Izrada kooperativne pucačine iz ptičje perspektive u Unity razvojnom okruženju

Salopek, Danko

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

Danko Salopek

Izrada kooperativne pucačine iz ptičje perspektive u Unity razvojnom okruženju

Diplomski rad

Pula, rujan, 2023.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

Danko Salopek

Izrada kooperativne pucačine iz ptičje perspektive u Unity razvojnom okruženju

Diplomski rad

JMBAG: 0303069638, redoviti student

Studijski smjer: Informatika

Kolegij: Dizajn i programiranje računalnih igara

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

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

Pula, rujan, 2023.

IZJAVA O AKADEMSKOJ ČESTITOSTI

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

Student

Danko Salopek

U Puli, rujan, 2023. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Danko Salopek dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Izrada kooperativne pucačine iz ptičje perspektive u Unity razvojnom okruženju koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

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

U Puli, rujan, 2023. godine

Potpis

Danko Salopek

Sadržaj

1. Uvod 1
2. Inspiracija 2
   2.1. Darkwood 2
   2.2. Call of Duty: Zombies 3
   2.3. Alien Swarm: Reactive Drop 3
3. Pucačina iz ptičje perspektive 4
4. Unity 4
   4.1. Univerzalni cjevovod renderiranja (Universal Render Pipeline (URP)) 5
5. Unity Relay 7
   5.1. Implementacija Relaya 7
6. Unity Lobby Service 8
   6.1. Implementacija Lobby servisa 9
7. Unity Netcode for GameObjects (NGO) 13
   7.1. Stope ažuriranja 13
      7.1.1. Server authoritative igre 14
      7.1.2. Klijent autoritativna igra 15
   7.2. Network Manager 15
   7.3. NetworkObject 17
   7.4. NetworkBehaviour 18
      7.4.1. RPC funkcije i NetworkVariables 18
   7.5. NetworkTransform 19
   7.6. NetworkAnimator 20
   7.7. ClientNetwork komponente 20
8. Glavni izbornik 20
9. Korisničko sučelje u igri 25
10. Igrač 28
   10.1. Player skripta 32
   10.2. PlayerInteraction skripta 34
   10.3. AudioPlayer skripta 35
   10.4. Stats skripta 36
   10.5. PlayerDead skripta 39
   10.6. PlayerGuns skripta 41
11. Oružje 45
   11.1. Weapon skripta 46

12. Zombi 51
12.1. Zombie skripta 54
13. Škrinja s oružjem 56
13.1. SpawnGun skripta 56
13.2. InteractableText skripta 58
14. Generator 58
14.1. PowerGeneratorV2 skripta 59
15. Granata i baklja 61
16. Izrada mape 62
16.1. Osvjetljenje mape 63
16.1.1. Svjetleće probe (Light Probes) 63
16.1.2. Promjena svjetla 64
16.2. Kamera i praćenje igrača 68
17. Zvukovi 71
18. Zaključak 73
Literatura 76
Popis slika 78

1. Uvod

U ovom diplomskom radu će biti opisan proces izrade kooperativne pucačine iz ptičje perspektive pod nazivom Stellar Decay. Verzija Unity razvojnog okruženja koja je korištena za izradu ovog diplomskog rada je 2022.2.8f1.

Za izradu ovoga rada korišten je Unity Netcode okvir za povezivanje razvijen od strane Unity Technologies za stvaranje višekorisničkih igara i aplikacija. namijenjen da pojednostavi proces dodavanja funkcionalnosti višekorisničkog načina rada u Unity projektima.

Uz Unity Netcode će se isto koristiti Unity Relay usluga koja omogućuje lakšu implementaciju mrežnih komunikacija s Unity igrama i aplikacijama. Posebno je koristan zato što olakšava povezivanje igrača putem mreže, bez potrebe za konfiguracijom vlastitih poslužitelja ili upravljanjem složenih mrežnih veza.

Također, dodatno se koristi Unity Lobby sustav koji omogućuje virtualni prostor unutar igre ili aplikacije gdje se igrači mogu skupiti i pripremiti za kooperativno igranje. Omogućuje igračima da kreiraju svoje virtualne sobe ili da se pridruže već postojećim sobama. Te da mijenjaju parametre igre i komuniciraju međusobno prije nego što započnu stvarnu igru.

Kada igra započne, igračev glavni cilj je preživjeti zombi epidemiju koja je zahvatila svemirski brod na kojem se trenutno igrači nalaze. Igrači na raspolaganju imaju asortiman oružja s kojima mogu ubijati zombije dok ne poprave sve generatore i aktiviraju zaštitno polje. Pokretanje generatora igračima omogućuje bolju vidljivost na samoj mapi. Na mapi su raspršene škrinje koje igrač može otvoriti te će mu one tada dati neko novo oružje.

U radu su detaljno opisane skripte koje se koriste na pojedinim objektima, kao i komponente prisutne na tim objektima. Poseban fokus je na dubljem razumijevanju funkcionalnosti igračevog objekta i objekta neprijatelja. Interakcija igrača s raznim elementima igre poput škrinja, oružja i generatora detaljno je objašnjena.

