobitele, razne konzole te sustave koji općenito podržavaju virtualnu stvarnost. Postoje razne verzije samog programa u smislu njegove cijene, no za izradu ovog projekta diplomskog rada bila je dovoljna besplatna verzija, koja je također dovoljna i za većinu programera koji su se tek počeli baviti razvojem igara. Što se tiče tehničke strane, glavni procesi koji se izvršavaju prilikom rada igre podjeljeni su u nekoliko metoda. One koje se izvršavaju na samom početku, odnosno učitavanju određenog dijela igre jesu *Awake() i Start()* i pozivaju se samo jednom na početku. Razlike su što se *Awake()* poziva bez obzira bila skripta, na inicijaliziranom objektu, aktivna ili ne i to uvijek prije *Start()* metode. Tokom rada igre izvršavaju se *Update(), FixedUpdate() i LateUpdate()*, gdje se *Update()* izvršava svaku sliku u sekundi, a *FixedUpdate()* više ili manje puta od *Update()* metode jer je vezana za fiksno vrijeme koje može biti kraće ili duže od slike u sekundi te se iz tog razlika koristi za primjenu fizike u igrama. Metoda *LateUpdate()* izvršava se nakon što se svi procesi, aktivirani unutar *Update()* metode, završe i sva stanja izmjene, tako da se u toj metodi mogu, na primjer, provjeravati određeni rezultati ili ishodi igre. Alat je poznat po sljedećim funkcionalnostima koje ga čine vrlo kompaktnim, vrlo jednostavnim i intuitivnim za korištenje.

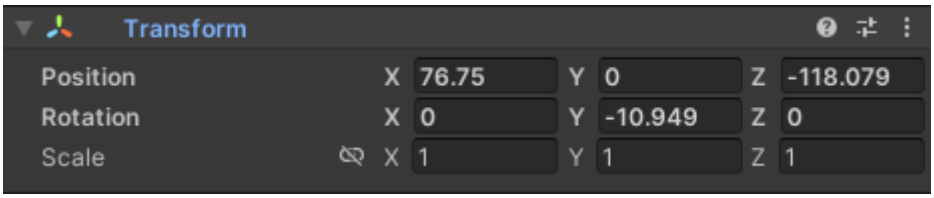
3.0.2. Objekti

Cijela logika razvoja bazira se na objektima koji se prilikom učitavanja igre instanciraju odmah ili kasnije, ovisno o želji programera, koji može direktno ili indirektno njima manipulirati. Pod direktno, misli se na skripte koje su kroz korisničko sučelje, odnosno radni prozor, pridružene na sam objekt te nije potrebna eksplicitna referenca na njih. S druge strane ako se želi putem skripte, odnosno programskog koda manipulirati objektom koji nije direktno povezan skriptom, tada za to postoje razni pristupi. Dva najviše korištena pristupa u ovom rad jesu, referenciranje na objekt putem skripte u kombinaciji sa korisničkim sučeljem i pronalaženje aktivne skripte u sceni. Prvi pristup odnosi se na definiranje varijable anotacijom *SerializeField* koja omogućuje pristup i uređivanje stanja varijable kroz korisničko sučelje gdje se unosom objekta u definirano polje može pristupiti svim njegovim atributima, od kojih je najvažniji *Transform* i *Rect Transform* (samo kod objekata koji čine korisničko sučelje igre) kojim se mijenja rotacija, pozicija i veličina samog objekta, a od ostalih atributa to mogu biti, boja, vidljivost, ali i bilo koja

skripta koja se nalazi na njima. Drugi pristup odnosi se na traženje aktivne skrite predefiniranom metodom *FindObjectOfType<Type>()*. U tom slučaju objekt, koji ima traženu skriptu na sebi, mora biti aktivan u samoj sceni, u protivnom će program izbacivati null iznimke. Objekti su zapravo sve ono što igrač vidi, ali i ne vidi na ekranu [1].

```
[SerializeField] private Slider ammoSlider;
```

Kod 3.1: Definiranje varijable dostupne u korisničkom sučelju Unity-a



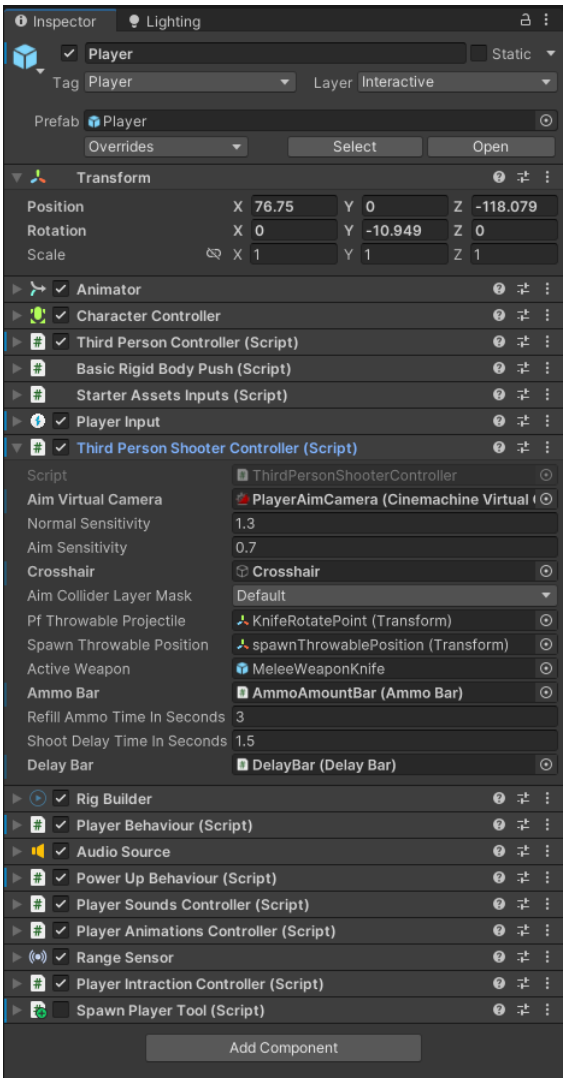
Slika 1: Opcije Transform objekta

Slika 2: Opcije Rect Transform objekata korisničkog sučelja

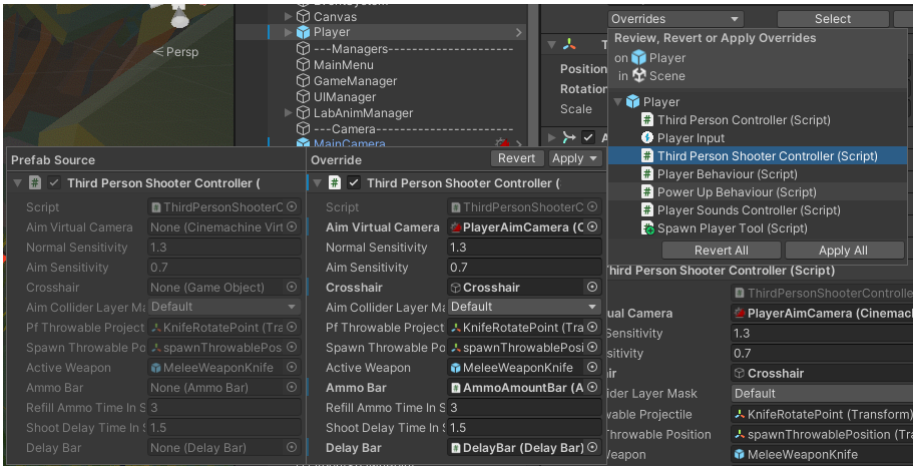
```
MLabInfoManager mLabInfoManager = FindObjectOfType<MLabInfoManager>();
```

Kod 3.2: Traženje aktivne skripte objekta

Sve vezano za jedan objekt, kao što su razne skripte, konfiguracije i specifikacije, dostupno je u prozoru *Inspector*. Tamo se mogu kroz pojedine skripte uređivati dostupne varijable, kao što je spomenuto ranije, ali se mogu uređivati i predefinirane specifikacije objekta kao što su postavke kolizije, zvukovi, razni senzori, fizika objekta, skripte potrebne za rad s višeagentnim sustavima itd. Također ako pojedina skripta nije potrebna ili je viška ne mora se nužno obrisati, nego se može samo označiti kao neaktivna što uvelike olakšava buduće preinake. Objekti mogu biti samostalni ili predefinirani. Samostalni objekti su oni koji se koriste jednom i svaka promjena je primjenjena samo na taj jedan objekt, dok su predefinirani objekti instance jednog specifičnog objekta te se može birati hoće li promjene biti aplicirane samo na jedan specfičan objekt ili na sve njih, slika 4. Važno je napomenuti da se kroz prozor inspektora može promjeniti naziv, oznaka i sloj na kojem se objekt nalazi.



Slika 3: Prozor inspektora



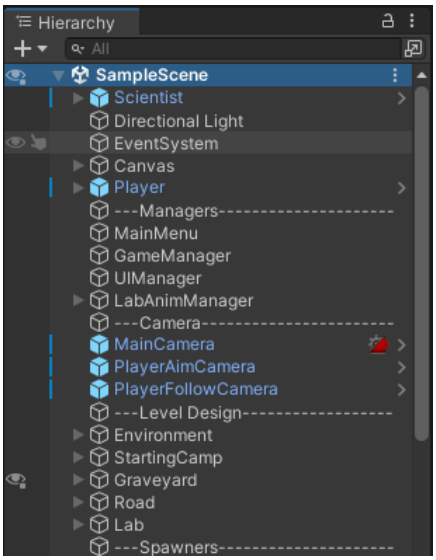
Slika 4: Prozor promjena objekta

3.0.3. Scena i hijerarhija

Svi objekti nalaze se u sceni, slika 5. Scena je dio radnog prozora *Unity-a* gdje su prikazani svi aktivni objekti. Pod aktivni, misli se na objekte s kojima je moguće manipulirati, odnosno mogu biti u interakciji s ostalim objektima. To je glavni prozor za uređivanje raznih aspekata igre, od korisničkog sučelja, do izrade mape s objektima.

Slika 5: Prozor scene

Kada se govori o opcijama koje nudi važno je spomenuti da postoji globalna i lokana orjentacija objekta. Pod globalnu orjentaciju odnose se x, y i z osi umjerene s obzirom na globalnu perspektivu scene. Lokalna orjentacija je orjentacija objekta u odnosu na globalnu perspektivu. Prilikom rada u sceni također je važan alat koji omogućuje prikazivanje ili sakrivanje raznih elemenata radnog prozora, jer na jednom objektu može biti mnogo stavki, kao što su razni senzori te se oni iscrtavaju u radnom prozoru i ponekada mogu smetati prilikom izrade mape i slično. Također svi objekti nalaze se unutar hijerarhije, slika 6, koja je usko vezana sa scenom. U hijerarhiji se zapravo nalaze svi objekti koji jesu ili nisu prikazani u sceni, tako da je ona na neki način općenitiji prikaz od same scene [2]. Unutar nje se objekti mogu posložiti u odnosima roditelj - dijete objekt i tako se grupirati, što je na sceni često lako uočljivo jer se grupe objekata mogu zajedno, odjednom uređivati u smislu rotacije, pozicije, veličine, vidljivost itd. te nije potrebno uređivati svaki objekt zasebno.



Slika 6: Prozor hijerarhije

3.0.4. Dizajn

Prilikom izrade računalne igre primjenjeni su razni dizajni vezani uz izgled likova, zvučne i vizualne efetke, korisničko sučelje i izbornike. Većina objekata koji su korišteni prilikom izrade preuzeti su putem funkcionalnosti *Unity Package Manager* kojim se direktno sa službenog *Unity* internetskog dućana objekti uvoze u razvojno okruženje i odmah su dostupni za rad. Nekoliko njih uvezeno je na način da su preuzeti sa službene stranice autora i ručno uvezeni u foldere projekta.

3.0.4.1. Animacije i Eventi

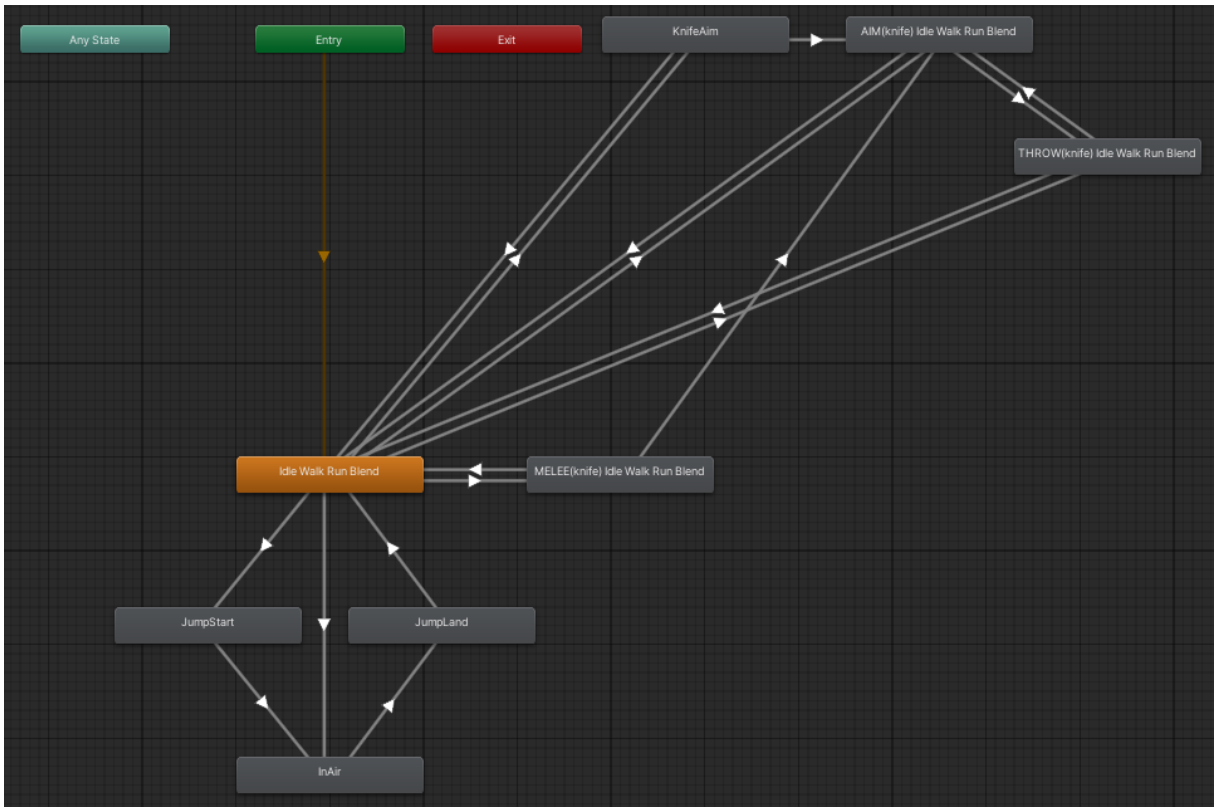
Kao što je prije spomenuto, tako i kod animacija postoji više načina pristupa. Kroz ovaj rad, jednim dijelom, korištene su uvezene, a drugim dijelom ručno dorađene ili napravljene animacije. Dio koji je uvezen jesu animacije povezane s osnovnim pokretima glavnog lika kojeg upravlja igrač, a to su skok, slika 7, padanje, hodanje, trčanje i stajanje na mjestu. Na te uvezene animacije dodatno su napravljene animacije ciljanja, slika 8, i stajanja/hodanja/trčanja te bacanja, u ovom slučaju, noža i stajanja/hodanja/tračanja te napad na blizinu i stajanje/hodanje/trčanje.

Slika 7: Animacija skoka

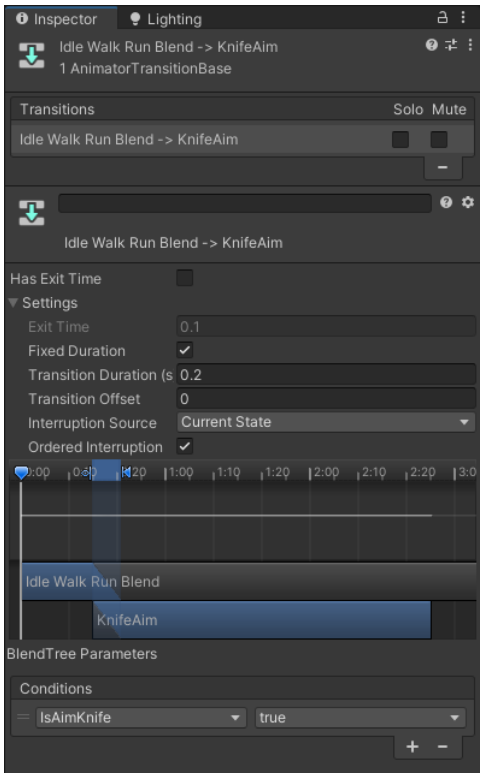
Slika 8: Animacija ciljanja

Dva radna prozora koja su važna kako bi se objekt povezao s animacijama, kako bi se konfigurirale te naposljetku izrađivale ili dorađivale animacije jesu *Animator i Animation*. Prozor *Animator* je glavni prozor za upravljanje i povezivanje animacija s određenim objektom. Stavke animacije povezane su u mrežu, slika 9, u kojoj povezne linije između stavki predstavljaju uvjeti njihovog početka izvođenja te razne konfiguracije tranzicija između animacija. Svaka stavka animacije ima zasebne postavke izvođenja kao što su sama animacija, brzina izvođenja te popis svih drugih animacija u koje je predviđeno da ide tranzicija, bilo po završetku izvođenja ili pod određenim uvjetima koje programeri mogu sami odrediti.