Objašnjeno je osvjetljenje mape te način na koji se ono koristi, kako se mijenjaju svjetlosne teksture i što je sve potrebno da se svjetla mogu izmjenjivati tijekom igre. Dodatno opisuje svjetlosne probe, kakav one utjecaj imaju na objekte te kako se izmjenjuju sami podaci svjetlosnih proba.

2. Inspiracija

Inspiracija za ovu igru dolazi iz tri igre: Darkwood, Call of Duty: Zombies i Alien Swarm: Reactive Drop. Igra ima za cilj unijeti osjećaj sličan Darkwoodu, gdje igrač ne može lako vidjeti odakle dolaze neprijatelji sve dok ne upali svjetlo, stvarajući konstantan osjećaj nesigurnosti. Neprijatelji su slični onima iz Call of Duty: Zombiesa i Alien Swarm: Reactive Dropa, gdje pokušavaju doći do igrača i ubiti ga, dok igrač ima mogućnost mijenjati oružje prema potrebi. U nastavku se nalazi malo detaljniji opis igara koji su inspirirali ovaj projekt.

2.1. Darkwood

Darkwood je razvijen i izdan od strane nezavisnog razvojnog studija iz Poljske zvanog Acid Wizard Studio. Riječ je o survival horor igri koja je privukla pažnju svojom mračnom atmosferom i jedinstvenim stilom igranja. Igra je izdana 2017 godine [11].

Igra ima polu otvoren svijet, sustav za stvaranje predmeta, dan/noć ciklus, interakciju s NPC-jevima, sustav vještina koje igrač postepeno otključava, borbu s neprijateljima i više krajeva priče. Tijekom dana igrač istražuje svijet, izrađuje predmete i brani skloništе, dok se noću mora braniti od neprijatelja koji mu provaljuju u skloništе. Ako preživi noć dobiva ugled s trgovcem. Priča se oblikuje prema odabirima igrača, s različitim završecima i promjenama u svijetu. Slika 1 prikazuje snimku ekrana iz igrice Darkwood.

Slika 1. Snimka ekrana iz igrice Darkwood

2.2. Call of Duty: Zombies

Call of Duty: Zombies, poznata kao Nazi Zombies ili Zombies mod, razvijena je od strane tima iz Treyarcha, koji je poznat po svojim doprinosima Call of Duty serijalu. Treyarch je razvio ovaj mod koji se pojavljuje u nekoliko Call of Duty igara, počevši sa Call of Duty: World at Warom. Mod je stekao veliku popularnost zbog svoje jednostavne igrivosti, borbe s valovima zombija i kompleksnih tajni koje igrači mogu otkriti tijekom igranja [12].

Cilj igre je preživjeti valove napada zombija. Igrač počinje s pištoljem i nožem. Tijekom borbe protiv zombija, igrači zarađuju bodove koje mogu koristiti za kupovinu oružja, bonusa (perkova) i otvaranja novih dijelova mape. Zombiji ulaze u mapu kroz barikade, a kako runde napreduju, postaju sve brži u trčanju i rušenju tih barikada. Bodovi se dobivaju ubijanjem zombija i popravljanjem barikada. Ponekad se, nakon ubijanja zombija, pojavljuju slučajni predmeti koji pomažu igračima. Slika 2 prikazuje snimku ekrana iz zombi moda Call of Duty: World at Wara.

Slika 2. Snimka ekrana iz zombi moda Call of Duty: World at War

2.3. Alien Swarm: Reactive Drop

Alien Swarm: Reactive Drop je samostalno proširenje na igru Alien Swarm koju je razvio Valve. Ova igra igrače stavlja u epski lov na kukce koji uključuje jednostavnu kombinaciju kooperativnog igranja i taktika na razini tima. Igrači formiraju četveročlani ili osmeročlani tim s četiri različite klase IAF (Interstellar Armed Forces) marinaca. Planiraju svoje napade

koristeći raznovrstan arsenal oružja s brojnim konfiguracijama protiv različitih vrsta kukaca. Kroz različite lokacije, zajedno se bore protiv invazije kukaca [10]. Slika 3 prikazuje snimku ekrana iz igrice Alien Swarm: Reactive drop.

Slika 3. Snimka ekrana iz Alien Swarm: Reactive Dropa

3. Pucačina iz ptičje perspektive

Pucačina iz ptičje perspektive (eng. Top-Down Shooter) je žanr video igara koji se odlikuje perspektivom gledanja odozgo prema dolje, što znači da igrač promatra akciju s ptičje perspektive, gledajući na likove i okoliš odozgo, što igračima omogućuje bolji pregled mape.

Primjeri poznatih igara ovog žanra su Hotline Miami, Enter the Gungeon, Helldivers i Alien Swarm. Ovaj žanr naglašava brzu akciju i pucanje te pruža dinamično iskustvo borbe, a igračima omogućuje da koriste svoje vještine i strategije kako bi preživjeli i napredovali kroz izazove.

4. Unity

Unity je platforma za razvoj video igara i interaktivnih sadržaja osnovana 2005. Godine. Svojom pristupačnosti i fleksibilnosti brzo je osvojio razvojnu scenu, te je postao ključni alat za izradu igara i simulacija. Kroz verzije poput Unity 4 i Unity 5 dodana je podrška za mobilne

platforme i unaprijeđene su grafičke sposobnosti. Danas je jedan od najpoznatijih alata za izradu video igara te se uz to koristi u filmskoj industriji, autoindustriji i obrazovanju.

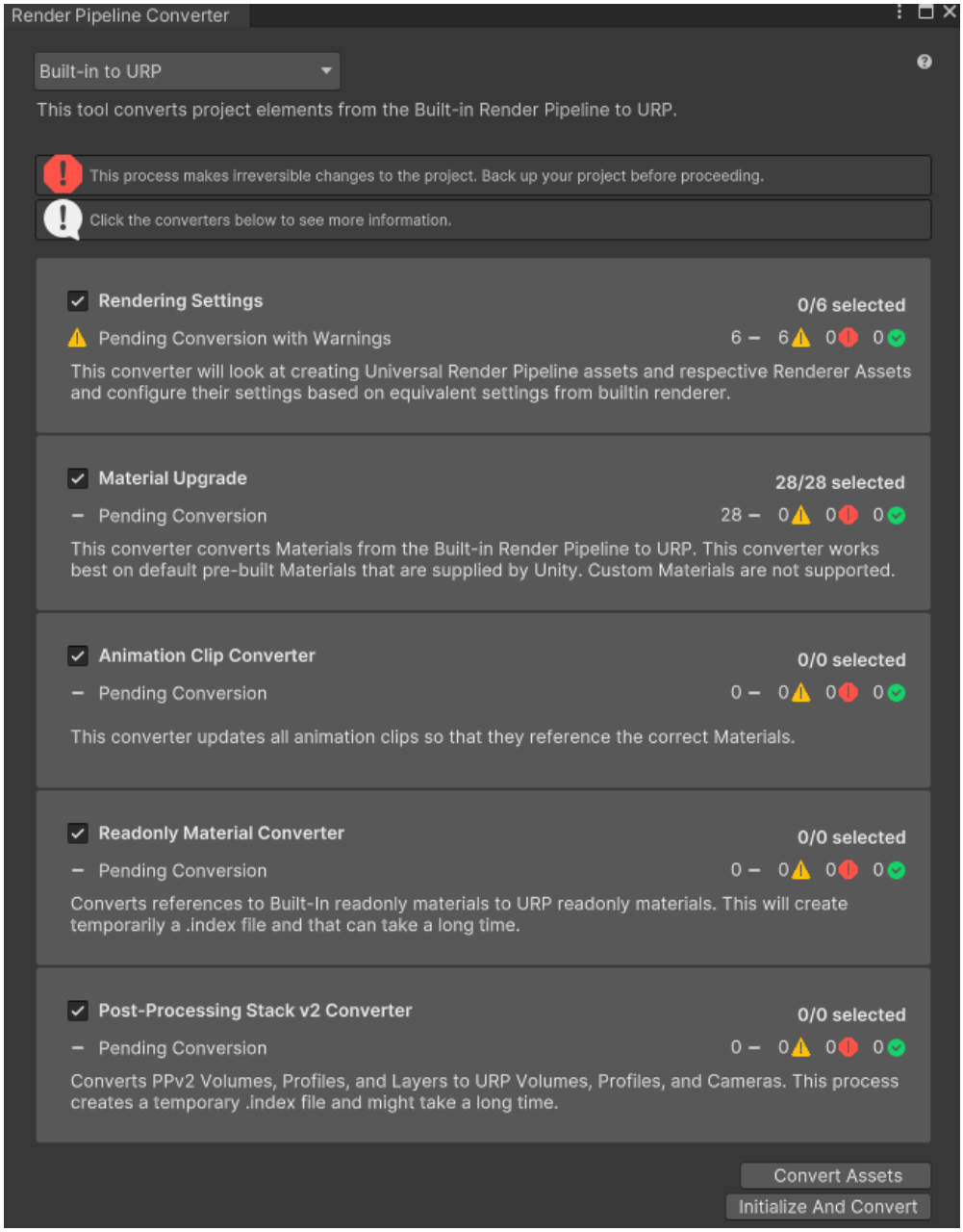
Jedan od ključnih razloga za popularnost Unityja je njegova ogromna zajednica i podrška. Developeri mogu pristupiti širokom rasponu resursa, tutorijala, foruma i dodataka koji im pomažu da brže napreduju u svojim projektima.