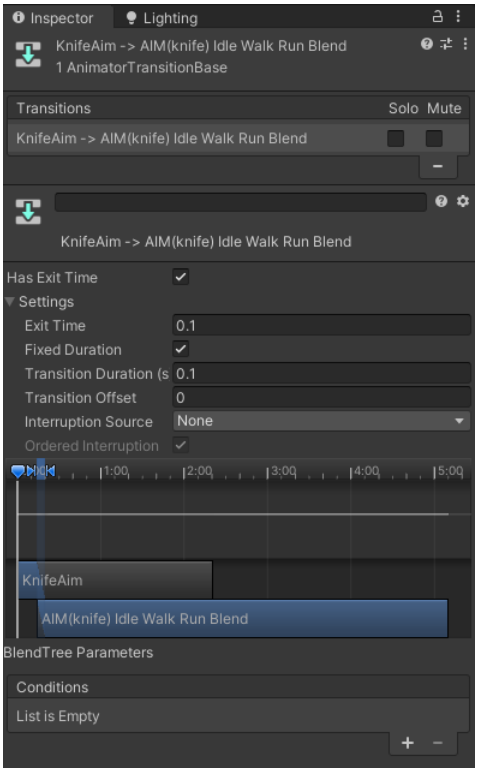
Primjer može biti izvedba animacije ciljanja. Kada igrač pritisne desni klik miša tada se aktivira ciljanje i omogućeno je bacanje noževa. S druge strane kada igrač ne cilja ne može niti bacati nož već samo izvršiti napad na blizinu. Što se tiče animacije bacanja noža, lik s kojim igrač upravlja je prvobitno u animaciji mirovanja gdje se oponaša disanje, potom se promjenom brzine kretanja lika prelazi iz jedne u drugu animaciju hodanje/trčanje, što omogućava funkcionalnost *Blend Tree*, slika 12. U trenutku kada uvjet za ciljanje, bool varijabla *IsAimKnife*, postane true, odnosno igrač pritisne desnu tipku miša, animacija se odmah mijenja iz početne u animaciju ciljanja, slika 10. Kada otpusti tipku miša, animacija se odmah mijenja iz animacije ciljanja u animaciju običnog bivanja lika. U slučaju da igrač stisne i odmah otpusti tipku ili ju drži određeno vrijeme te ju onda otpusti, događa se jednaka stvar jer u konfiguracijama izlazne animacije postavka *Has Exit Time* postavljena je na false, odnosno odznačena je, što znači, čim je uvjet ispunjen pređi u drugo stanje, slike 11 i 13.



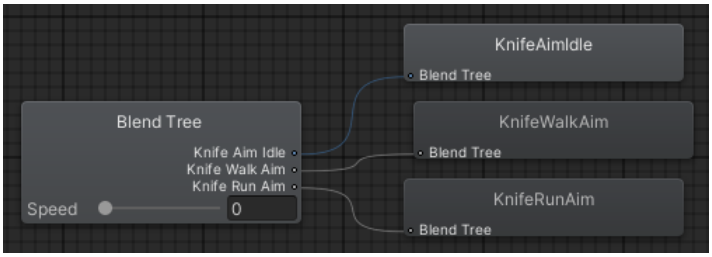
Slika 9: Prozor animatora



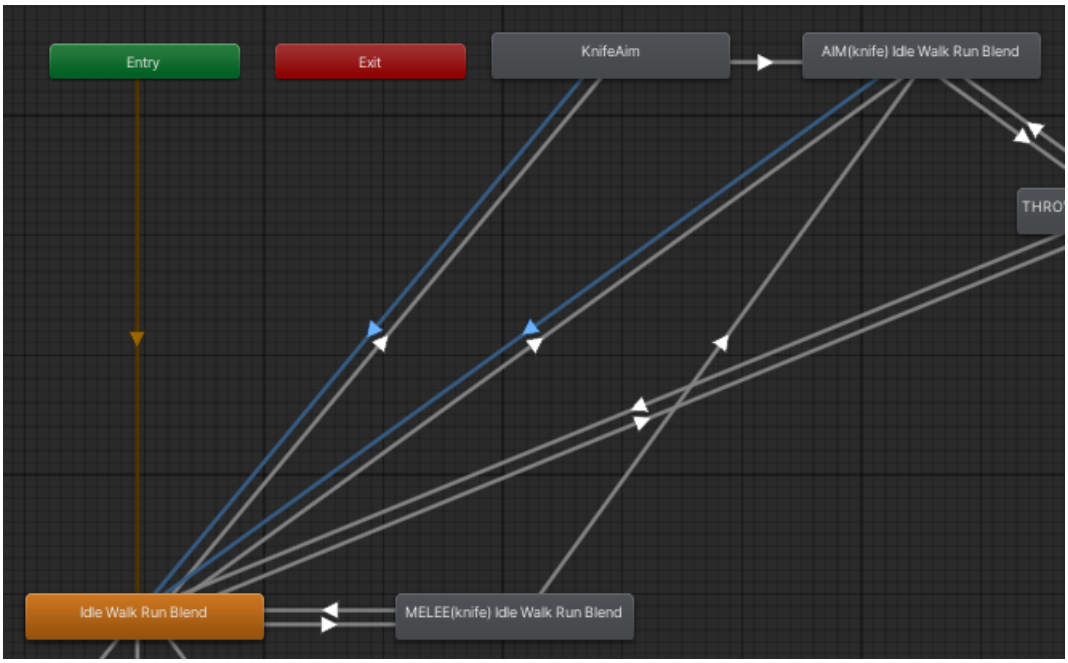
Slika 10: Postavke prijelaza animacije pod uvjetom



Slika 11: Postavke prijelaza animacije bez uvjeta



Slika 12: Prozor blend tree-a

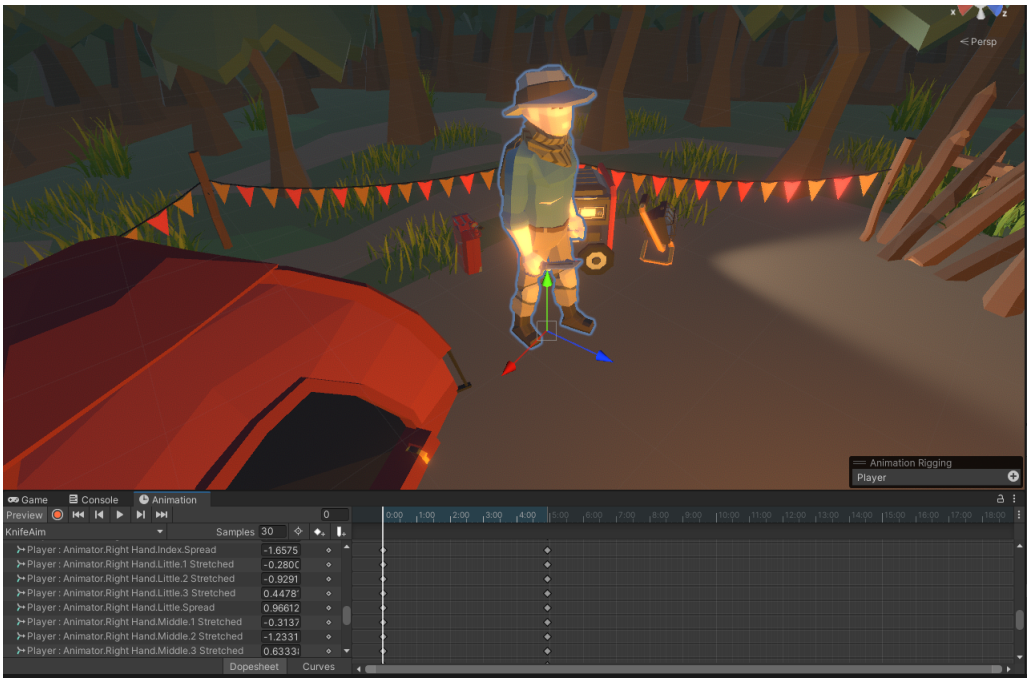


Slika 13: Različite stavke animacije s istim uvjetom prestanka izvršavanja

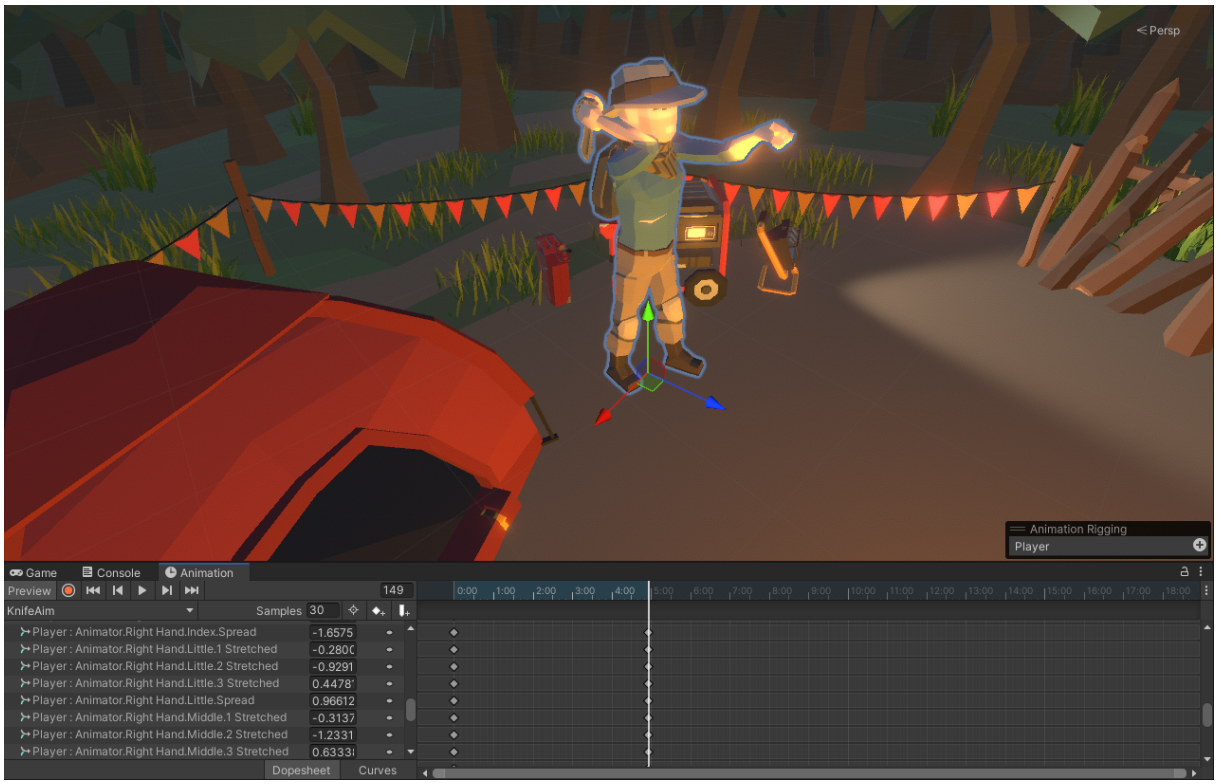
```
private Animator animator;

private void Awake()
{
    animator = GetComponent<Animator>();
}

private void Update()
{
    if(starterAssetsInputs.aim)
    {
        animator.SetBool("IsAimKnife", true);
    }
    else
    {
        animator.SetBool("IsAimKnife", false);
    }

}
```

Kod 3.3: Promjena uvjeta za izvođenje animacije

Prozor *Animation* jednako je važan kao *Animator*, no u njemu se izrađuju sami pokreti animacija i to za svaku animaciju zasebno. Na primjeru animacije ciljanja, u prozoru *Animation* nalaze se vremenska skala te točke koje predstavljaju kada se koji dio skeletona lika bude aktivirao ili deaktivirao. Na primjer, od nulte do pete sekunde kosti desne i lijeve ruke će promijeniti svoju poziciju i rotaciju iz početne, slika 14 u finalnu, slika 15. Također taj radni prozor nudi snimanje pokreta te kada se pokrene snimanje animacije i pomakne dio skeleta objekta ili bilo koji drugi objekt kao cjelina, tada se to zabilježi kao promjena pozicije i rotacije u tom trenutku.



Slika 14: Početak animacije



Slika 15: Kraj animacije

Animacije se također mogu pokrenuti ručno kroz kod što omogućava programeru potpunu kontrolu nad njihovim izvršavanjem, jer se isto kao i objekti, mogu instancirati i manipulirati gdje i kada je to potrebno. Primjer je proces izlječenja zombi agenata, slika 16, gdje igrač sa likom prvo mora doći do kompjutera pritisnuti tipku F kako bi zatvorio vrata od kontejnera, što je prva animacija, te nakon toga nakon određenog vremena krene dim što predstavlja ispuštanje kemikalije u kontejner. Kada je proces završen vrata kontejnera se otvaraju i zombi agenti nestaju. Tehnička izvedba sastoji se od više skripti koje su indirektno povezane putem *Unity EventHandler-a*. To je funkcionalnost, iz službene *Unity* biblioteke *UnityEngine*, koja omogućava jednostavno okidanje raznih dijelova koda u raznim skriptama po principu pretplate. Prva skripta koja ulazi u ovaj proces je *PlayerInteractionController* koja se nalazi na objektu glavnog lika te uz pomoć alata *SensorToolkit* koji nudi upotrebu i konfiguraciju raznih senzora, provjerava se je li u određenom radijusu lika objekt s određenom oznakom. Ako je, tada se provjerava je li igrač pritisnuo tipku F i tek tada se okida event *OnHealChamberComputerPressed* koji šalje obavijest svim pretplaćenim skriptama kako bi se u njima moglo nadalje izvršiti što je već predviđeno.

```
using UnityEngine;

public event EventHandler OnHealChamberComputerPressed;

if (playerSensor.Detections.Count != 0)
{
    foreach (var detection in playerSensor.Detections)
    {
        ...
```

```
        else if (detection.CompareTag("Computer"))
        {
            UIManager.uiManager.isItemAvailable = true;
            UIManager.uiManager.itemToPickUp = detection.tag;
            if (Input.GetKeyDown(KeyCode.F))
            {
                OnHealChamberComputerPressed?.Invoke(this, EventArgs.Empty);
            }
        }
        ...
    }
}
```

Kod 3.4: Okidanje eventa

Pretplaćena skripta je *HealChamberDoorController* koja se nalazi na objektu vrata od kontejnera. Pretplata funkcionira na način da se prvo nađe skripta u kojoj je event definiran te se tada samo metoda nadoda u *EventHandler* objekt. Kroz ovaj rad izvršavanje preplate na evente odrađivano je u *Awake()* metodi, ali ponekad i u *Start()* metodi.

```
private bool canHealChamberDoorsBeClosed = true;

void Awake()
{
    PlayerIntractionController playerIntractionController = FindObjectOfType<
        PlayerIntractionController>();
    playerIntractionController.OnHealChamberComputerPressed +=
        PlayerIntractionController_OnHealChamberComputerPressed;
}

private void PlayerIntractionController_OnHealChamberComputerPressed(object sender,
    System.EventArgs e)
{
    canHealChamberDoorsBeClosed = true;
}
```

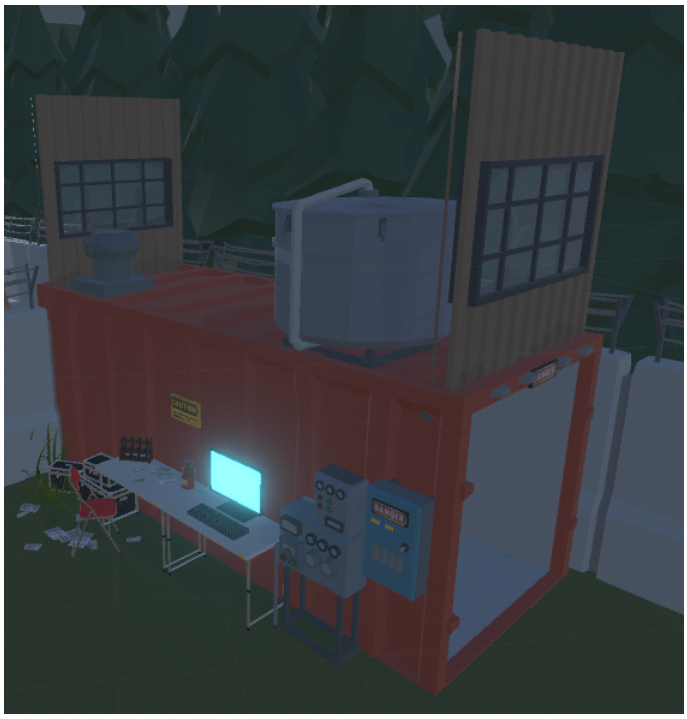
Kod 3.5: Princip pretplate koristeći EventHandler

Kada su vrata kontejnera zatvorena izvršava se event *OnChamberDoorClose*. Skripta koja je pretplaćena na taj event nalazi se na objektu unutar hijerarhije koji je nevidljiv i glavna svrha mu je da instancira skriptu *LabAnimController*. Unutar nje se nalazi ručno intanciranje animacije. Za to je potreno kao i prije, izvršiti pretplatu na ranije spomenuti event i potom unutar pretplaćene metode pomoću *unity* medode *Instantiate()* instancirati animaciju. Metoda kao parametre prima *Transform* od animacije, poziciju instanciranja te rotaciju instanciranja.

```
[SerializeField] public Transform vfxHealSmoke;
[SerializeField] private GameObject healSmokePosition;

private void Start()
{
    HealChamberDoorController healChamberDoorController = FindObjectOfType<
        HealChamberDoorController>();
```

```
    healChamberDoorController.OnChamberDoorClose +=
        HealChamberDoorController_OnChamberDoorClose;
}

private void HealChamberDoorController_OnChamberDoorClose(object sender, System.
    EventArgs e)
{
    Instantiate(vfxHealSmoke, healSmokePosition.transform.position, Quaternion.
        identity);
}
```

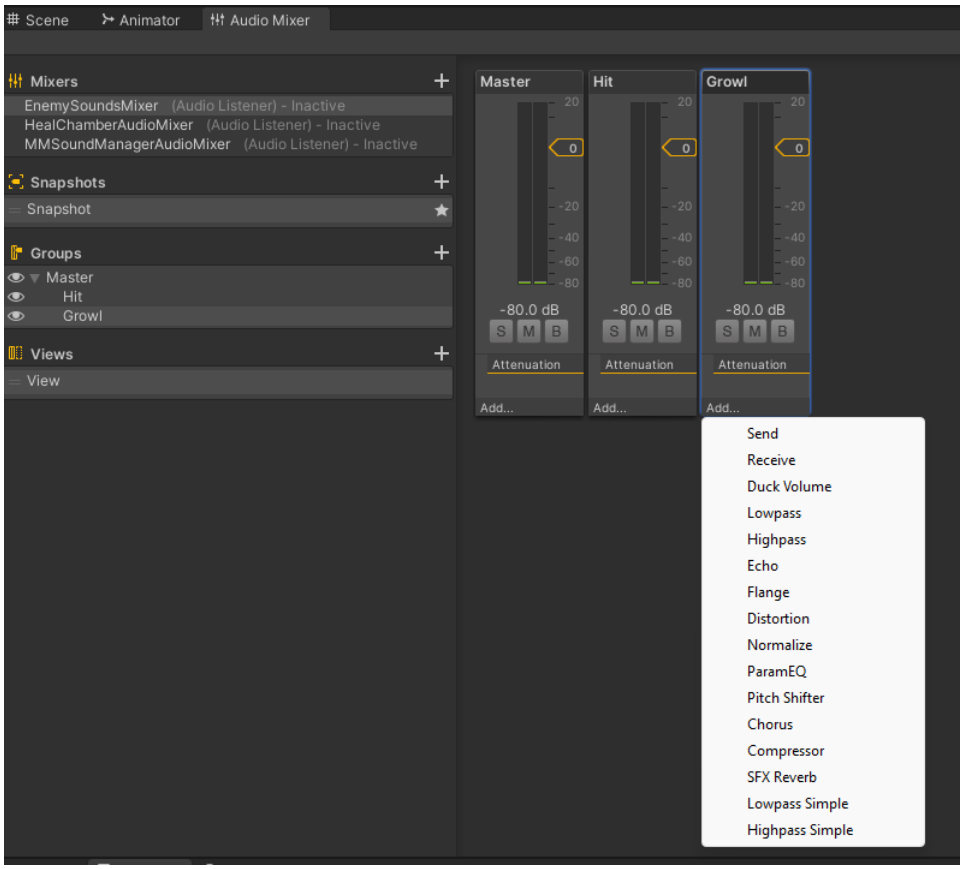
Kod 3.6: Ručno instanciranje dima kontejnera



Slika 16: Kontejner za izlječenje zombi agenata

3.0.4.2. Zvuk

Slično kao i animacije funkcionalnost zvuka u *Unity-u* vrlo je intuitivno za konfigurirati, implementirati te koristiti. Kao i kod animacija postoji radni prozor *Audio Mixer* gdje se zvukovi mogu smanjivati, pojačavati te dodavati razni efekti, slika 17. Ta funkcionalnost koristila se kako bi se postiglo da se zvukovi zombija i rada kontejnera stišavaju proporcionalno distanci glavnog lika od njih. Tehnička izvedba je osmišljena tako što je dodana zasebna skripta *EnemySoundsController* na objekt zombija koja putem ranije spomenutih evenata prati udaljenost glavnog lika te ako je *distanceFromPlayer* varijabla manja od minimalne udaljenosti pojačava stavku u *Audio Mixer-u* na najviše, ako je veča od maksimalne udaljenosti tada skroz smanji, te ako je između minimalne i maksimalne tada proporcionalno pojačava i smanjuje.



Slika 17: Audio Mixer

```
[SerializeField] private AudioMixer enemySoundsMixer;

private void EnemyBehaviour_OnPlayerNearZombie(object sender,EnemyBehaviour.
    OnPlayerNearZombieEventArgs e)
{
    distanceFromPlayer = e.distance;
}

private void Update()
{
    if (distanceFromPlayer != 0)
    {
        if (distanceFromPlayer < minZombieGrowlRange)
        {
            enemySoundsMixer.SetFloat(MIXER_GROWL, Mathf.Log10(maxSoundVolume) * 20)
                ;
            if (!isZombieGrowl)
            {
                StartCoroutine(PlayZombieSounds());
            }
        }
        else if (distanceFromPlayer > maxZombieGrowlRange)
        {
            enemySoundsMixer.SetFloat(MIXER_GROWL, Mathf.Log10(minSoundVolume) * 20)
                ;
```

```
        }
        else if (distanceFromPlayer > minZombieGrowlRange)
        {
            enemySoundsMixer.SetFloat(MIXER_GROWL, Mathf.Log10(minZombieGrowlRange /
                distanceFromPlayer) * 20);
            if (!isZombieGrowl)
            {
                StartCoroutine(PlayZombieSounds());
            }
        }
    }
}
```

Kod 3.7: Reguliranje zvuka putem Audio Mixera

Uz to postoji funkcionalnost *AudioClip* koja se dodijeli svakom objektu koji emitira određeni zvuk [2]. Tako su svi glavni zvukovi vezani uz glavnog lika regulirani preko skripte *PlayerSoundsController* koja se nalazi na njemu. Ponovo u kombinaciji sa eventima, kada se u kodu okine određena radnja kao što su otvaranje vrata, bacanje noža, udarci, zvukovi korisničkog sučelja i drugi, pomoću predefinirane metode *PlayOneShot()* od klase *AudioSource*, jednom se pokreće određeni zvuk određene glasnoće.

```
[SerializeField] private AudioClip knifeThrowSound;
[SerializeField] private float knifeThrowSoundVolume;

private AudioSource playerAudio;

void Start()
{
    playerAudio = gameObject.GetComponent<AudioSource>();
}

private void KnifeThrowEvent_OnKnifeThrow(object sender, System.EventArgs e)
{
    playerAudio.PlayOneShot(knifeThrowSound, knifeThrowSoundVolume);
}
```

Kod 3.8: Manipulacija zvuka pomoću AudioClip-a

3.0.4.3. Izgled zombi agenata

Sam dizajn objekta zombija, dio animacija i zvukovi preuzeti su sa službenog *Unity internetskog dućana*. Ideja je bila napraviti dvije vrste zombia, dobre i loše, između kojih postoji nekoliko razlika, slika 18. Zli zombiji rade više štete pri udarcu te ne mogu biti izlječeni u kontejneru već samo nestanu i broje se kao dodatna eliminacija, dok dobri zombiji također nestanu ali se broje kao bodovi izlječenja. Što se tiče izgleda, dobri zombiji imaju nešto svjetliju krv i bijele oči, dok zli zombiji imaju nešto tamniju krv i žute oči. Varijacije su prvenstveno napravljene u svrhu zabavnog faktora kojeg igra može pružiti, ali i provjere jednostavnosti primjene višeagentnog sustava na različite objekte.

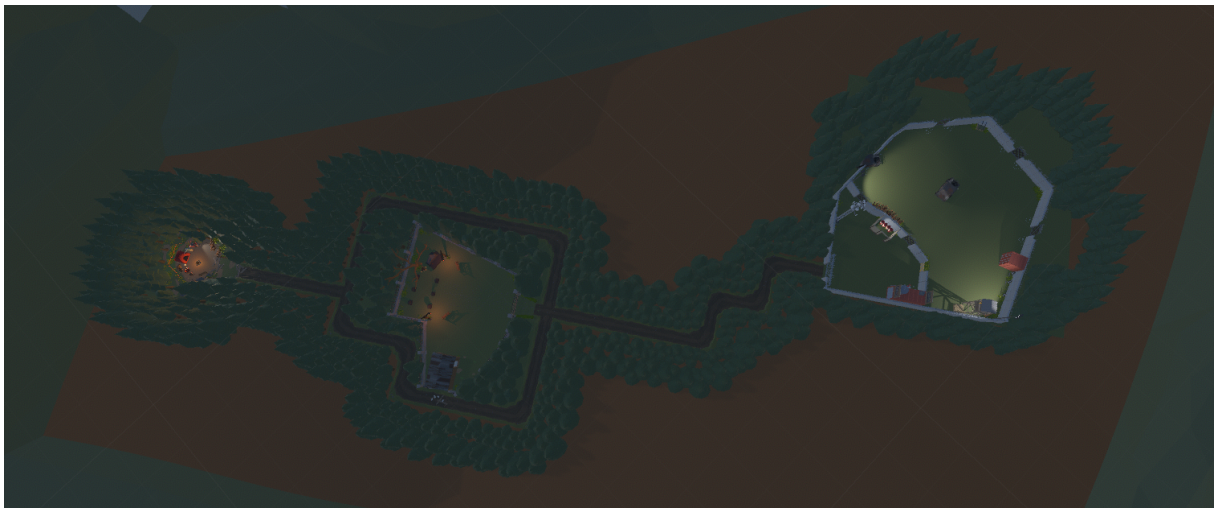
Slika 18: Izgled zombia

3.0.5. Tok igre

U ovom podpoglavlju objašnjene su ukratko pojedinosti vezane uz tok igre kao što su ideje iza dizajna mape, kako je sama mapa osmišljena te kako može utjecati na iskustvo igranja igrača. Također uz to prikazana su korisnička sučelja razvijena za potrebe igre, ali i izbornici koji vode do same igre.

3.0.5.1. Mapa

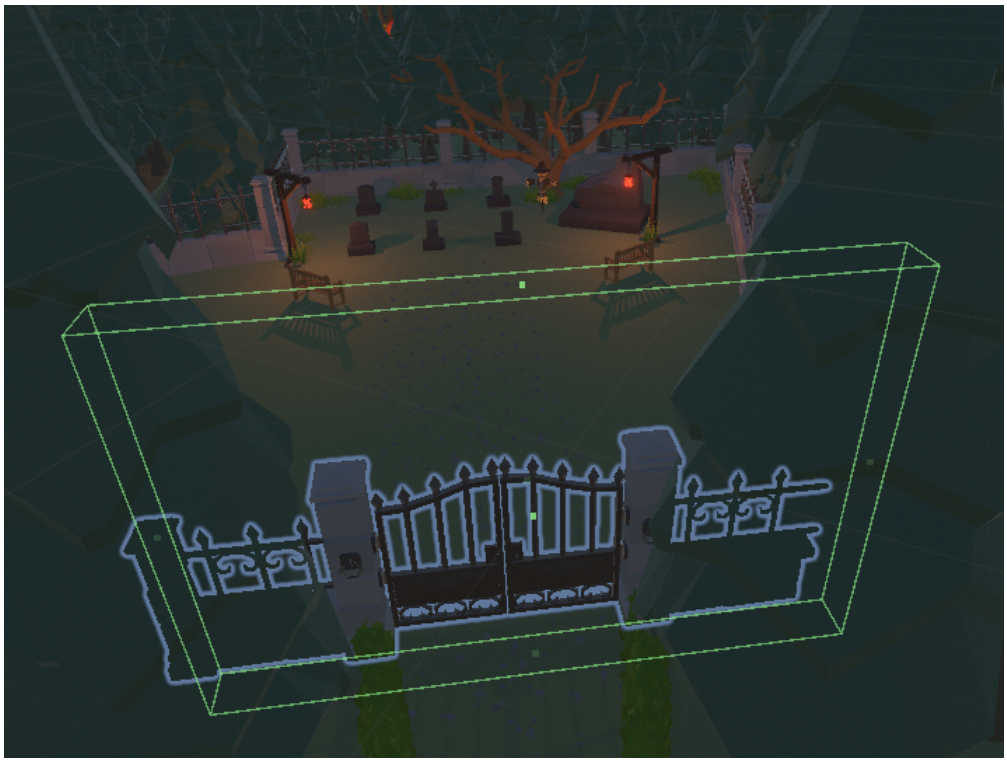
Uz sve do sada objašnjene aspekte igre, mapa je onaj glavni dio koji igrača smješta u virtualni svijet i vodi ga kroz iskustvo igranja. Uz zabavni faktor mapa je razvijena uz pretpostavku da će na njoj biti provedena učenja agenata te iz tog razloga ima mjesta gdje se agenti stvaraju, razne prepreke i točke interesa koje igrača vode kroz igru. Podjeljena je u tri glavne cjeline, a to su početak igre, srednji dio i završni dio, slika 19.



Slika 19: Mapa

Početni dio je zamišljen kao kamp glavnog lika, odnosno uvodni dio gdje se igrač kroz

isprobavanje upoznaje sa osnovnim kontrolama i istražuje svoju okolinu. Srednji dio je zamišljen, osim za prezentaciju agenata, kao dio gdje se igrač po prvi puta susreće za zombijima i mora ih sve eliminirati kako bi mu se otkrio ključ od vojne baze, odnosno završnog dijela. Vojna baza je dio gdje se odvija glavni dio elimininiranja zlih i izlječivanje dobrih zombija, odnosno gdje je prisutan puni potencijal razvijenih agenata. Sa tehničkog stajališta važno je spomenuti da su dijelovi mape koji ne spadaju u dio igre ograđeni objektima koji na sebi imaju jednu vrstu *sudarača (eng. collidera)*. To je *Unity* funkcionalnost koja omugućuje detekciju kolizije objekata. S time se mogu regulirati razni sustavi kao što su život glavnog lika, udarci, tok igre, animacije, zvukovi te ostale pojedinosti. Na primjer, kada zombi ili glavni lik dođe do *Collidera*, tada njihovi *Collideri* blokiraju prolaz kroz druge i na taj način djeluju kao barijera, slika 20.



Slika 20: Prikaz collider-a

3.0.5.2. Korisničko sučelje

Kako igrač dolazi sa jednog izbornika na drugi ili što sve vidi od informacija na ekranu, također je važno za dobro iskustvo igranja. Glavna funkcionalnost za izradu korisničkog sučelja je *Canvas*, slika 21. Canvas se također kao i bilo koji drugi element sastoji od objekata i skripti, samo se u ovom slučaju radi o objektima slike i teksta. Svi ti elementi uređuju se u *Sceni* kao i ostali objekti kao što su glavni lik, zombiji, okolina i slično. Također mogu se učiniti aktivnima i neaktivnima što je korisno kada se u određenom dijelu igre ne želi prikazati određena stavka korisničkog sučelja ili se želi prikazati ali sa izmjenjenim tekstom ili slikom. Za ovaj rad izrađeno je nekoliko korisničkih sučelja, od toga jedno je vidljivo tokom igre na kojem se nalaze količina životnih bodova, količina noževa za bacanje, broj bodova eliminacija i izlječenja te ostale stavke kao indikacija ubrzanja nakon pokupljenog pojačanja i druge. Od korisničkih sučelja izbornika

to su početni izbornik, izbornik pauze, ekran pobjede i ekran gubitka, slike 22, 23, 24.

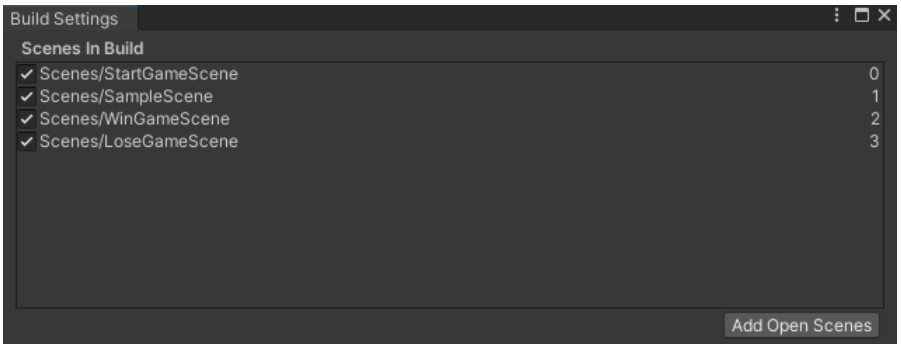
Slika 21: Canvas

Slika 22: Početni izbornik

Slika 23: Ekran pobjede

Slika 24: Ekran poraza

Sa stajališta tehničke izvedbe prelaz sa jedne na drugu scenu, odnosno ekran, radi se na način da se prvo u *Build Settings-ima* od Unity-a postave sve scene koje će biti korištene tokom igre, slika 25 i potom u kodu određuje njihovo izvršavanje. Unutar skripte *MainMenu* nalaze se izrađene metode *NextScene(), ExitGame() i druge* te se pomoću *Unity* alata *SceneManager* metodom *LoadScene* pristupa dodanim scenama čiji se indexi tada povečavaju ili smanjuju čime se određuje koja će scena biti prikazana. Ima više načina prikaza scena, no za potrebe ovog rada odabran je već spomenuti način iz razloga jednostavnosti funkcionalnosti koje izbornici pružaju.



Slika 25: Build Settings

```
public void NextScene()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}

public void ExitGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
}

public void WinGoMenu()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 2);
}

public void LoseGoMenu()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 3);
}

public void LoseScene()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
}

public void QuitGame()
{
    Application.Quit();
}
```

Kod 3.9: Manipulacija scenama

4. Teorijska pozadina višeagentnih sustava

Ovo poglavlje služi kao teorijski uvod u umjetnu inteligenciju i njezine aspekte koji su korišteni za tehničku realizaciju agenata, odnosno zombija. Objašnjeni su pojmovi umjetne inteligencije, njezinih grana, strojno učenje kao jedna od grana te višeagentni sustavi.

4.0.1. Umjetna inteligencija

Pojam umjetne inteligencije je interpretiran na razne načine, no svi se vrte oko riječi inteligencija. Jedna od njih navodi kako je cilj umjetne inteligencije da razvije robote koji će se ponašati kao da su inteligentni [3]. Uzevši tu interpretaciju i razvijen višeagentni sustav ovog rada, to bi značilo da iako se agenti/zombiji kretali po mapi besciljno u krug, bivali na jednom mjestu ili nakon dužeg vremenskog perioda slučajno naletili na lika kojeg upravlja igrač, da su agenti inteligentni. U ovoj interpretaciji nedostaje dio svrhe agenata koja također mora biti ispunjena, a to bi bila pronalazak igrača, trganje ograde ili izlazak iz određenih dijelova mape. Sljedeća interpretacija govori kako je umjetna inteligencija sposobnost računala/agenata da riješe problem koji je, u teoriji, vrlo kompleksan za ljude [3]. Ova interpretacija je nešto bliža agentima razvijenim za potrebe ovog rada, gdje oni u vrlo kratkom vremenskom periodu obrađuju gomilu podataka koje su prethodno primili i na temelju toga donose odluke što napraviti. Dio gdje se obrađuje vrlo velika količina podataka u kratkom vremenu je kompleksan zadatak za ljude, no sam primjer kretanja zombija i nije toliko jer ljudima, nakon što nauče komande i što trebaju raditi kako bi pobijedili u određenoj igri, treba puno krače vremena da to i postignu, dok umjetnoj inteligenciji treba mnogo duže kako bi naučila osnove, no kada ih nauči, u određenim slučajevima, može biti mnogo spretnija od ljudi. Najbliža interpretacija koja je isto tako i bila cilj ovog rada je ta da je umjetna inteligencija način učenja računala/agenata kako da odrade određeni postupak u kojem su trenutno ljudi, u ovom slučaju igrači ove igre, bolji i spretniji [3]. Neki od primjera bili bi kretanje po mapi, izbjegavanje prepreka i zidova, trganje ograde, pronalazak glavnog lika te pokušaj njegove eliminacije.