4.1. Univerzalni cjevovod renderiranja (Universal Render Pipeline (URP))

Postoji nekoliko vrsta cjevovoda za renderiranje [1]:

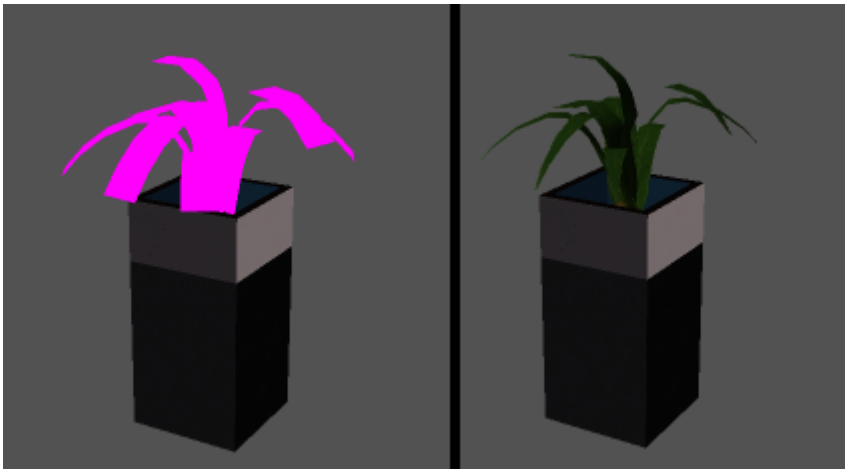
- Ugrađeni cjevovod za renderiranje je Unityjev zadani cjevovod. Cjevovod je opće namjene koji ima ograničene mogućnosti prilagodbe. Stariji je cjevovod koji je sada zamijenjen URP-om i HDRP-om. Iako je manje fleksibilan i nudi manje vizualnih efekata u usporedbi s novim cjevovodima, može biti koristan za starije projekte ili manje zahtjevne aplikacije
- Univerzalni cjevovod (URP) služi za renderiranje koje se može skriptirati. Brzo i jednostavno se prilagođava i omogućuje stvaranje optimiziranje grafike za širok raspon platformi. Nudi balans između performansi i vizualne kvalitete te podržava efekte poput sjena, svjetla i reflektiranja
- Visoko definirani cjevovod za renderiranje (High Definition Render Pipeline (HDRP) služi za renderiranje koje se može skriptirati. Stvara grafike visoke kvalitete na hardverski jakim platformama. Pruža napredne efekte kao što su globalna iluminacija, fotorealistični materijali i visoko kvalitetne teksture. Idealan je za projekte koji zahtijevaju visoku kvalitetu, poput realističnih igara i filmske produkcije
- Zadnja opcija je kreiranje novog cjevovoda za renderiranje pomoću Unityjevog API-a za kreiranje cjevovoda (Scriptable Render Pipeline (SRP))

Ovaj diplomski projekt je prvobitno bio izrađen na Ugrađenom cjevovodu, ali nakon nekog vremena izrade došli su na vidjelo limitacije tog cjevovoda. Tako da se cjevovod projekta promijenjen na Univerzalni. Sama promjena s jednog cjevovoda na drugi je relativno lagana. Unity daje mogućnost promjene cijelog projekta iz Ugrađenog cjevovoda u Univerzalni cjevovod, ako se ode pod Window->Rendering->Render Pipeline Converter otvori se novi prozor s opcijama za promjenu cjevovoda prikazan na slici 4.



Slika 4. Prikaz Render Pipeline Converter prozora u Unityju

Nakon promjene postoji mogućnost da se neki elementi nisu mogli promijeniti. Ti će elementi biti prikazani u prozoru. Pa se ti elementi mogu pronaći u projektu i moraju se popraviti manualno. Ako se elementi ne poprave, neki objekti u sceni mogu imati različite probleme, poput nepostojećih tekstura, kao što je prikazano na slici 5.



Slika 5. Prikaz problema nepostojeće teksture

5. Unity Relay

Unity Relay omogućuje developerima igara da pruže poboljšanu komunikaciju između igrača koristeći efikasan pristup povezivanja, bez potrebe za složenim trećim stranama ili održavanjem vlastitih poslužitelja za igre. Umjesto tradicionalnog pristupa koji zahtijevaju namjenske poslužitelje ili kompleksno rukovanje mrežama, Relay olakšava ovaj proces putem jednostavnog tijeka rada. Relay poslužitelj djeluje kao posrednik, omogućavajući sigurno povezivanje između igrača bez potrebe dodatnih ulaganja ili infrastrukture. Ovo pruža razvojnim timovima više resursa za ulaganje u samu igru i poboljšava iskustvo igrača kroz stabilno i pouzdano online igranje [3].

Relay je idealan za igre gdje jedan igrač djeluje kao host, dok su ostali igrači klijenti.
Relay djeluje kao javna točka dostupna svim igračima. Ovaj dizajn rješava uobičajene probleme promjena mreža i IP adresa, provođenje mrežnih adresa (NAT) i vatrozida između igrača. Svaki igrač se može povezati s istom IP adresom i priključkom (IP adresa i priključak odabranog relay poslužitelja), a igrači mogu vjerovati da će informacije o vezi ostati iste tijekom cijele igre. Kroz ovu neizravnu vezu, igrači ne trebaju znati IP adrese jedni drugih, čime se povećava sigurnost i privatnost.