4.0.2. Grane umjetne inteligencije

Grane na koje se umjetna inteligencija dijeli su strojno učenje (eng. *Machine Learning*), obrada prirodnog jezika (eng. *Natural Language Processing*), računalni vid (eng. *Computer Vision*), ekspertni sustavi (eng. *Expert Systems*), robotika (eng. *Robotics*) te prikaz znanja i zaključivanje (eng. *Knowledge Representation and Reasoning*) [4]. Strojno učenje je usko vezano uz agente koji predstavljaju računalo koje nakon faze opažanja i izrade modela na temelju prikupljenih podataka, formira svoje odluke koje ga dovode do riješavanja problema. U ovom radu se tako koristi jedna vrsta strojnog učenja, točnije Poticano Učenje (eng. *Reinforcement Learning*) gdje je agentu, odnosno zombiju krajnji cilj pronaći igrača. Kako bi to postigao, prvo mora kroz opžanja i izrade modela naučiti riješiti niz manjih ciljeva kako bi postigao ono za što je zapravo treniran. Obrada Prirodnog Jezika vezana je uz funkcionalnosti prijevoda teksta i prepoznavanja glasa. To je zapravo spoj dubokog učenja (eng. *Deep Learning*) i neuronskih

mreža (eng. *Neural Networks*) gdje se riječi grupiraju po njihvom kontekstu pojavljivanja i tako procesiraju u finalne prijevode. Računalni vid radi sa dva glavna problema, rekonstrukcijom i prepoznavanjem, gdje se raznim senzorima mapira okolina promatrača. To zahtjeva rad sa što jasnijim podacima, odnosno slikama dobrih rezolucija ili dobro osvijetljenim područjem promatranja. Rekonstrukcija se odnosi na izradu raznih modela na temelju slika, a prepoznavanje se odnosi na određivanje razlike između objekata promatranja. Ekspertni sustavi koristili su se intezivno između 1969 i 1986 godine, a predstavljali su sustave zadužene za jedno specifično područje za koje se zna većina ishoda i to u raznim djelatnostima. U trenutku kada je napredak tehnologije i društva dostigao razinu gdje su procesi počeli imati velik broj čimbenika, veću razinu kompleksnosti, nesigurnost krajnjih rezultata te kada se tome pridodala nemogućnost učenja sustava iz prošlih iteracija, ekspertni sustavi, kao takvi, jednostavno nisu više zadovoljavali očekivanja industrije. Robotika se odnosi na robote koji se koriste u stvarnom svijetu. Robot kao agent opremljen je sa nekoliko senzora koji prate ulazne podatke o okolini kao što su, na primjer, kamere, radari, laseri i mikrofoni te nekoliko senzora koji prate ulazne podatke samog agenta kao što su, na primjer, žiroskopi, senzori naprezanja i momenta i akcelerometri. Problematika koja se javlja kod njih je ta da se jedino u virtualnom okruženju može napraviti veliki broj iteracija učenja, koje potom može uzeti i do nekoliko godina usavršavanja u stvarnom svijetu. Prikaz znanja i zaključivanje je dio koji povezuje ulazne podatke stvarnog svijeta i računalne obrade tih podataka. Dakle pomaže agentima razumjeti podatke iz okoline kroz formalne metode obrade, interpretirati ih te donositi odluke u raznolikim i složenim okolinama [4].

4.0.3. Strojno učenje

Područje strojnog učenja je usko vezano sa analizom podataka i statistikom. Kako bi se pojasnila povezanost sa ta dva pojma potrebno je razjasniti kako se strojno učenje primjenjuje i što mu je potrebno za rad. Osnova su prošla iskustva koja se stječu kroz brojne iteracije učenja [5]. Kroz ta iskustva agent biva izložen okolini predviđenoj za učenje, iz koje sakuplja podatke. Sakupljeni podaci su tako važni za analize koje slijede i za koje je poželjan što veći i što kvalitetniji skup iz kojeg je moguće agentovo savladavanje krajnjeg cilja. Nakon analize agent bi trebao biti sposoban, na temelju obrađenih podataka, predvidjeti najoptimalniji put do cilja. Primjer koji je E. Alpaydin [6] naveo je procjena cijene rabljenog automobila. Navodi kako je to dobar primjer iz razloga što ne postoji konkretna formula za izračunavanje cijene vozila, već je potrebno uzeti u obzir niz karakteristika i usporediti ih. Neke od karakteristika koje utječu na cijenu su marka, starost, kilometraža, pa čak i karakteristika koja nije direktno povezan sa samim vozilom kao što je stanje ekonomije. Usporedbom rabljenih auta na tržištu može se doći do odnosa cijene i karakteristika, no treba uzeti u obzir da dva različita auta sa istim karakteristikama mogu imati različitu cijenu obzirom na to kako su održavani i voženi. Tako zaključuje, da iako postoji određen broj karakteristika prema kojima se mogu urediti ulazni podaci za učenje agenata, uvijek postoji određen broj neočekivanih faktora koji unose nesigurnost krajnjeg rezultata, što bi za navedeni primjer predstavljala okvirna cijena rabljenog vozila između x i y cijene, umjesto konkretne x cijene.

Tri su osnovna načina primjene strojnog učenja kroz koje agent može biti treniran, a to

su Nadzirano učenje, Nenadzirano učenje i Poticano učenje. Nadzirano učenje je vrsta učenja gdje se agentu prije početka učenja daju preddefinirani označeni podaci. Kasnije kroz trening agent sam pokušava prepoznati neoznačene podatke i svrstati ih u točne kategorije. Primjer bi bio program za detekciju neželjene pošte, gdje je potrebno razlikovati strukturu i namjeru dobre od one loše. Nenadzirano učenje je obratno od nadziranog, gdje se agentu isključivo daju neoznačeni podaci. Taj način učenja je za jedan stupanj apstrakcije viši jer se ne radi o specifičnim podacima, već o grupama sličnih podataka za što se uglavnom i koristi [5]. Poticano učenje, koje se međuostalim koristilo u ovom radu za treniranje agenata, je učenje gdje se sve vrti oko nagrade. U ovoj vrsti učenja agent je često u interakciji sa okolinom te može utjecati na nju što mu, ako radi što bi trebao, donosi nagradu. Stoga je cilj povećati broj dobivenih nagradi i broj interakcija s okolinom. Postoji jedna dilema u kojoj se mogu naći agenti prilikom učenja, a to je, istraživati i otkrivati nove aspekte treninga i potencijalno biti nagrađeni za to ili iskorištavati dosad naučeno i biti sigurno nagrađeni. Zbog te dileme, kasnije u ovom radu, prikazana je važnost dobre pripreme testne okoline u kojoj se agent neće samo oslanjati na iskorištavanje naučenog znanja već se konstantno prilagođavati svojoj okolini kako bi maksimizirao šanse savladavanja krajnjeg cilja [7].

4.0.4. Višeagentni sustav

Agent kao takav je računalni sustav koji samostalno donosi određene odluke. Računala znaju samo ono što im programeri zadaju, a u trenutku kada određena radnja izlazi iz zadane domene, odnosno dogodi se ne očekivani rezultat, program kao takav prestaje s radom. Agenti baš u takvom okruženju funkcioniraju, gdje su konstantne promjenjive i nepredvidive situacije [8]. Kada agent donosi odluke u okruženju u kojem se nalazi više faktora, odnosno više drugih agenata tada se to okruženje može nazvati višeagentnim sustavom. Agent u takvom okruženju zna kako i što napraviti. Kako, se odnosi na to da zna odraditi više akcija odjednom kao što na primjer neki ljudi mogu hodati i pričati u isto vrijeme, a što, se odnosi na to da odluke može donositi ovisno ili ne ovisno o drugim agentima u sustavu [4]. Može se reći da agenti predstavljaju inteligentne sustave koji opažaju okolinu u kojoj se nalaze, potom donose odluke, na temelju njih poduzimaju set akcija i na kraju ovisno o akcijama, ako se radi o poticanom učenju, dobivaju nagradu ili kaznu za dobar ili loš potez. G. Weiss [8] navodi kako su proaktivnost, reaktivnost i društvena sposobnost, odličja koja agenta čine inteligentnim. Proaktivnost podrazumjeva sposobnost agenta da svojim ponašanjem i akcijama izvrši određeni zadatak. Reaktivnost podrazumjeva sposobnost agenta da pravovremeno reagira na promjename nastale u njegovom okruženju kako bi uspješno izvršio određeni zadatak. Društvena sposobnost odnosi se na interakciju između agenata ili između agenata i ljudi koja je ponekad potrebna za ostvarenje određenog zadatka.

Sustav koji je razvijen za potrebe ovog rada sastoji se od više agenata koji u isto vrijeme pokušavaju zajedničkim akcijama postići isti cilj te se stoga to zove višeagentni sustav. Dakle, to je sustav u kojem svaki agent ima nedovoljno informacija kako bi sam izvršio zadatak, ne postoji globalna kontrola, podaci su decentralizirani između agenata i izračuni su asinkroni [8]. Na primjeru igre može se vidjeti da zadovoljava sva četiri zahtjeva višeagentnog sustava,

svaki zombij kao jedinka sakuplja podatke sa svog stajališta o tome gdje se mora kretati i što mora napraviti kako bi došao do igrača, svaki zombij kontrolira samo sebe i svoje kretnje te se izračuni dešavaju paralelno, odnosno svaki zombij donosi asinkrono odluke o tome kako će u što kraćem vremenu skupiti što više nagrada. M. S. Mahmoud [9] navodi definiciju koja podupire svojstva višeagentnih sustava gdje su agenti unutar njih skup slabo povezanih dinamičnih jedinki. Također uspoređuje sustav agenata sa rojem (eng. *Swarm*) jer time simbolizira nešto dinamično, gdje agenti nisu u jednoj specifičnoj formaciji, već zbog svoje slabe strukturiranosti i velike brojnosti mogu postignuti niz različitih formacija i tako preduhitriti nepredvidivost određenih situacija. Prilikom razvoja i konfiguracije agenata također treba uzeti u obzir da je svaki agent jedinka, iako se radi o višeagentnom sustavu, svaki agent donosi odluke samo za sebe. Ako se taj detalj zanemari tada može doći do potpuno obrnute situacije od učenja i napretka, gdje jedni drugima smetaju i onemogućuju postizanje najoptimalnijeg puta prema cilju. Primjer toga naveden je niže u dijelu implementacije.

5. Implementacija i analiza višeagentnog sustava

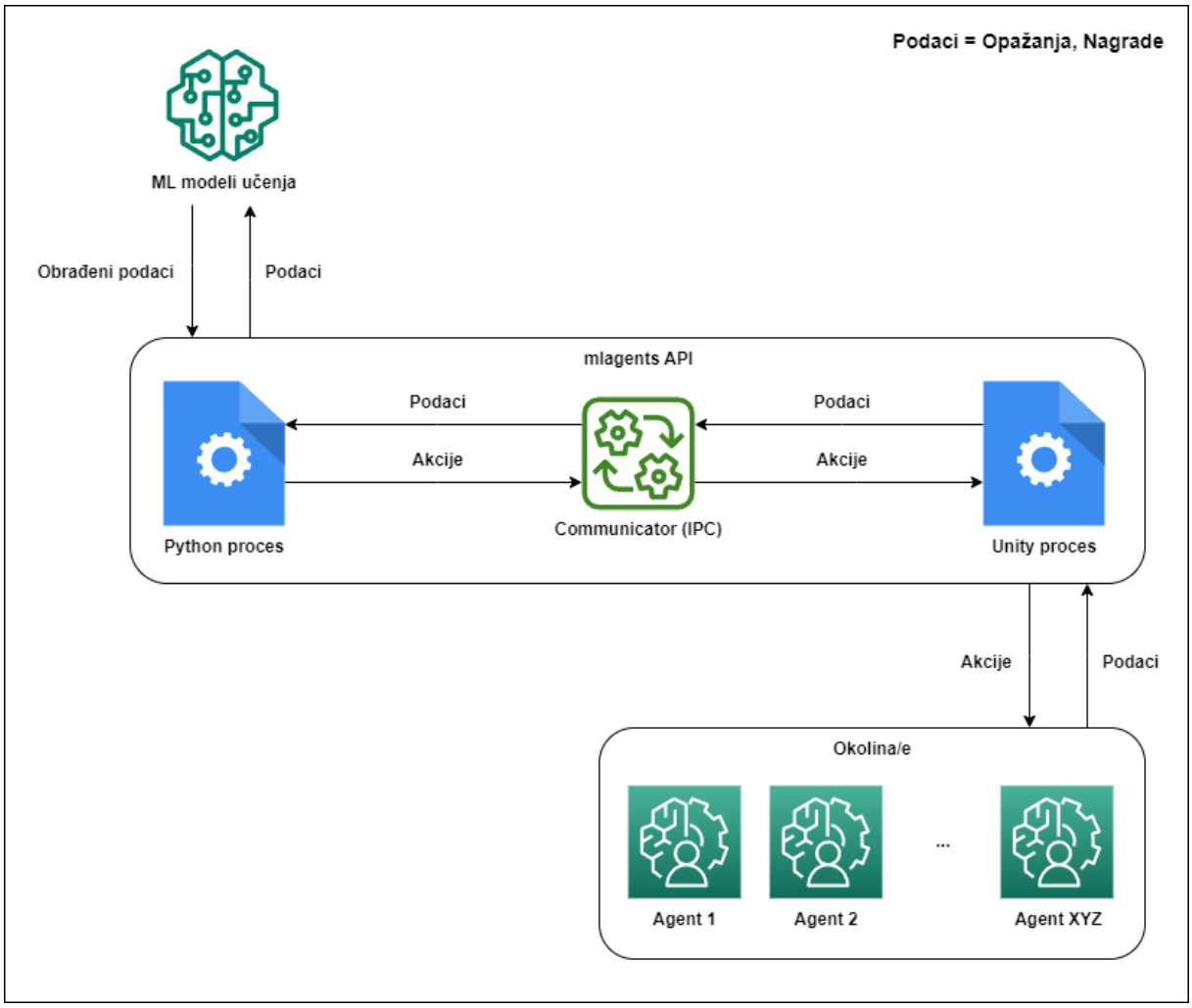
U ovom poglavlju prikazani su i objašnjeni svi aspekti implementacije višegentnog sustava u igru. Pobliže je objašnjen način na koji je cijeli sustav konfiguriran u smislu toka podataka, simulacija, regulacija učenja agenta, agentov način poimanja okoline te prikupljanje i obrada podataka. Nadalje opisan je i prikazan sam proces učenja, kao i konfiguracije agenta i njegove okoline. Također je kroz analize pojedinih faza učenja pojašnjeno kako i zašto je agent donosio određene odluke, odnosno akcije i što je bilo poduzeto u datom trenutku.

5.0.1. Konfiguracije učenja

Svaki agent prije početka učenja mora dobiti određene direktive i pravila ponašanja kako bi uspješno naučio izvršavati zadatke koji su mu namijenjeni. U ovom radu to je izvedeno pomoću platforme *Unity* koja predstavlja okruženje za pokretanje simulacija učenja i alata *ML-Agents* razvijenog specifično za upotrebu u *Unity-u* kao jedan od njegovih alata. Drugi dio alata *ML-Agents* razvijen je u jeziku *Python* kao dio okvira *PyTorch* koji je namijenjen za rad sa raznim oblicima umjetne inteligencije, ali i ujedno kao most između simulacije i obrade podataka.

5.0.1.1. Tok podataka

Cijeli sustav učenja agenata, slika 26, funkcionira na način da se podaci razmjenjuju između platforme *Unity* i Python procesa. Prvo agenti unutar predefiniranog okruženja opažaju svoju okolinu. Potom se pomoću *mlagents API-a*, koji osigurava sve aspekte komunikacije između Unity procesa i Pythone procesa, ti podaci opažanja prenose sve do modela strojnog učenja. Tamo se obrađuju i na temelju njih šalje se akcija koju agent treba izvršiti. Potom se sve to ponavlja samo ovaj put nije u pitanju opažanje već nagrada koju je agent dobio ovisno o akciji, te se ta nagrada također sprema u model strojnog učenja kako bi umjetna inteligencija na temelju budućih opažanja i akcija bila korak bliže ostvarenju krajnjeg cilja [10]. Važno je razjasniti stavku *Communicator* i njezinu ulogu u prijenosu podataka pomuću *mlagents API-a*. *Communicator* služi za međuprocesnu komunikaciju (eng. *Interprocess communication IPC*), odnosno olakšava prijenos podataka uspostavljanjem veze između već spomenutih procesa. Uz sam prijenos podataka također je odgovoran za reguliranje okruženja, odnosno njegovo resetiranje nakon završene epizode, što je detaljnije objašnjeno u nadolazećim poglavljima. Također omogućava paralelno učenje više agenata, kao i sinkroniziranje njihovih opažanja i akcija. Postoji više vrsta *Communicator-a* iz razloga jer se učenje ne mora nužno izvršavati na jednom računalnom sustavu, nego se može izvršavati paralelno na više njih.



Slika 26: Prikaz toka podataka između Unity-a i Python-a

5.0.1.2. Unity simulacije

Dio koji se direktno koristi sa *Unity-em* je alat *ML-Agents* koji se može jednostavno preuzeti putem *Package Managera* i tako integrirati u projekt. Prije samog kretanja sa izradom agenata, alat nudi više različitih, gotovih primjera okruženja sa različitim primjenama agenata i samog alata. Kroz te primjere prezentirane su sve funkcionalnosti koje nudi te se putem njih može relativno brzo pohvatati osnove koje su potrebne kako bi se shvatio princip funkcioniranja cjelokupnog sustava. Uz primjere dostupan je i službeni *Git* repozitorij [11] sa detaljnim uputama od instalacije do objašnjenja svih detalja koji se pojavljuju prilikom rada s alatom. Tehnički dio objašnjen je i prikazan pomoću izrađene igre. Objekt zombija predstavlja agenta koji sadrži sve skripte potrebne za učenje.

5.0.1.3. Skripta za reguliranje ciklusa učenja agenta

Ručno izrađena skripta *AIZombieMovement* nasljeđuje klasu *Agent* koja dolazi s alatom *MLAgents* kroz biblioteku *Unity.MLAgents* i u njoj se nalaze sve metode zadužene za provođenje simulacija te ponašanje agenta. Glavne metode koje su korištene za provođenje

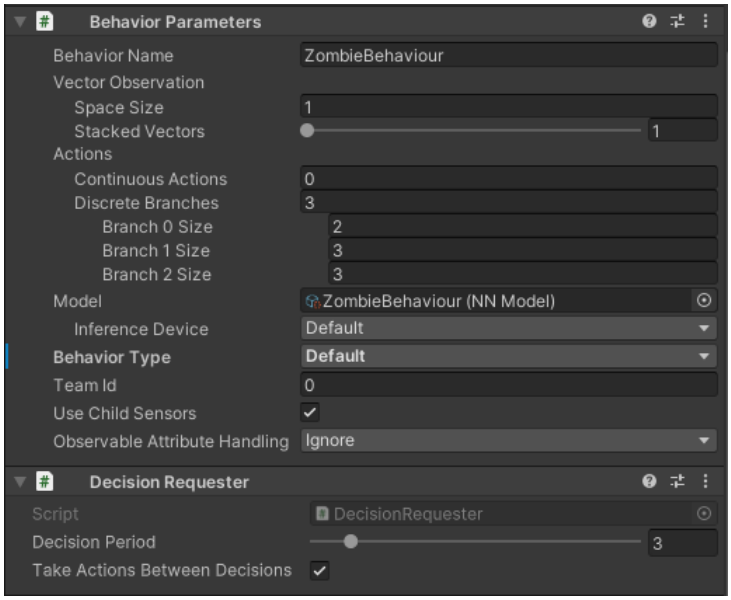
simulacija su *OnEpisodeBegin() i EndEpisode()*. One reguliraju tok, odnosno epizode učenja, gdje metoda *OnEpisodeBegin()* iznova pokreće epizodu i ponovo postavlja okruženje na inicijalne postavke, dok metoda *EndEpisode()* završava aktualnu epizodu. Epizoda je zapravo određena količina koraka odnosno akcija koje agent može napraviti prije vračanja okruženja na inicijalne postavke i ponovnog pokušaja učenja. Ta metoda se najčešće koristi u dvije situacije, kada je agent uspješno izvršio zadatak i kada agent napravi akciju koja je zabranjena ili jednostavno smatrana nevažnom za ostvarenje cilja. Također važno je spomenuti atribut *Max Steps*, koji se može uređivati preko *Unity* radnog sučelja, a određuje koliko akcija, odnosno koraka agent može napraviti prije ponovnog pokretanja epizode.