5.1. Implementacija Relaya

TestRelay skripta omogućuje igraču da pokrene server s funkcijom CreateRelay ili da se pridruži igraču koji je pokrenuo neki server s funkcijom JoinRelay. Te dvije funkcije se pozivaju u LobbySystem skripti prilikom pokretanja igre iz sobe gdje se nalaze igrači. CreateRelay i JoinRelay funkcije su prikazane na slikama 6 i 7.

```
public async Task<string> CreateRelay()
{
    try
    {
        Allocation allocation = await RelayService.Instance.CreateAllocationAsync(4);

        string joinCode = await RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);
        serverCode = joinCode;
        RelayServerData relayServerData = new RelayServerData(allocation, "dtls");
        NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);
        NetworkManager.Singleton.StartHost();
        Game.PlayerSpawnerNewScene.Instance.SceneLoadEvent();
        NetworkManager.Singleton.SceneManager.LoadScene("Sample",LoadSceneMode.Single);

        //canvas.gameObject.SetActive(false);
        Debug.Log(serverCode);
        return serverCode;
    }
    catch (RelayServiceException e)
    {

        Debug.Log(e);
        return "0";
    }
}
```

Slika 6. CreateRelay funkcija iz TestRelay skripte

```
public async void JoinRelay(string joinCode)
{
    try
    {
        Debug.Log("Joining relay with " + joinCode);
        JoinAllocation joinAllocation = await RelayService.Instance.JoinAllocationAsync(joinCode);
        serverCode = joinCode;
        RelayServerData relayServerData = new RelayServerData(joinAllocation, "dtls");
        NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);
        NetworkManager.Singleton.StartClient();

        Game.PlayerSpawnerNewScene.Instance.SceneLoadEvent();

        //canvas.gameObject.SetActive(false);
    }
    catch (RelayServiceException e)
    {

        Debug.Log(e);
    }

}
```

Slika 7. JoinRelay funkcija iz TestRelay skripte

6. Unity Lobby Service

Lobby servis developerima omogućuje da lako implementiraju povezivanje između igrača. Neke od opcija koje Lobby servis nudi developerima su [2]:

- Preglednik svih trenutno aktivnih soba (lobbyja) što igračima pruža opciju da se pridruže nekoj od tih soba
- Svaka soba ima svoj kod za pristup, tako da igrač koji je napravio sobu (host) može poslati taj kod nekoj drugoj osobi preko kojeg se ta osoba može pridružiti
- Opcija za brzo spajanje u neku sobu (Quick join)
- Kreiranje privatne sobe koja neće biti vidljiva u pregledniku već se toj sobi može samo pridružiti s kodom
- Mogućnost kreiranja sobe s nekog servera te korištenje tog servera za upravljanje i ograničavanje pristupa
- Mogućnost traženja soba s određenim filterima (npr. način igre, težina igre, itd)

Soba može biti aktivna tijekom igre kako bi se osigurao mehanizam za korisnike da se ponovno mogu pridružiti postojećoj igri ili olakšati migraciju hosta nakon neočekivanog prekida veze.

6.1. Implementacija Lobby servisa

Za korištenje Lobby servisa potrebna je autentifikacija korisnika koristeći Unity autentifikator koji igračima daje identitet. Autentifikator pruža i anonimno rješenje za autentikaciju, anonimni pristup ne zahtjeva da igrač unese podatke za prijavu ili stvori profil. Za ovaj projekt je korišteno anonimno rješenje.

Cijela implementacija Lobby servisa se nalazi u LobbySystem skripti. Slika 8 prikazuje kod koji konfigurira funkcionalnosti korisničkog sučelja. Prilikom pokretanja, postavlja akcije na gumbe koji omogućuju stvaranje, pridruživanje i upravljanje sobama, kao i akciju za pokretanje igre. Također inicijalizira Unity usluge i anonimnu prijavu korisnika.

```
private async void Start()
{

    createLobby.onClick.AddListener(CreateLobby);

    listLobbies.onClick.AddListener(() =>
    {
        ListLobbies();
    });
    joinLobby.onClick.AddListener(() =>
    {
        JoinLobbyByCode(lobbyCode.text);
    });
    privateLobby.onClick.AddListener(() =>
    {
        PrivateLobby();
    });
    quickJoin.onClick.AddListener(() =>
    {
        QuickJoinLobby();
    });
    leaveLobby.onClick.AddListener(() =>
    {
        LeaveLobby();
    });

    startGameButton.onClick.AddListener(() =>
    {
        StartGame();
    });

    await UnityServices.InitializeAsync();

    await AuthenticationService.Instance.SignInAnonymouslyAsync();
}
```