Metode koje je važno spomenuti, a vezane su uz samog agenta, njegovo učenje i korištene su prilikom izrade igre su *CollectObservations(VectorSensor sensor), OnActionReceived(ActionBuffers actions) i Heuristic(in ActionBuffers actionsOut)*. Metoda *CollectObservations(VectorSensor sensor)* kao ulazni parametar prima tip *VectorSensor* koji kao važnu metodu posjeduje *AddObservation()* koja prima razne tipove podataka koje potom šalje na daljnju obradu modelima strojnog učenja, dakle služi sa prikupljanje opažanja, koja se proizvoljno definiraju. Nadalje metoda *OnActionReceived(ActionBuffers actions)* kao parametar prima tip *ActionBuffers* koji se odnosi na akcije koje agent može napraviti i koje se također definiraju proizvoljno prema prije određenim konfiguracijama u skripti *BehaviorParameters* koja je objašnjena nešto kasnije. I kao zadnja bila bi metoda *Heuristic(in ActionBuffers actionsOut)* koja služi za ručno davanje komandi agentu i čija je svrha, prvenstveno, testiranje postavljenih konfiguracija prije početka samog treninga, odnosno učenja.

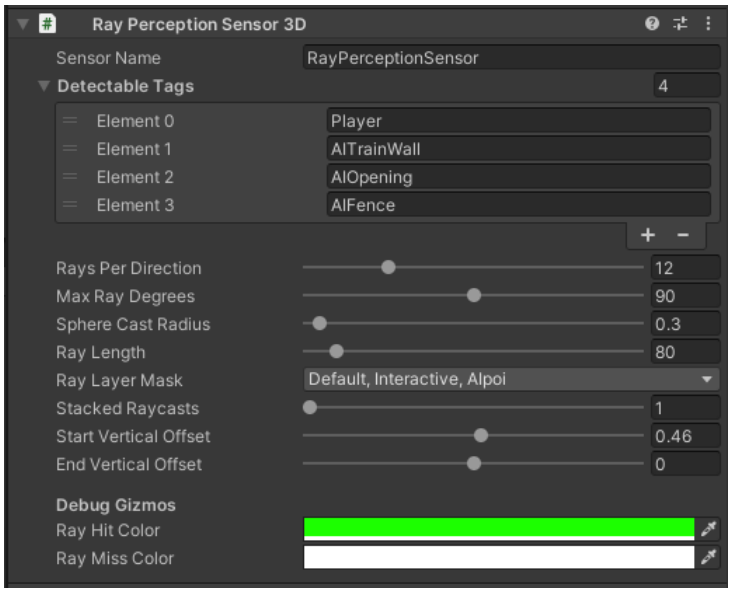
5.0.1.4. Skripte postavki simulacije ponašanja i opažanja agenta

Uz mnoge funkcionalnosti koje alat *ML-Agents* nudi korištena je jedna osnovna skripta *BehaviorParameters* kao i dvije dodatne skripte pod nazivima *DecisionRequester i RayPerceptionSensor3D*, slike 27 i 28. Kada se na objektu definira skripta koja nasljeđuje klasu *Agent* tada se na njemu također automatski generira navedena osnovna skripta. Skripta *BehaviorParameters* sadrži sve postavke koje su važne za agentovo opažanje i akcije. Atributi koje je važno istaknuti su *BehaviorName, VectorObservation, Actions, Model i BehaviourType*. *BehaviorName* odnosi se na ime ponašanja, odnosno učenja koje se želi postići, ali isto tako kako bi se agenta ili skup agenata koji dijele istu funkcionalnost moglo identificirati. *VectorObservation* govori o tome koliko će podataka agent opažati iz okoline. Točnije to se definira pod atributom *SpaceSize*. Na primjer, ako se prati pozicija određenog objekta tada se *SapceSize* mora postaviti na 3 jer se prate x, y i z koordinate. *Actions* atribut dijeli se na *ContinuousActions i DiscreteBranches* i govori o tome koliko akcija odnosno ulaznih komandi agent može podržavati po pitanju kretanja u okruženju simulacije. *ContinuousActions* odnosi se na decimalne brojeve između -1.0 i 1.0, što znači da ako se zada 2 tada će agent kao akcije primati dvije varijable, odnosno listu od dva elementa tipa *float* npr. [0.546123, -0.123987], što se tada može koristiti po potrebi za što je namijenjeno. *DiscreteBranches* odnosi se na cijele brojeve i to na način ako se zada 3 tada će se stvoriti lista s tri elementa tipa *integer* gdje svaki može biti maksimalno minus 1 od onoga što je navedeno pod atributem *Branch X Size*. Na primjer ako se navede *Branch 0 Size* 3, tada agent raspolaže za bojevima 0, 1 i 2 za određenu akciju za

koju se taj ulazni podatak odredi. *Model* se odnosi na model neuronske mreže te ako se trenira novi agent taj atribut ostaje prazan jer se neuronska mreža tek treba kreirati pokretanjem simulacije učenja. U slučaju da je učenje odrađeno tada se može taj atribut postaviti na neuronsku mrežu treniranog *.onnx* modela. Postavi li se *BehaviourType* na *Inference Only*, tada će agent donositi odluke na temelju sakupljenih podataka kroz prethodne treninge. I na posljetku atribut *BehaviourType*, kao što je već spomenuto, određuje na temelju čega će agent donositi odluke, bilo to na temelju komandi igrača ili na temelju modela neuronske mreže. Ako se tokom simulacije učenja atribut *Model* ostavi prazan, što je i poželjno te *BehaviourType* ostavi na *Default* tada će agent učiti na temelju svojih vlastitih opažanja i akcija. Uloge dvije dodatne skripte *DecisionRequester i RayPerceptionSensor3D* su zahtjevanje odluke nakon opažanja, na temelju čega se donosi akcija te kreiranje dodatnih senzora koji pomažu agentu prilikom opažanja okoline [10]. Detaljniji opis skripte *RayPerceptionSensor3D* opisan je niže gdje su pojedini atributi objašnjeni s više tehničke strane.



Slika 27: Skripte postavki simulacije učenja agenta *BehaviorParameters* i *DecisionRequester*



Slika 28: Skripte postavki simulacije učenja agenta *RayPerceptionSensor3D*

5.0.1.5. Podatkovni Python dio

Dio koji se indirektno koristi sa *Unity-em* je Python virtualno okruženje unutar kojeg se izvršavaju komande za početak i kraj treninga, odnosno učenja agenta. Nakon što je sve instalirano prema uputstvima koje se mogu pronaći na službenom *Git* repozitoriju od *Unity-a* [11] tada je prvo potrebno konfigurirati postavke vezane uz treninge, ali i modele strojnog učenja. Ako se konfiguracija ne promijeni tada će se učenja unutar simulacija i dalje provoditi ali postoji mogućnost da krajnji rezultat ne bude na razini očekivanog ili će se prikupljati nerelevantni podaci. Konfiguracija koja je korištena za trening agenata koji su implementirani u ovom radu prikazana je i objašnjena ispod.

```
behaviors:
  ZombieBehaviour:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 3.0e-4
      beta: 5.0e-3
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
```

```
    strength: 1.0
max_steps: 1000000
time_horizon: 64
summary_freq: 20000
```

Kod 5.1: ML-Agents konfiguracijska datoteka

Konfiguracijska datoteka započinje imenom ponašanja, odnosno učenja koje isto tako mora biti specificirano unutar *Unity-a* u skripti *BehaviorParameters* koja je ranije spomenuta, ovdje je to *ZombieBehaviour*. Nadalje neke od važnijih postavki koje su posebno namještene za potrebe treniranja zombija su *trainer_type, batch_size, learning_rate_schedule, normalize, hidden_units, reward_signals, max_steps i summary_freq*. *Trainer_type* naznačuje koji će se algoritam koristiti za učenje. Mogu biti PPO (eng. *Proximal Policy Optimization*), SAC (eng. *Soft Actor-Critic*) i POCa (eng. Policy Optimization with Continuous Actions). Za potrebe ovog rada korišten je PPO zbog njegove karakteristike da optimizira učenje, odnosno osigurava stabilne promjene učenja kako one ne bi bile prevelike i tako potencijalno negativno utjecale na učenje. Ostale dvije su SAC koji osigurava učenje putem otkrivanja novih informacija kako bi postigao što optimiziranije učenje i POCa koji se sličan SAC-u no primjenjuje se na scenarije koji mogu imati više od jednog ishoda učenja kao krajnji cilj te ih tako pokušava optimizirati. *Batch_size* se odnosi na broj iskustava (opažanje/nagrada) koje agent ima u jednoj iteraciji, koji je u ovom radu postavljen na 1024 i 10 puta je manji od *buffer_size* koji iznosi 10240 i predstavlja broj iskustava koje se skupljaju prije ažuriranja modela strojnog učenja. *Learning_rate_schedule* određuje kako se učenje mijenja tokom vremena i pošto se koristi PPO tada je taj atribut postavljen na *linear*. To znači da kada agent nije blizu ostvarivanju željenog zadatka, stopa učenja će biti veća, a kada dođe sve bliže ostvarenju, stopa učenja će se smanjiti kako bi se model strojnog učenja mogao izgladiti. *Normalize* služi kako bi kompleksniji podaci bili što konzistentnije prikazani, odnosno ako se radi sa cijelim brojevima, kao što su 0, 1, 2 tada to nije potrebno i zato je u ovom primjeru to postavljeno na false. *Hidden_units* predstavljaju broj zasebnih neurona ili čvorišta neuronske mreže te može imati vrijednost od 32 do 512. Za potrebe ovog rada, pošto se radi sa jednostavnijim vrijednostima tada je 128 neurona i više nego dovoljno za sve izračune koje agent zombi treba odraditi. *Reward_signals* je dio gdje se definiraju eksterni i/ili interni signali za nagrade. Ako se radi samo o eksternim signalima, tada će agent, kao što je i u ovom radu, učiti samo iz okoline u kojoj se nalazi, ako su samo definirani interni signali, tada će agent samo učiti iz predefiniranog seta podataka koji je unaprijed snimljen od strane stvarnog čovjeka i tako pokušati naučiti što bolje izvršavati svoj cilj. Isto tako mogu biti oba definirana gdje se onda gleda atribut *strength* koji određuje u kojem postotku će se svaki od njih koristiti prilikom učenja agenta. *Max_steps* označuje maksimalan broj koraka ili akcija nakon kojeg se trening završava, a *summary_freq* označuje broj nakon koliko će se iskustava agenta generirati i prikazati statistika učenja [11]. Statistika se prikazuje lokalno preko web aplikacije (TensorBoard), a objašnjena je kasnije prilikom analize učenja agenta.

5.0.2. Učenje agenata u virtualnom okruženju

Početna faza učenja agenta je njihova priprema kao i priprema okoline u kojoj će biti trenirani. Nakon toga slijedi provođenje učenja, odnosno treninga, gdje agenti kroz niz koraka

pokušavaju svojim istraživanjem okoline i raznim izračunima koji se paralelno izvode u modelu strojnog učenja, njihovom mozgu, odraditi svaki zadatak koji je stavljen pred njih. Svaka faza u ovom poglavlju objašnjena je i prikazana s tehničke strane, odnosno iz perspektive tehničke izrade igre ovog diplomskog rada.

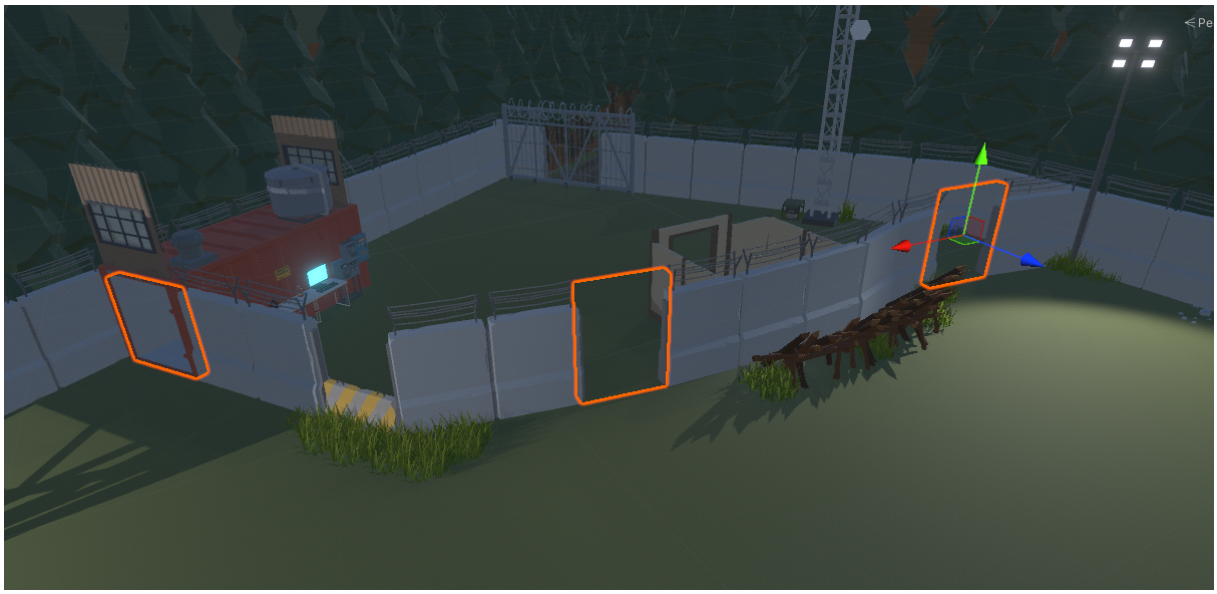
5.0.2.1. Priprema agenata i okruženja

Prilikom izrade agenata i okruženja cilj je bio jasno odvojiti ono što je agentu važno, s čime može biti u interakciji i pomoću kojih akcija će doći do pozitivnih nagrada, od onoga što ne bi smio ili nije predviđeno da radi. Prema tome je prvenstveno osmišljen tok igre, odnosno tok kojim bi igrač trebao pobijediti ili izgubiti. Dakle, krajnji cilj agenta u kojem igrač gubi, a agent postiže najoptimalniju nagradu je pronalazak i udaranje igrača sve dok ne izgubi sve životne bodove.

Dio mape, nazvan *Vojna baza*, slika 29, razvijen je na način da može poslužiti i kao okruženje za učenje agenata, ali i kao sastavni dio cijele igre, odnosno levela. Kroz niz agentovih zadataka objašnjeni su pojedini dijelovi mape. Prvo je trebalo definirati pojedinosti iz kojih bi agent mogao naučiti kako da izađe iz mjesta gdje je instanciran (eng. *Spawn-a*), slika 30. Oko tog mjesta, sa tri strane, postavljena su tri nevidljiva zida sa oznakom *AITrainWall*, a na četvrtoj strani nalaze se objekti zida iste oznake, objekt ograde sa oznakom *AIFence* i nevidljivi objekt sa oznakom *AIOpening* koji predstavlja otvor za prolaz agenta, slika 31. Drugo, trebalo je definirati dio gdje agent pronalazi svoj put do igrača zaobilazeći prepreke. Na tom dijelu, odnosno okruženju, svi objekti koji bi se trebali izbjegavati, kao što su auti, kontejneri, kutije i slično, označeni su istom oznakom *AITrainWall*, a igrač do kojeg agent ultimativno mora doći označen je posebnom oznakom *Player*. Također uz sve to, ogradi, odnosno objektu roditelj, koji za svoju djecu ima ogradu i prolaz, dodana je skripta *AIFence* koja regulira uništavanja ograde, odnosno aktiviranje objekta prolaza i deaktiviranje objekta ograde, što je objašnjeno detaljnije u sljedećem podpoglavlju *Proces učenja*.

Slika 29: Dio mape namijenjen učenju agenata

Slika 30: Mjesto instanciranja agenta zombija, ograda i zidovi



Slika 31: Objekti prolaza koje agent zna prepoznati

Agent koji svoj proces učenja odrađuje u prethodno opisanom okruženju može imati interakciju sa ogradom, prolazima, zidovima, preprekama te sa igračem. Kako bi to bilo izvedivo korištene su *Unity* metode *OnCollisionEnter(Collision collision), OnCollisionStay(Collision collision) i OnTriggerExit(Collider other)*. Te metode omogućuju detekciju "sudara" dva, ranije spomenuta *Collidera*, koji ujedno mogu biti i okidači (eng. *Trigger*) što im omogućuje da drugi objekti sa *Colliderim-a* na sebi prolaze kroz njih. Uz sudare može se točno pratiti koji objekt je ušao, ostao ili izašao iz sudara, dok se isto može pratiti i kod okidača, sa jedinom razlikom, a to je da se ne može pratiti objekt koji se sudario, nego objekt koji je prošao kroz ili ostao u njemu. Ta mehanika sudara i okidača reprezentira interakcije agenta s okolinom koja se sprema i obrađuje u modelu strojnog učenja. Pošto agent mora znati kako odraditi niz, ranije spomenutih, zadataka i to odjednom, odabrano je kurikularno učenje, gdje se kroz niz iteracija, odnosno učenja, agenta uči jedan po jedan zadatak. Učenje traje onoliko dugo koliko je potrebno da agent u jednoj iteraciji uspješno odradi sve zadatke u što kraćem vremenskom periodu uz što veću nagradu.