Slika 8. Start funkcija iz LobbySystem skripte

Slika 9 prikazuje stvaranje igračeve sobe. Provjerava uneseno ime sobe i maksimalan broj igrača koji je limitiran slajderom od 1-4. Zatim stavlja opcije za sobu, provjerava je li igrač kojim slučajem u nekoj drugoj sobi te po potrebi izlazi iz te sobe te kreira novu. CreatePlayerSlots funkcija pravi sva mjesta za igrače koji mogu stati u sobu. Nakon toga poziva dvije ponavljajuće funkcije HeartBeat i UpdateLobby, HeartBeat funkcija je tu da se soba održi živom jer ako se u sobi ništa ne mijenja, nakon nekog vremena soba se ugasi. To vrijeme je konstanta od 30 sekundi zato HeartBeat funkcija šalje impuls svakih nekoliko sekundi, u ovom slučaju to je 20, da se soba ne ugasi. Dok UpdateLobby funkcija dohvaća promjene svakih 2 sekunde i stavlja ih na instancu lokalnog igrača. Te dvije funkcije su potrebne jer Lobby servis nema provjeru u stvarnom vremenu.

```
private async void CreateLobby()
{
    try
    {

        string lobbyName;
        if (lobbyNameField.text != "")
            lobbyName = lobbyNameField.text;
        else
            lobbyName = "My lobby";

        int maxPlayers = (int)maxPlayersSlider.value;
        CreateLobbyOptions lobbyOptions = new CreateLobbyOptions
        {
            IsPrivate = privateLobbyBool,
            Player = GetPlayer(),
            Data = new Dictionary<string, DataObject>{
                {"StartGameCode", new DataObject(DataObject.VisibilityOptions.Member, "0")}
            }
        };

        try
        {
            LeaveLobby();
        }
        catch (System.Exception)
        {
            Debug.Log("Either there is no lobby that has been jooined or this is the first create");
            throw;
        }

        hostLobby = await LobbyService.Instance.CreateLobbyAsync(lobbyName, maxPlayers, lobbyOptions);
        joinedLobby = hostLobby;

        CreatePlayerSlots(joinedLobby.MaxPlayers);

        Debug.Log("Created lobby = " + hostLobby.Name + " " + hostLobby.MaxPlayers + " " + hostLobby.Id + " " + hostLobby.LobbyCode);
        PrintPlayers(hostLobby);
        CancelInvoke();
        InvokeRepeating("HeartBeat", heartbeatTime, heartbeatTime);
        InvokeRepeating("UpdateLobby", 0.2f, lobbyUpdateTime);

        LobbyName();
        MenuManager.Instance.ActivateLobby();
        Debug.Log("Created lobby");
    }
    catch (LobbyServiceException e)
    {
        popUpMessage.MakePopUpMessage("Lobby creation failed.", 2);
        Debug.Log(e);
    }
}
```

Slika 9. CreateLobby funkcija iz LobbySystem skripte

ListLobbies funkcija prikazana na slici 10 uzima 10 soba uz filtere da soba nije puna. Nakon što dobije sve potrebne sobe provjerava veličinu dobivenih podataka. Ako nema nikakvih soba igrač dobiva notifikaciju da nema aktivnih soba. Ako lista ima već postojeće sobe, one se brišu te se kreiraju novete se zatim kreiraju novi slotovi za sve nađene sobe.

LobbySystem ima tri funkcije za pridruživanje sobama, to su JoinLobbyByCode, JoinLobbyById i QuickJoin. Sve tri funkcije rade na sličan način. JoinLobbyByCode, prikazana na slici 11, koristi se prilikom pridruživanja privatnim sobama uz kod kojeg igrač dobije od nekog drugog igrača, JoinLobbyById koristi se prilikom pridruživanjima sobama s liste lobbyja dobivene iz ListLobbies funkcije, a QuickJoin se pokušava pridružiti nekoj sobi koja nije privatna niti puna.

```
private async void ListLobbies()
{
    try
    {
        QueryLobbiesOptions queryLobbiesOptions = new QueryLobbiesOptions
        {
            Count = 10,
            Filters = new List<QueryFilter>{
                new QueryFilter(QueryFilter.FieldOptions.AvailableSlots, "0",QueryFilter.OpOptions.GT)
            },
            Order = new List<QueryOrder>{
                new QueryOrder (false, QueryOrder.FieldOptions.Name)
            }
        };
        QueryResponse queryResponse = await Lobbies.Instance.QueryLobbiesAsync();

        Debug.Log("Lobbies found: " + queryResponse.Results.Count);
        if (queryResponse.Results.Count == 0)
        {
            popUpMessage.MakePopUpMessage("No lobbies found.", 2);
        }

        //brise listu ako vec postoji
        if (allLobbiesListed.Count > 0)
        {
            for (int i = allLobbiesListed.Count - 1; i >= 0; i--)
            {
                Destroy(allLobbiesListed[i]);
            }
            allLobbiesListed.Clear();
        }

        foreach (Lobby lobby in queryResponse.Results)
        {
            LobbyInfo newLobbyInstance = Instantiate(lobbyListPrefab, lobbyListContent.transform.position, Quaternion.identity,
            newLobbyInstance.SetGameName(lobby.Name);
            newLobbyInstance.SetLobbyID(lobby.Id);
            Debug.Log(lobby.Id);
            newLobbyInstance.SetPlayerCount(lobby.Players.Count, lobby.MaxPlayers);
            allLobbiesListed.Add(newLobbyInstance.gameObject);
        }
    }
    catch (LobbyServiceException e)
    {
        popUpMessage.MakePopUpMessage("Cant list lobbies. Check internet connection.", 3f);
        Debug.Log(e);
    }
}
```