5.0.2.2. Proces učenja

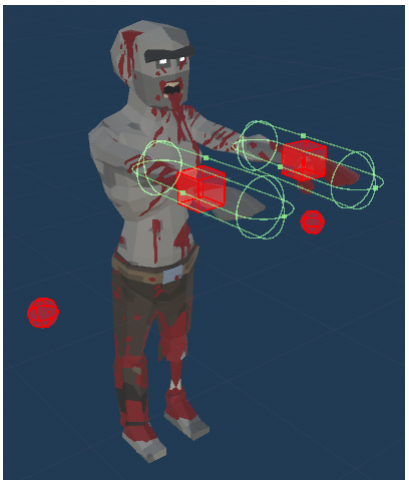
Kurikularno učenje ostvareno je kombinacijom većine funkcionalnosti koje *ML-Agents* alat nudi. U skripti *AIZombieMovement* definirane su sve kretnje, opažanja i moguće interakcije pomoću kojih agent može učiti. Kretnje agenta odrađene su u metodi *MoveAgent(ActionSegment<int> discreteAction)*, koja kao parametar prima podatak tipa *ActionSegment<int>* koji predstavlja jednu akciju. Sa slike 27 se vidi da su *Branch 1 i Branch 2* određeni za akcije kretanja. *Branch 1* predstavlja akcije mirovanja i kretanja napred i nazad, dok *Branch 2* predstavlja akcije mirovanja i kretanja lijevo i desno. Nakon pohranjivanja tih akcija u varijable *forwardBack i leftRight* tipa *int*, tada se kretanje agenta realizira pomoću dvije *Switch* petlje u kojima se određuje smjer kretanja i kasnije *Unity* metodom *Translate* objekta pomiče u željenom pravcu.

```
private void MoveAgent(ActionSegment<int> discreteAction)
```

```
{
    Vector3 direction = Vector3.zero;
    int forwardBack = discreteAction[1];
    int leftRight = discreteAction[2];

    switch (forwardBack)
    {
        case 1:
            direction = Vector3.forward;
            break;
        case 2:
            direction = Vector3.back;
            break;
        default:
            break;
    }

    switch (leftRight)
    {
        case 1:
            direction = Vector3.left;
            break;
        case 2:
            direction = Vector3.right;
            break;
        default:
            break;
    }

    transform.Translate(direction * Time.deltaTime * randMovementSpeed);
}
```

Kod 5.2: Tehnička izvedba akcije kretanja

Objašnjena metoda *MoveAgent()* poziva se u metodi *OnActionReceived(ActionBuffers actions)* iz *Unity.MLAgents* biblioteke. U toj metodi uz samo odrađivanje kretanja agenta, odrađuje se i akcija udarca, bilo ograde ili igrača. To se izvodi na način da se prvo provjeri akcija na indeksu 0 atributa *Branch 0*, slika 27, gdje može doći broj 0 ili 1. Ako model strojnog učenja odluči poslati broj 1 tada će agent izvesti udarac. Interakcija agenta i objekta koji je udaren (ograda ili igrač) izvedena je pomoću *collidera* koji djeluju kao okidači koji se nalaze na lijevoj i desnoj komponenti ruke agenta zombija, slika 32.



Slika 32: Izvedba collidera na rukama agenta

Način registracije udraca i prepoznavanja udarenog objekta izvedeno je pomoću, ranije spomenutih, *Unity* evenata u skripti *AIZombieHand* koja je primijenjena na obje ruke. Ovisno o tome je li okidač ušao u koliziju sa objektom oznake *Player* ili *AIFence* poziva se tome shodna metoda eventa unutar skripte *AIZombieMovement*.

```
public event EventHandler OnZombieHandHitPlayer;
public event EventHandler OnZombieHandHitFence;

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Player"))
    {
        OnZombieHandHitPlayer?.Invoke(this, EventArgs.Empty);
    }

    if (other.gameObject.CompareTag("AIFence"))
    {
        OnZombieHandHitFence?.Invoke(this, EventArgs.Empty);
    }
}
```

Kod 5.3: Tehnička izvedba registriranja udarca agenta

Akcija udarca dodatno je regulirana *bool* varijablom *isClose*, koja se unutar *Unity Update()* metode postavlja na *true* ako je agent blizu ograde ili igrača, te se uz sve ostalo agentu dodjeljuje nagrada u iznosu jedan. Kada je agent izveo akciju udarca, ovisno o objektu koji je udario, dodjeljuju mu se dodatni Bodovi. Ako je udario ogradu dodjeljuje mu se nagrada u iznosu 1 te ako je udario igrača dodjeljuje mu se nagrada u iznosu 3. Nagrade se tokom koraka, odnosno akcija koje je agent odlučio napraviti, sumiraju.

```
public override void OnActionReceived(ActionBuffers actions)
{
    bool isAttacking = actions.DiscreteActions[0] == 1;

    if (isAttacking)
    {
```

```
        if (isClose)
        {
            OnCloseToPlayer.Invoke(this, EventArgs.Empty);
            animator.SetBool("IsClose", true);
            AddReward(1f);
        }
    }

    MoveAgent(actions.DiscreteActions);
}
```

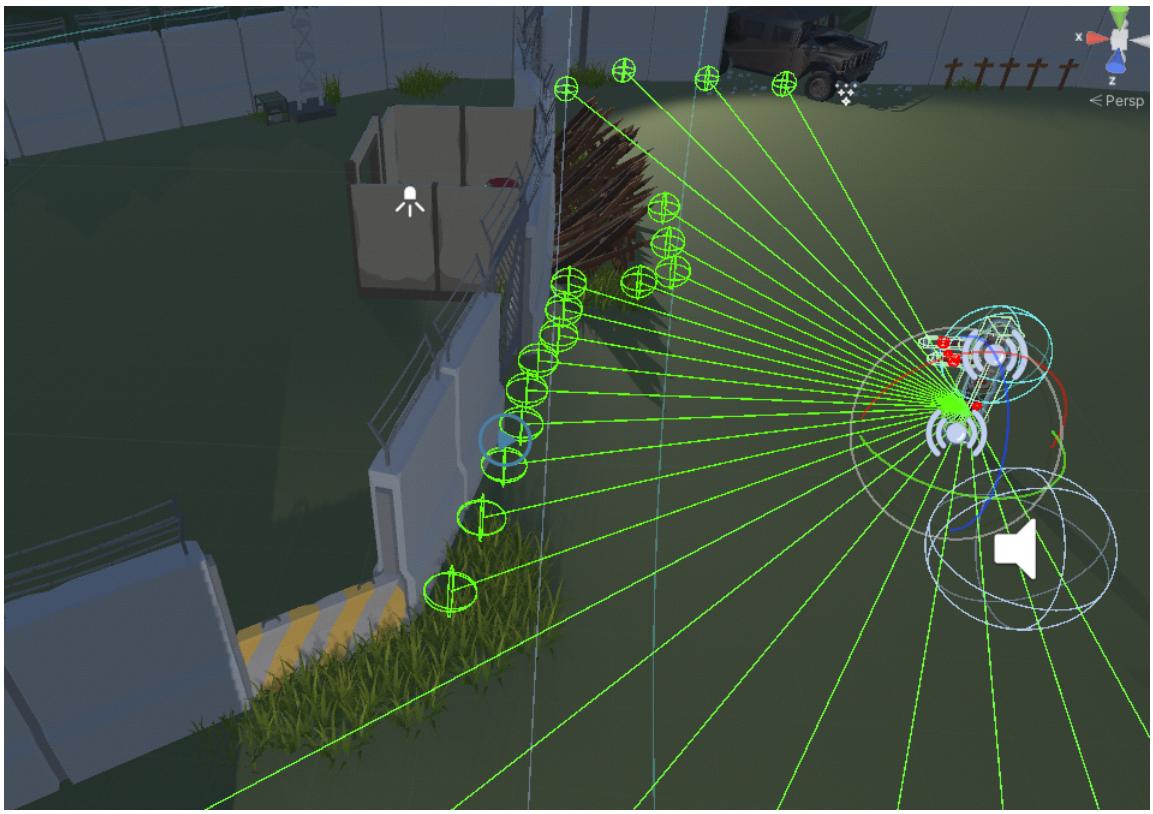
Kod 5.4: Tehnička izvedba akcije udaranja 1

```
private AIZombieHand aiZombieLeftHand;
private AIZombieHand aiZombieRightHand;

private void Awake()
{
    aiZombieLeftHand = leftHand.gameObject.GetComponent<AIZombieHand>();
    aiZombieRightHand = rightHand.gameObject.GetComponent<AIZombieHand>();
    aiZombieLeftHand.OnZombieHandHitPlayer += AiZombieLeftHand_OnZombieHandHitPlayer
        ;
    aiZombieLeftHand.OnZombieHandHitFence += AiZombieLeftHand_OnZombieHandHitFence;
    aiZombieRightHand.OnZombieHandHitPlayer +=
        AiZombieRightHand_OnZombieHandHitPlayer;
    aiZombieRightHand.OnZombieHandHitFence += AiZombieRightHand_OnZombieHandHitFence
        ;

}

private void AiZombieRightHand_OnZombieHandHitFence(object sender, EventArgs e)
{
    AddReward(1f);
}

private void AiZombieLeftHand_OnZombieHandHitFence(object sender, EventArgs e)
{
    AddReward(1f);
}

private void AiZombieRightHand_OnZombieHandHitPlayer(object sender, EventArgs e)
{
    AddReward(3f);
}

private void AiZombieLeftHand_OnZombieHandHitPlayer(object sender, EventArgs e)
{
    AddReward(3f);
}
```

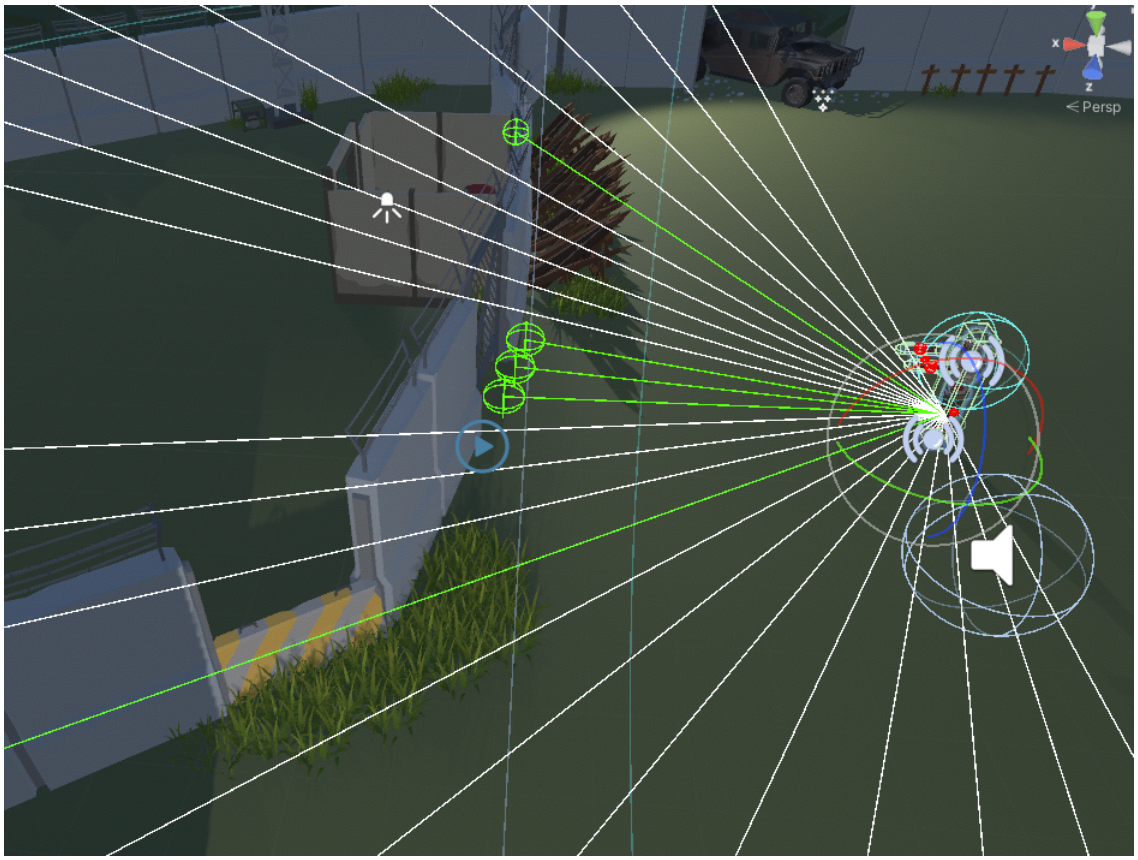
Kod 5.5: Tehnička izvedba akcije udaranja 2

Zadnja funkcionalnost koja je dodana procesu učenja je *RayPerceptionSensor3D*, slika 28. Ona omogućuje agentu prepoznavanje objekata prema njihovoj oznaci i to pomoću zraka

koje se iscrtavaju u željenim smjerovima. Postavke koje nudi su izrada liste poznatih oznaka objekata u okruženju, uređivanje izgleda detekcijskih zraka te definiranje slojeva koje će zrake detektirati. Za poznate oznake određene su *Player, AITrainWall, AIOPening i AIFence*. Što se tiče izgleda zraka, ideja je bila napraviti dovoljan broj zraka, koje nisu prevelike i predugačke kako bi što točnije detektirale okolinu agenta. Najvažnija postavka bila bi mogućnost određivanja slojeva na kojima agent, odnosno zrake mogu detektirati objekte. Za to su određeni slojevi *Default, Interactive i AIpoi*. U sloju *Default* nalaze se svi objekti unutar *Unity* scene koji nisu toliko važan segment igre, u sloju *Interactive* nalaze se svi objekti s kojima igrač može imati interakciju, kao i igrač sam, a u sloju *AIpoi* (skraćeno od eng. *Artificial Intelligence Point Of Interest*) nalaze se svi objekti kao što su ograde, prolazi i prepreke. Sa svim slojevima postavljenim, opažanje agenta sa senzorima izgleda kao na slici 33 gdje se vidi točna detekcija svih određenih objekata u okruženju, a kada je na primjer ostavljen samo sloj *AIpoi* tada se sa slike 34 može vidjeti kako se detekcija prepreka, odnosno zidova u ovom slučaju, ignorira. Također postoji i sloj *Zombie* u kojem se nalaze svi agenti iz razloga kako ne bi jedni drugima smetali prilikom učenja, odnosno kako se ne bi detektirali.



Slika 33: Prikaz detekcije objekata zrakama senzora



Slika 34: Prikaz djelomične detekcije objekata zrakama senzora

Simulacije učenja odrađene su prema spomenutom kurikularnom učenju. Svaka simulacija iznosila je milijun koraka, odnosno akcija i od toga se svakih dvadeset tisuća bilježilo za statistiku. Početak svakog učenja, osim prvog, nadovezivao se na prethodno i tako je agent postepeno nadograđivao svoje znanje putem pojedinačnih zadataka, sve do kada ih nije znao odraditi sve odjednom. Agent (točnije njih 21) je početkom svake epizode učenja ponovo instanciran na svojem specifično određenom mjestu. Pokretanjem prve simulacije agenta se suočilo sa zadatkom dolaska do ograde i njezino udaranje, gdje se prilikom udarca, za tog agenta, pokretala nova epizoda. U simulaciji broj dva agent je znao uspješno uništiti ogradu, te je bio cilj naučiti kako doći do prolaza. U simulaciji broj 3 objekt igrača bio je namješten odmah nakon prolaza, te se agenta ovdje učilo kako da izađe iz prolaza, dođe do igrača i udari ga. Od 4. do 6. simulacije ponovljene su radnje iz simulacija 1, 2 i 3 kako bi agent još dodatno uvježbao izvršavanje tih osnovnih zadataka. Od 7. do 26. simulacije agenta se učilo kako da pronađe i udari objekt igrača koji se nakon određenog broja udaraca instancirao na drugoj nasumičnoj lokaciji mape, ali zasada još uvijek van baze (naziv dijela gdje igrač može aktivirati proces izlječenja zombija). U simulaciji broj 27 cilj učenja agenta bio je doći što bliže bazi što je bilo potaknuto dinamičnim instanciranjem objekta igrača ispred ulaza od baze. Što znači da je objekt igrača svaki puta nakon određenog broja udaraca nasumično stvoren na jednom od tri ulaza u bazu, dvije ograde i otvor kontejnera za izlječivanje. Od 28. do 31. simulacije cilj učenja agenta bio je dolazak do ograde od baze i udariti ju, čime bi se epizoda završila i agent bi krenu ponovo s istom radnjom. Objekt igrača je u ovom slučaju bio na jednom statičnom mjestu unutar baze. Nakon toga uočeno je da agentovo zaobilaženje prepreki nije na zadovoljavajućoj

razini te da je potrebno provesti dodatna učenja tog zadatka. Povodom toga, od 32. do 40. simulacije objekt igrača je kroz svaku od simulacija bio na drugoj statičnoj poziciji, što znači da nije mijenjao svoju poziciju nakon što bi ga agenti izudarali. Time je bio cilj agenta dodatno upoznati sa svim dijelovima mape van baze. Od 41. do 51. simulacije također je primijenjena metoda statičnog objekta igrača samo ovaj put unutar baze, gdje je cilj bio naučiti agenta o vrlo preciznom dolasku do i ulasku u bazu. Od 52. do 70. simulacije dodatno su razrađeni pristup učenja agenta i princip interakcije agenta sa objektom igrača. Do 52. simulacije u sva su se tri područja, slika 35, paralelno instancirali svi agenti i objekt igrača bi promijenio svoju poziciju samo nakon određenog broja udaraca. To je uzrokovalo da na pojedinim preprekama agenti jednostavno zapnu te do kraja simulacije nikada ili vrlo rijetko dođu do objekta igrača i udare ga, što je impliciralo ponavljanje jednog ili više setova simulacija jer su podaci modela strojnog učenja postali nedostatni. Kako bi se učenje optimiziralo, od 52. simulacije, uključujući i nju, agenti su instancirani samo u jednom području po jednom učenju, a objekt igrača se neminovno o udarcima agenata instancirao na drugoj poziciji nakon određenog vremenskog perioda. Pozitivni efekti realizacije tih promjena bili su preciznije učenje agenta, kao i njegovo dodatno učenje zaobilaženja prepreki. Dakle, od 52. do 70. simulacije agenta se učilo po novom principu i uzorku instanciranja koje je bilo donje, srednje pa gornje područje i tako u krug. Nakon 70 milijuna akcija, odnosno koraka, agent je uspio postići vrlo optimalan model strojnog učenja koji je potom uspješno implementiran u igru.