Slika 10. ListLobbies funkcija iz LobbySystem skripte

```
private async void JoinLobbyByCode(string lobbyCode)
{
    if (lobbyCode == "")
    {
        popUpMessage.MakePopUpMessage("Code to join cannot be empty", 2f);
        return;
    }
    try
    {
        JoinLobbyByCodeOptions joinLobbyByCodeOptions = new JoinLobbyByCodeOptions
        {
            Player = GetPlayer()
        };
        joinedLobby = await Lobbies.Instance.JoinLobbyByCodeAsync(lobbyCode, joinLobbyByCodeOptions);
        CreatePlayerSlots(joinedLobby.MaxPlayers);
        CancelInvoke();
        InvokeRepeating("UpdateLobby", 0.2f, lobbyUpdateTime);
        PrintPlayers(joinedLobby);
        LobbyName();
        MenuManager.Instance.ActivateLobby();
    }
    catch (LobbyServiceException e)
    {
        popUpMessage.MakePopUpMessage("Incorrect lobby code.", 2f);
        Debug.Log(e);
    }
}
```

Slika 11. JoinLobbyByCode funkcija iz LobbySystem skripte

Host može izbaciti bilo kojeg igrača iz sobe koriseći KickPlayer funkciju prikazanu na slici 12.

```
public async void KickPlayer(string id)
{
    try
    {
        await LobbyService.Instance.RemovePlayerAsync(joinedLobby.Id, id);
        Debug.Log("Lobby left successfully");
    }
    catch (LobbyServiceException e)
    {
        Debug.Log("Leaving lobby unsuccessfull");
        Debug.Log(e);
    }
}
```

Slika 12. KickPlayer funkcija iz LobbySystem skripte

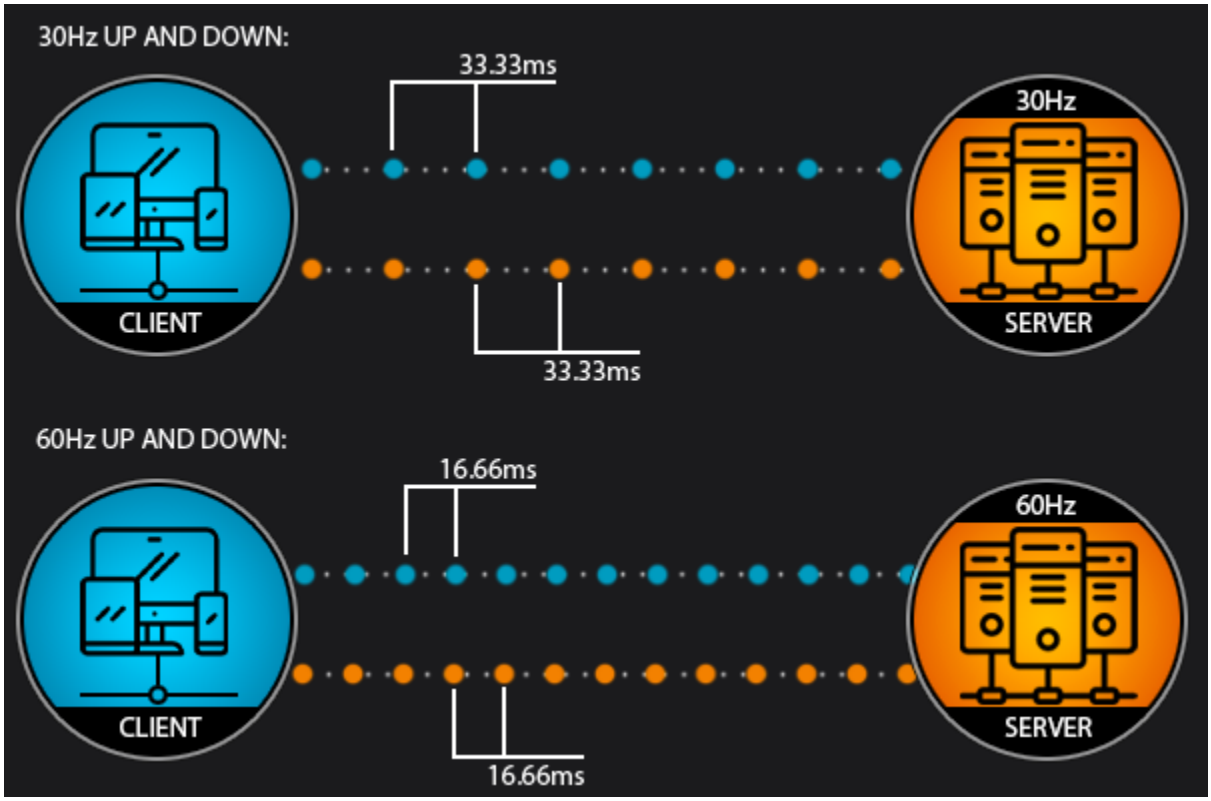
7. Unity Netcode for GameObjects (NGO)

Netcode for GameObjects (NGO) je visokorazinska biblioteka za umrežavanje izgrađena za Unity, koja omogućava apstrakciju logike umrežavanja. Omogućava slanje GameObjecta i globalnih podataka preko mrežne sesije svim igračima istovremeno. Pomoću NGO-a developeri se mogu fokusirati na izgradnju svoje igre umjesto na razne protokole i okvire za umrežavanje [8].