Slika 35: Prikaz područja instanciranja agenata

5.0.3. Analize učenja

Proces učenja pokazao je kako je svaki detalj agenta i njegove okoline bitan. Na samom početku učenja cilj je bio naučiti agenta njegovom zadatku, a kroz sam proces učenja pojavili su se detalji, kako naučiti, kada naučiti, s čim naučiti i tako niz drugih pitanja, čiji su odgovori zapravo i doveli do finalnog rezultata. Jedan od tih detalja je agentovo prioritiziranje objekata. Pod time se misli na slučaj kada su objekti ograde, prolaza i igrača ispred agenta na relativno jednakoj udaljenosti. Ispostavilo se da je prioritiziranje, u smislu do čega će agent prvo otići, usko vezano sa nagradama koje je dobivao. Dakle, kada se povežu nagrade i razlog njihovog ostvarivanja jasno se vidi kojim redosljedom je model strojnog učenja naučio odlučivati, prvo objekt igrača (*nagrada = 3*), drugo objekt prolaza (*nagrada = 2*) i treće objekt ograde (*nagrada = 1*). Ako vidi sva tri objekta ići će prema igraču, ako nema igrača, ići će prema otvoru, a ako nema ni jedan drugi izbor ići će prema ogradi.

Prepreke su sastavni dio svakog procesa učenja pa tako i ovog. Glavni izazovi koji su trebali biti savladani, u ovom radu, su agentovo iskorištavanje naučenog znanja, uočavanje nelogičnosti konfiguracije agenta na vrijeme te uočavanje napretka agenta koji je ponekada bio jako malen, gotovo neprimjetan te je bilo potrebno procijeniti kada nastaviti učenje, a kada prekinuti i napraviti određene izmjene. Iskorištavanje naučenog znanja bio je velik izazov koji se manifestirao tako što je agent došao do te razine da bi kontinuirano samo ulazio i izlazio kroz prolaz bez namjere pronalaska objekta igrača, odnosno daljnjeg napredovanja. Iz tog razloga kreirana je dodatna metoda *DelayOpeningExitReward()* u skripti *AIZombieMovement*, koja je regulirala vremenski razmak u kojem je agent nakon prolaska kroz otvor, umjesto ostvarivanja jedne pozitivne nagrade, ostvario jednu negativnu nagradu, što je agentu ukazalo da to nije najoptimalniji ishod. Uočavanje nelogičnosti je također važan aspekt analiziranja rada agenta. Ako se ne isprave na vrijeme, jedini način njihovog ispravka, uz promjenu konfiguracije, je ponavljanje svih onih učenja koja su bila zahvačena tom nelogičnosti. Primjer iz ovog rada je konfiguracija skripte *RayPerceptionSensor3D* u kojoj je pod postavkom *RayLayerMask* bio naveden i sloj agenta *Zombie*. To je prouzročilo da agenti opažaju jedni druge što je negativno utjecalo na učenje. Prvi agent koji bi naučio izaći iz područja instanciranja, navigirati svoj put po mapi do igrača i udariti ga, svi ostali iza njega bi samo slijedili njegovo ponašanje i učili iz njegovog opažanja, odnosno opažajući njega. Većina agenata je u toj situaciji učila da je najoptimalnije slijediti druge agente, a ne pronaći igrača i udariti ga, što je dovodilo da se agenti besciljno vrte u krug, jedni iza drugih. Promjenom te postavke *RayLayerMask*, micanjem sloja *Zombie*, agenti su ponovo postali autonomni.

```
private bool isOpeningExit = false;

private void OnTriggerExit(Collider other)
{
    if (other.CompareTag("AIOpening"))
    {
        if (isOpeningExit)
        {
            AddReward(-1f);
        } else
        {
```

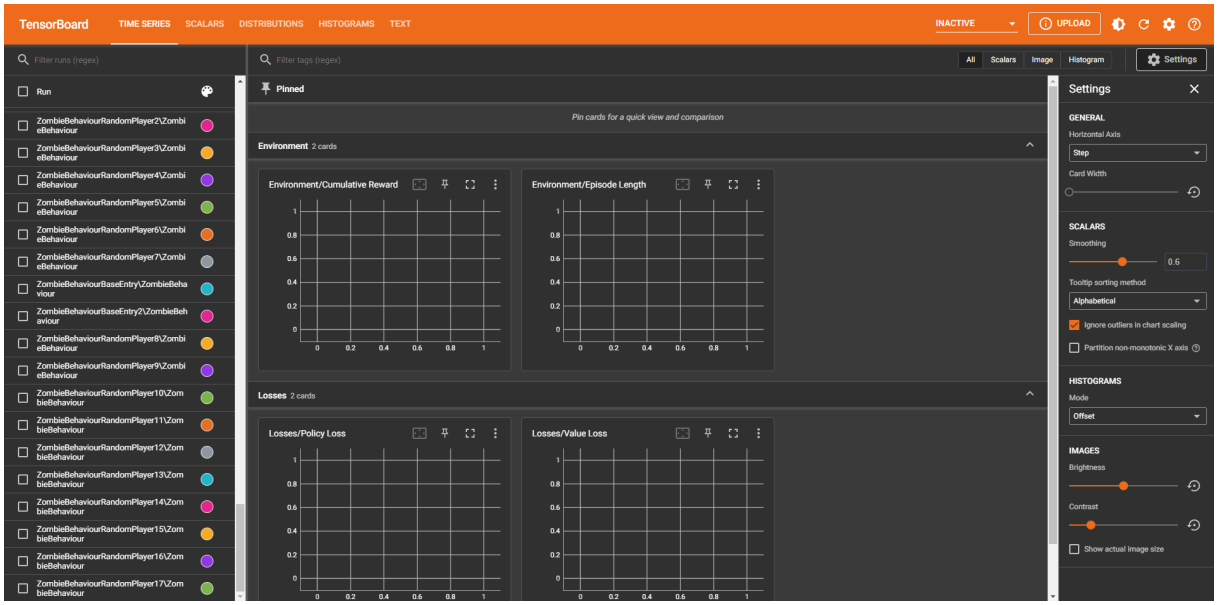
```
            AddReward(2f);
        }
        StartCoroutine(DelayOpeningExitReward());
    }
}

private IEnumerator DelayOpeningExitReward()
{
    isOpeningExit = true;
    yield return new WaitForSeconds(2f);
    isOpeningExit = false;
}
```

Kod 5.6: Reguliranje dodjele nagrade za prolaz kroz otvor

Nakon završetka svih planiranih učenja i implementacije agenta u samu igru, znanja agenta stavljena su na još jedan dodatan test. Dio mape nazvan *Groblje*, nije postavljen kao okruženje koje bi agent u potpunosti znao prepoznati, nema otvora s prolazima, nema ograda niti posebnih oznaka za prepreke. Usprkos tome, već s prvim pokušajem, bilo je jasno da se agent može uz manje poteškoće kretati po novom, neadaptiranom okruženju i da je dovoljan objekt igrača kao izvor podataka na temelju kojih može donijeti vrlo optimizirane odluke i akcije. Jedina poteškoća s kojom se agent susreo je povremeno zapinjanje na preprekama, ali što je u toj situaciji bilo sasvim prihvatljivo i razumljivo ponašanje.

Praćenje procesa učenja je također važan postupak kojim se mogu uočiti ranije spomenute nelogičnosti i usporeni napredak. U slučaju nelogičnosti, kontinuirano praćenje i analiziranje nudi pravovremeno reagiranje na njih, dok u drugom slučaju može spriječiti prijevremeni prekid učenja i nepotrebne dorade. Ono najvažnije je da se vrijeme optimizacije modela strojnog učenja uvelike može smanjiti i olakšati. Alat *ML-Agents* to nudi na dva načina, praćenje podataka o učenju putem terminala i praćenje vizualnog prikaza podataka putem *TensorBoard-a*, s time da su oba jednako pouzdana. Podaci o učenju agenta putem terminala mogu se pratiti odmah nakon pokretanja simulacije, gdje se svakih određenih broj koraka, odnosno akcija, ovisno o postavkama atributa *summary_freq* u konfiguracijskoj datoteci modela strojnog učenja, na ekranu ispisuju broj odrađenih koraka, proteklo vrijeme, broj skupljenih nagrada i razina odstupanja osvojenih nagrada. Iz tih podataka lako se može uočiti ako, na primjer, razina nagrada počne opadati, tada bi to moglo značiti dvije stvari, ili da je agent zapeo i ne zna kako napredovati ili da je odlučio iskorištavati naučeno znanje i tako zaustaviti proces daljnjeg učenja. U slučaju gdje su pozitivne i negativne nagrade ne izbalansirane može se dogoditi da se agent počne kretati u mjestu i biti zadovoljan sa razinom nagrade u iznosu 0. Dok se opcija praćenja putem terminala primarno koristila prilikom aktivnog učenja agenata, nakon svakih nekoliko simulacija otvoren je *TensorBoard* kako bi se provjerio generalan odnos između učenja, odnosno da se vizualizira brzina i kvaliteta kojom agent uči pojedine zadatke. Nakon što se putem *Python* virtualnog okruženja pokrene, *TensorBoard* se otvara na određenom portu u obliku lokalne web aplikacije. Za prikaz podataka koristi mapu *results* iz *Unity* projekta igre, gdje su pohranjeni svi podaci učenja. Na slici 36 može se vidjeti općenit izgled alata i većina funkcionalnosti koje nudi. S lijeve strane nalazi se popis svih rezultata učenja koji mogu biti uspoređeni jedni s drugima. U sredini je predviđeno mjesto za prikazivanje raznih grafova,

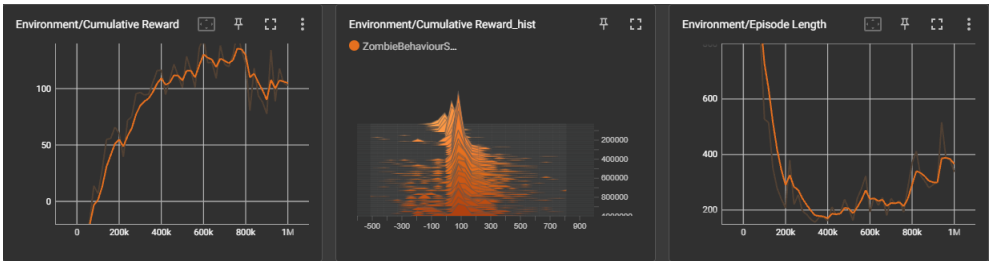
dok se skroz sa desne strane nalaze razne postavke kojima korisnik može prilagoditi sučelje *TensorBoard-a* svojim potrebama i željama. Analize grafova odrađene su po skupinama simulacija učenja pojedinih zadataka.



Slika 36: Prikaz alata *TensorBoard*

Simulacija broj 1

Iz prikazanih podataka simulacije 1, slika 37, može se vidjeti da je agent do 400000 koraka postepeno učio izvršavati zadatak te nakon toga pokušavao optimizirati svoje postupke. Usporedno krivulja grafa nagrada raste, dok se krivulja grafa trajanja epizoda smanjuje, što znači da je do kraja simulacije sigurno izvršio zadatak stavljen pred njega. Iz histograma nagrada može se vidjeti raspršenost podataka koja je krenula nakon 400000 koraka gdje je agent krenuo sa optimizacijom.

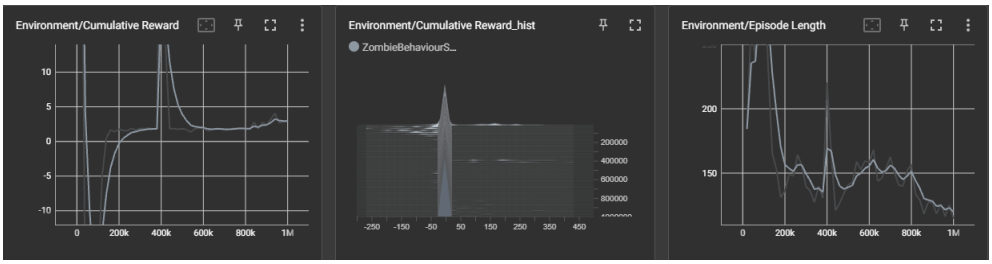


Slika 37: Analiza podataka simulacije 1

Simulacija broj 2

Iz prikazanih podataka simulacije 2, slika 38, može se uočiti jedno neuobičajeno ponašanje agenta tokom simulacije gdje od samog starta nagrada naglo opada u ekstremni minus, a trajanje epizode postaje ekstremno dugo. Sve do iza 200000 koraka gdje agent naglo, odjednom, otkriva novo znanje i nakon toga jako dobro optimizira pozitivno ponašanje. Isto pokazuje i graf trajanja epizoda, gdje se nakon 600000 koraka vrijeme postepeno smanjuje. Histogram nagrada je poprilično konstantan tokom cijele simulacije, sa naglaskom na početak gdje se vidi

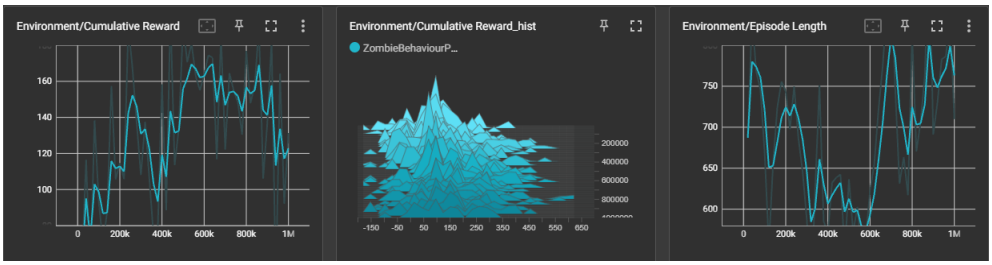
poprilična raspršenost podataka.



Slika 38: Analiza podataka simulacije 2

Simulacija broj 3

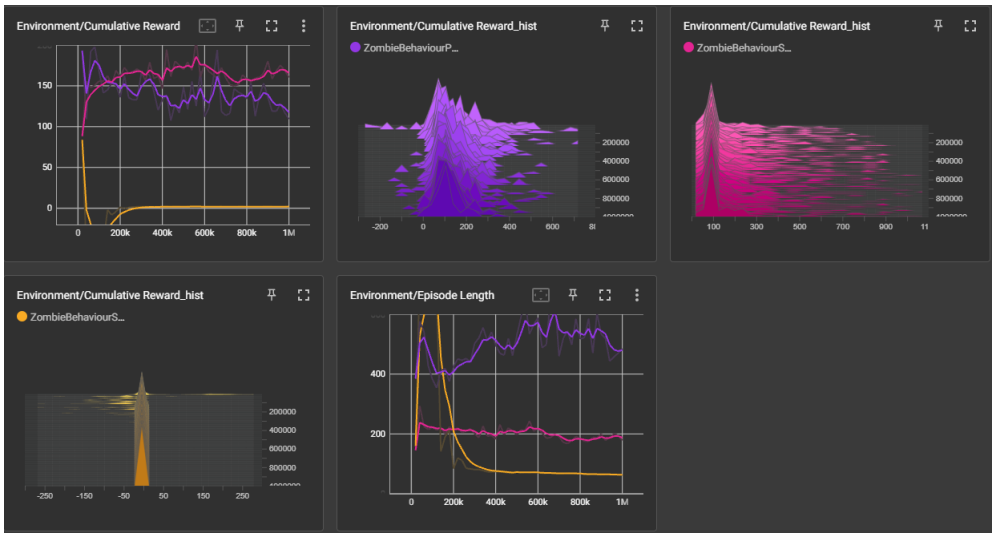
Iz prikazanih podataka simulacije 3, slika 39, prvo što je potrebno uočiti je histogram nagrada koji je od početnog koraka do zadnjeg, podatkovno, jako raspršen, što znači da tokom simulacije agent nije uspio optimizirati učeno ponašanje. Isto se može zaključiti iz grafova nagrada i trajanja epizoda koji su također vrlo nekonzistentni. Uspješnost se donekle može zaključiti iz toga da nema ekstrema koji bi totalno poremetili učenje i uzrokovali njegovo ponavljanje te potencijalne izmjene agenta ili okruženja.



Slika 39: Analiza podataka simulacije 3

Simulacija broj 4 do 6

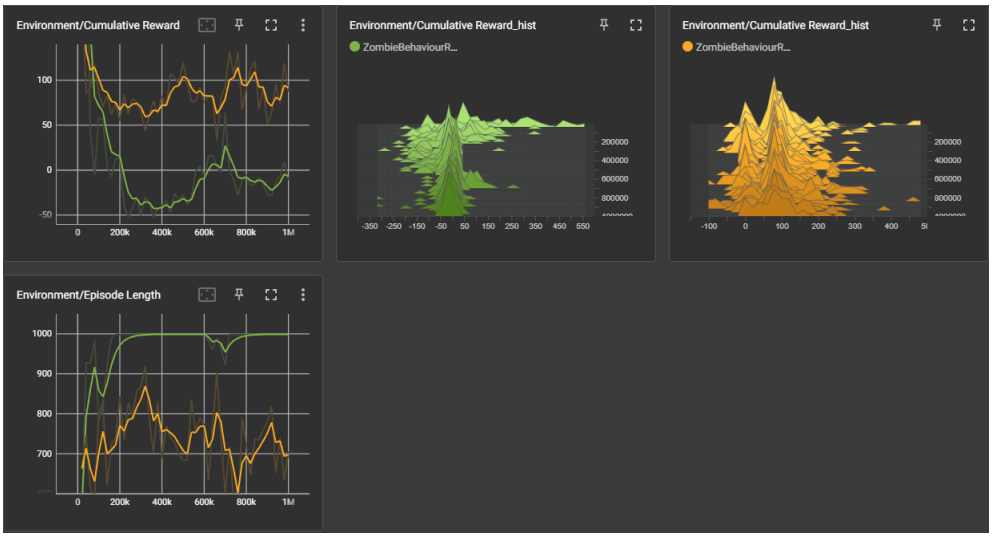
Iz prikazanih podataka simulacija 4, 5 i 6, slika 40, može se uočiti veliki napredak u odnosu na prethodne tri simulacije. Roza boja označava promjenu u odnosnu na simulaciju broj 1. Graf nagrada je konzistentniji i većih vrijednosti, histogram nagrada je i dalje raspršen no sa puno izraženijim i konzistentnijim vrhovima te graf trajanja epizoda pokazuje da epizode od samog starta traju kraće sa kojim trendom i završavaju. Žuta boja označava promjenu u odnosu na simulaciju broj 2 gdje se na sva tri grafa uočava dosta sličan, ali bolji trend. Ljubičasta boja označava promjenu u odnosu na simulaciju broj 3, koja je ujedno i najznačajnija. Grafovi nagrada i trajanja epizoda pokazuju mnogo više podatkovne konzistentnosti, što se također može potkrijepiti znatno manjom raspršenosti histograma nagrada.



Slika 40: Analiza podataka simulacija 4 i 6

Simulacija broj 7 do 26

Iz prikazanih podataka simulacija 7 i 26, slika 41, može se uočiti vrlo poželjno učenje agenta. Zelena boja označava simulaciju broj 7, dok žuta boja označava promjenu u odnosu na simulaciju broj 7, odnosno promjenu učenja koju je agent postigao u okviru 20 učenja. Grafovi nagrada i trajanja epizoda jasno pokazuju veliki napredak gdje je nagrada postala veća i konzistentnija, a vrijeme se uglavnom smanjilo uz malo veća podatkovna odstupanja. Trajanje učenja se u ovom slučaju može povezati sa histogramom nagrada, gdje se vidi da je agent prešao sa naučenog i relativno konzistentnog stanja u jedno istraživačko stanje gdje su podaci postali raspršeni. Zbog ostala dva grafa može se zaključiti napredak agenta i želja za postizanjem optimizacije učenja.

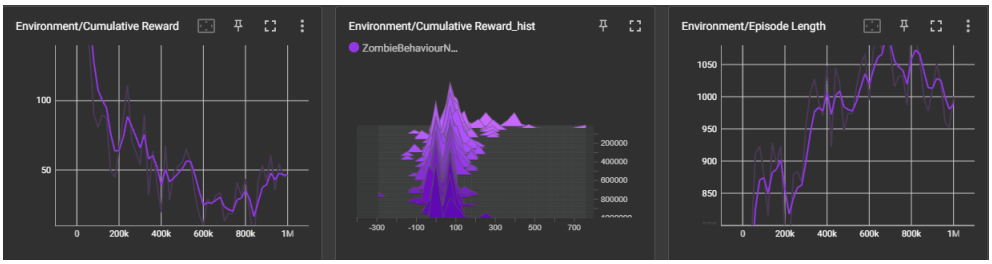


Slika 41: Analiza podataka simulacija 7 i 26

Simulacija broj 27

Iz prikazanih podataka simulacije 27, slika 42, može se uočiti kako je u početku agent iskorištavao naučeno znanje tako što graf nagrada do 100000 koraka ima ekstremno visoke

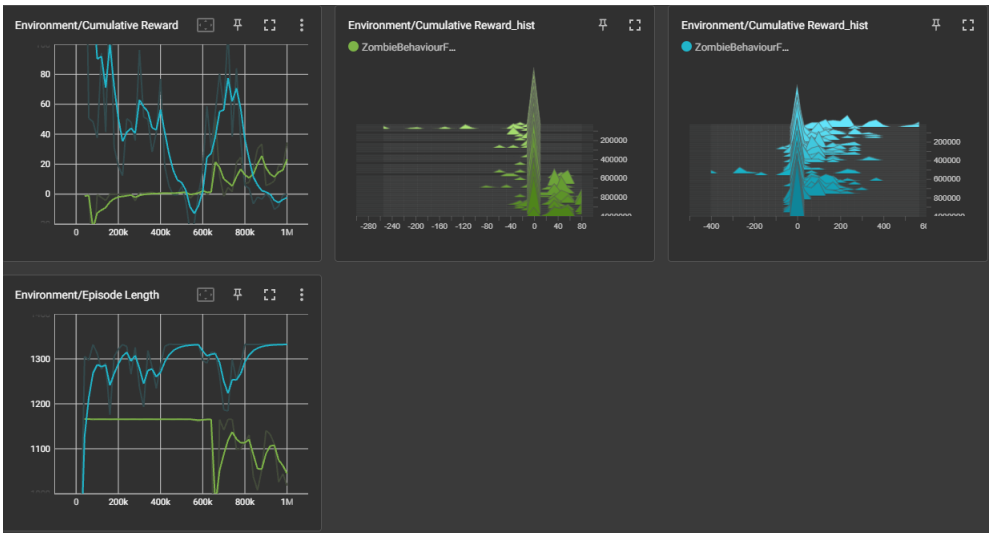
nagrade, a graf trajanja epizoda ekstremno kratke epizode. Nakon 200000 koraka agent postepeno počinje sa učenjem što se najviše vidi iz grafa trajanja epizoda gdje su one sve duže i taj trend traje sve do završnih epizoda simulacije nakon kojih kreće optimizacija.



Slika 42: Analiza podataka simulacije 27

Simulacija broj 28 do 31

Iz prikazanih podataka simulacija 28 i 31, slika 43, može se vidjeti jedan vrlo turbulentan set simulacija. Zelena boja označava simulaciju broj 28, plava boja označava simulaciju broj 31, odnosno promjenu učenja koju je agent postigao u okviru 4 učenja. Iz grafova nagrada i trajanja učenja vidi se da je početna, 28. simulacija, vrlo konzistentna sve do 600000 koraka gdje agent vrlo brzo otkriva novo znanje, vrijeme epizode se smanjuje, a nagrada raste. Set simulacija završava se učenjem koje jako odstupa po vrijednostima podataka, gdje se epizodama naglo povećava vrijeme trajanja, a nagrade variraju između manjih i većih. Donekle pozitivan trend može se uočiti sa histograma nagrada gdje su kroz simulaciju broj 31 podaci uglavnom raspršeni na strani pozitivnih vrijednosti. Ta informacija ukazuje da iako učenje agenta varira, ono varira u pozitivnom smjeru gdje agent uči nove kompleksne zadatke.

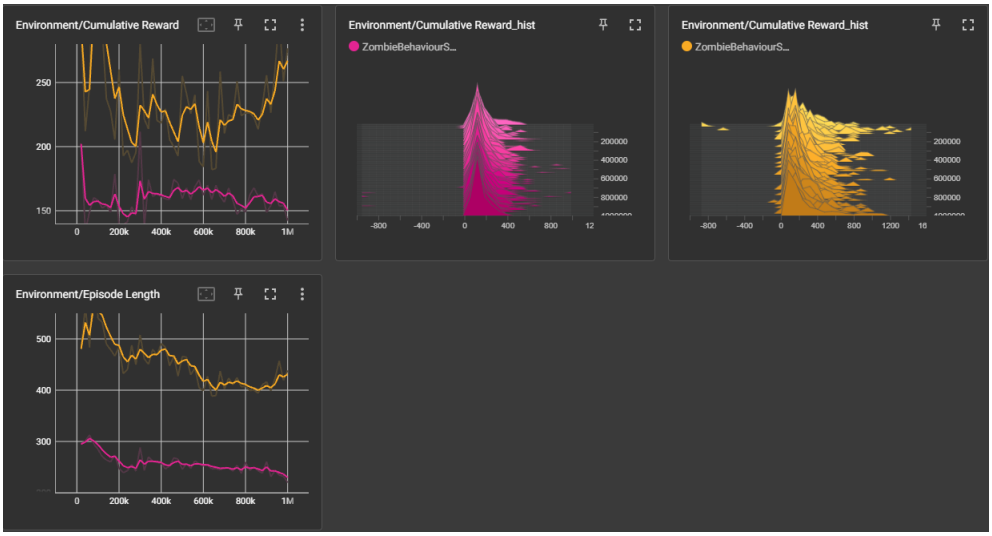


Slika 43: Analiza podataka simulacija 28 i 31

Simulacija broj 32 do 40

Iz prikazanih podataka simulacija 32 i 40, slika 44, uočava se jedan vrlo konzistentan pozitivan trend tokom svih učenja koje je agent odradio u okviru 9 simulacija. Roza boja označava simulaciju broj 32, a žuta boja označava simulaciju broj 40. Rezultati ovog seta simulacija govore o tome da su postavke agenta kao i njegovog okruženja bile postavljene vrlo primjereno

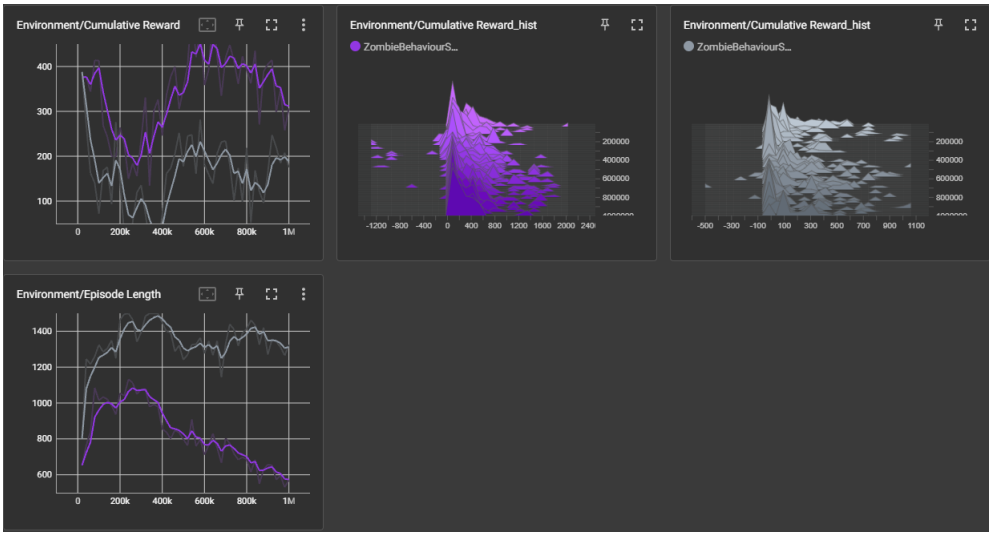
stadiju razvoja modela strojnog učenja. Iz histograma nagrada dodatno se još može uočiti namjera agenta za daljnjom optimizacijom, istraživanjem okruženja i novim spoznaja koje bi ga mogle dovesti do ostvarivanja zadatka koji mu je ultimativno i namijenjen.



Slika 44: Analiza podataka simulacija 32 i 40

Simulacija broj 41 do 51

Iz prikazanih podataka simulacija 41 i 51, slika 45, može se zaključiti da je agent kroz ovaj set simulacija naučio sve što je stavljeno pred njega. Ljubičasta boja označava simulaciju broj 41, dok siva boja označava promjenu u odnosu na simulaciju broj 41, odnosno promjenu učenja koju je agent postigao u okviru 11 učenja. Grafovi nagrada i trajanja epizoda pokazuju ujednačenost simulacija broj 41 i 51 jer su im krivulje gotovo identične, samo što su podaci optimiziraniji te se isto odnosi i na histogram nagrada. Što bi značilo, da je agent naučio izvršavati zadatak, te kako postupak izvršavanja optimizirati.

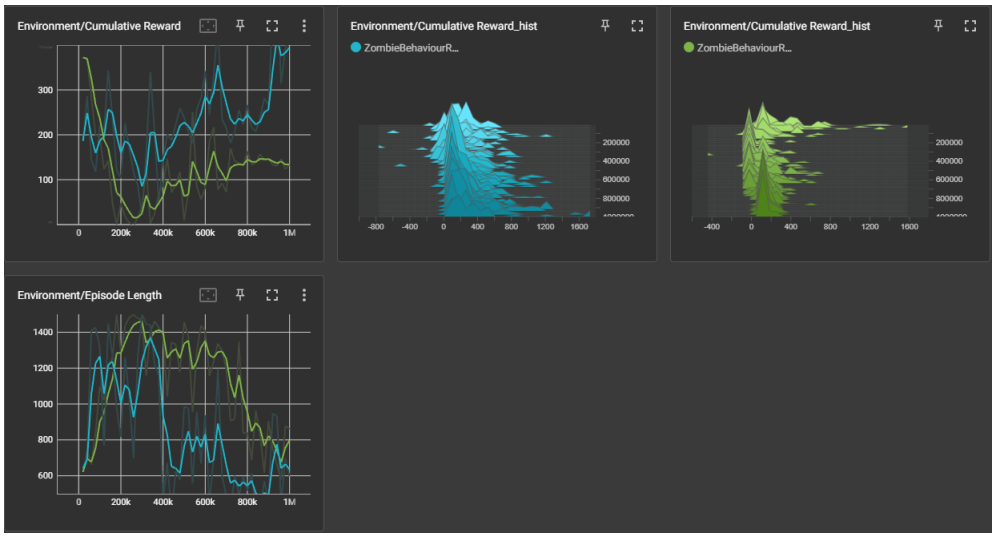


Slika 45: Analiza podataka simulacija 41 i 51

Simulacija broj 52 do 70

Iz prikazanih podataka simulacija 52 i 70, slika 46, mogu se uočiti finalne optimiza-

cije koje je agent proveo. Plava boja označava simulaciju broj 52, dok zelena boja označava promjenu u odnosu na simulaciju broj 52, odnosno promjenu učenja koju je agent postigao u okviru 19 učenja. Grafovi nagrada i trajanja epizoda, za prvu simulaciju broj 52 prikazuju oscilacije podataka učenja što zajedno sa pozitivnom raspršenosti podataka na histogrmu nagrada pokazuje da je agent na počeku ovog seta simulacija pokušavao istraživanjem dodatno optimizirati svoje ponašanje. Graf nagrada zadnje simulacije broj 70 jasno izgledom svoje krivulje ukazuje da je agent uspio do kraja optimizirati svoje učenje. To je vidljivo po tome što nakon 200000 koraka krivulja prelazi u trend postepenog rasta, gdje nakon 700000 koraka podaci postaju poprilično konzistentni. Graf trajanja epizoda, simulacije broj 70, svojom krivuljom prati trend grafa nagrada gdje u trenutku 700000 koraka kada nagrade postaju konzistentne, vrijeme trajanja epizoda znatno opada. Raspršenost podataka histograma nagrada zadnje simulacije, u odnosnu na simulaciju broj 52, je manja sa obje, negativne i pozitivne strane. Pošto se radi o zadnjoj simulaciji, to ukazuje na to da je agent uspješno optimizirao svoje učenje i da je u trenutnom okruženju i sa trenutnim konfiguracijama postigao najbolje rezultate koje je mogao.



Slika 46: Analiza podataka simulacija 52 i 70

6. Zaključak

Platforma *Unity* sa svojom širokom paletom alata i funkcionalnosti definitivno predstavlja konkurentno rješenje za izradu igara za razne sustave. Kada se svi alati upoznaju i savladaju vještine rada s njima, samo je mašta granica mogućnostima koje se mogu s njima napraviti. Kroz ovaj rad prikazane su pojedinosti povezane sa zvukovima, animacijama, objektima, raznim bibliotekama, ali i sa općenitim alatima kao što su scena i hijerarhija te druga radna sučelja. Pomoću njih izrađeni su mapa, korisnička sučelja, izgled i funkcionalnosti igrača, izgled i funkcionalnosti zombija te razni ostali sustavi koji ih povezuju u jednu veliku cjelinu. Dodatne funkcionalnosti vezane su uz rad korisničkog sučelja, toka igre i manipulacije zvučnih i vizualnih efekata. Korisničko sučelje na kojem su smješteni razni informacijski prozori o stanju igre, kao što su život igrača, količine noževa za bacanje i druge, također je vezano uz sve ekrane koji se nalaze prije, tokom i nakon glavnog dijela igre. Tok igre vodi igrača od početka do kraja i to raznim uputama koje su izvedene tekstualno kroz razne poruke na ekranu, vizualno kroz animacije i objekte te zvukovno kroz razne indikatore koji dodatno opisuju trenutno stanje igre. Manipulacija efektima odnosi se na razne načine kojima se može odrediti vrijeme i način njihovog izvođenja, kao i ostali aspekti ovisno o tome radili se o zvukovima ili animacijama. Sva logika iza spomenutih pojmova odrađena je uz pomoć programskog jezika *C#*.

Uz razvoj igre, kroz ovaj rad razvijen je i implementiran višeagentni sustav. To je sustav koji se sastoji od više agenata upravljanih od strane modela strojnog učenja, odnosno njihovog mozga. Pri tome pomogao je alat *ML-Agents* koji se osim kroz *Unity* koristi zajedno u kombinaciji sa *Pyton virtualnim okruženjem*. *Unity* služi kao okruženje u kojem agenti opažaju svoju okolinu i izvršavaju akcije, dok je *Python* zadužen za za obradu podataka tih opažanja i donošenja odluka na temelju njih, koje se potom šalju nazad agetnu u *Unity*. Sam proces učenja agenta vezan je uz poticano učenje, gdje se agenta nagrađuje ili kažnjava za njegove odluke, odnosno izvršene akcije. Prije razvoja i implementacije, teorijski su pokriveni svi važni aspekti višeagentnih sustava kao što su sam pojam umjetne inteligencije i njezine grane od kojih je strojno učenje jedna od njih. Kroz proces implementacije prikazane su konfiguracije agenta i njegovog okruženja, kao i konfiguracije modela strojnog učenja. Po uspješnom završetku učenja, pomoću funkcionalnosti *TensorBoard*, koju nudi alat *ML-Agents* analizirane su sve ključne simulacije u kojima je agent naučio kako se kretati po mapi, izbjegavati prepreke, uništavati ograde, pronaći igrača te prioritizirati interakcije s objektima.

Umjetna inteligencija i računalne igre vrlo su popularni pojmovi u današnje vrijeme. Obje teme na svoj način pridonose otkrivanju granica tehnologija koje se koriste svakodnevno u raznim oblicima. Umjetna inteligencija sa svojim opusom primjene obuhvaća gotovo svaki aspekt koji se fizički može promatrati, dok računalne igre, osim svoje zabavne strane, također nude oblik simulacije u kojem se inicijalna programska rješenja umjetne inteligencije mogu, brzo i jednostavno, testirati u virtualnom okruženju prije stvarne primjene.

Popis literature

[1] *Unity User Manual 2022.3 (LTS)*, `https://docs.unity3d.com/2022.3/Documentation/Manual/index.html`, Preuzeto: 03.01.2024.

[2] W. Goldstone, *Unity 3.x Game Development Essentials*. Packt, 2011.

[3] W. Ertel, *Introduction to Artificial Intelligence*. Springer, 2018.

[4] S. J. Russell i P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2021.

[5] M. Mohri, A. Rostamizadeh i A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2018.

[6] E. Alpaydin, *Machine Learning: The New AI*. The MIT Press, 2017.

[7] G. Bonaccorso, *Machine Learning Algorithms*. Packt, 2018.

[8] G. Weiss, *Multiagent Systems*. The MIT Press, 2013.

[9] M. S. Mahmoud, *Multiagent Systems: Introduction and Coordination Control*. CRC Press, 2020.

[10] D. Engelbrecht, *Introduction to Unity ML-Agents: Understand the Interplay of Neural Networks and Simulation Space Using the Unity ML-Agents Package*. Apress, 2023.

[11] *Unity-Technologies/ml-agents*, `https://github.com/Unity-Technologies/ml-agents`, Preuzeto: 05.01.2024.

Popis slika

Sadržaj