7.1. Stope ažuriranja

Svaki igrač je pod utjecajem određene količine latencije, koja proizlazi iz udaljenosti od servera ili hosta, broj skokova koje paket mora napraviti. Ovdje dolazi do izražaja upravljanje mrežnom latencijom. Zbog ovih kašnjenja igrač bi vidio drugog igrača gdje je bio prije nekoliko milisekundi, a ne gdje se trenutno nalazi. Ako se igra barem djelomično ne bi kompenzirala za ovo kašnjenje, igrači ne bi mogli pogoditi neprijatelje [9].

Ono što dodaje dodatno kašnjenje uz putovanje naših podataka (ping) je koliko često igra šalje i prima te podatke. Kada igra šalje i prima ažuriranja pri 30 Hz (30 ažuriranja u sekundi), tada između ažuriranja ima više vremena nego kada šalje i prima ažuriranja pri 60Hz [9]. Slika 13 prikazuje razliku vremena ažuriranja između 30Hz i 60Hz.



Slika 13. Prikaz vremena ažuriranja u ovisnosti stope ažuriranja [9]

Niske stope ažuriranja ne utječu samo na mrežno kašnjenje. Mogu i uzrokovati probleme poput “super metaka”, gdje jedan pogodak iz oružja nanosi više štete nego što bi trebao.

Dvije opcije rješavanja problema latencije kod igara su server autoritativne igre i klijent autoritativne igre. Svaka od njih ima svoje prednosti i mane.

7.1.1. Server authoritative igre

U server autoritativnim igrama, server donosi konačne odluke o igrivosti nad objektima. Sve odluke o igrivosti se izvršavaju putem servera [9].

Prednosti:

- Dosljednost svijeta: server donosi sve odluke, osiguravajući da se sve odluke donose istovremeno.
- Sigurnost: važni podaci poput zdravlja lika ili položaja mogu biti kontrolirani isključivo na serveru, sprječavajući varalice da manipuliraju tim podacima.

Problemi:

- Reaktivnost: igrači moraju čekati da server potvrdi promjene, što može uzrokovati kašnjenje pri prikazu efekata, kretanju samih igrača ili kreiranja novih objekata.

7.1.2. Klijent autoritativna igra

U klijent autoritativnim igrama, igrači donose konačne odluke o igrivosti. Iako postoji server za dijeljenje informacija, svaki klijent upravlja svojom igrom [9].
Prednosti:

- Reaktivnost: igrači mogu donositi brze odluke, što je korisno kad vjerujemo igračima ili uređajima.

Problemi:

- Dosljednost svijeta: postoji rizik da desinkronizacija, gdje odluke donesene na različitim klijentima mogu rezultirati neskladom u svijetu igre.
- Sigurnost: klijenti mogu pokušati varati, pa je potrebno implementirati dodatne mjere zaštite

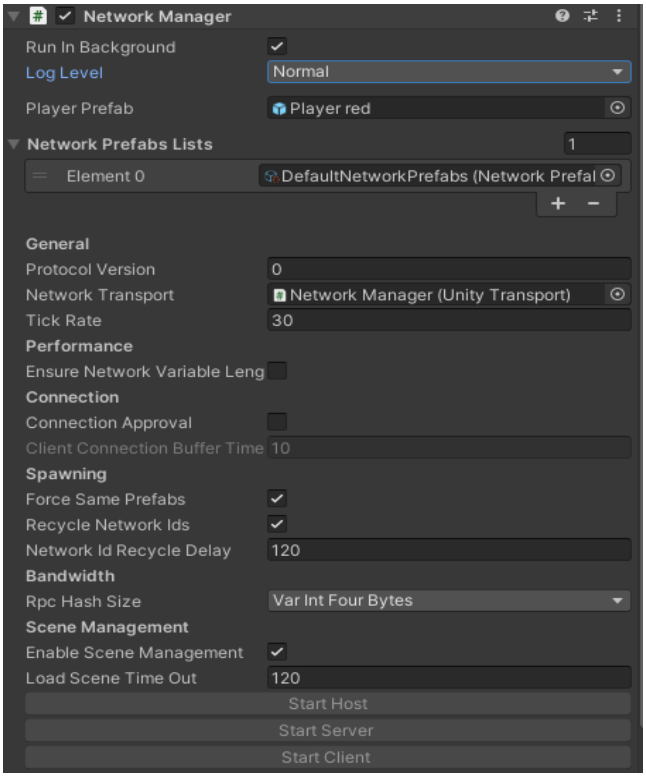
Kada se više klijenata istodobno poveže sa zajedničkim objektom i pokuša utjecati na njega, situacija brzo može postati komplicirana. Kako bi se to izbjeglo, preporučuje se korištenje autoriteta vlasnika klijenta, što omogućava samom vlasniku objekta da s njime reagira. Budući da Netcode kontrolira vlasništvo preko servera, nema mogućnosti da se dva klijenta nađu u utrci za stanje. Da bi se omogućilo dvojici klijenata da utječu na isti objekt moraju zatražiti to od servera, čekati odgovor, te tada provesti željenu logiku.

7.2. Network Manager

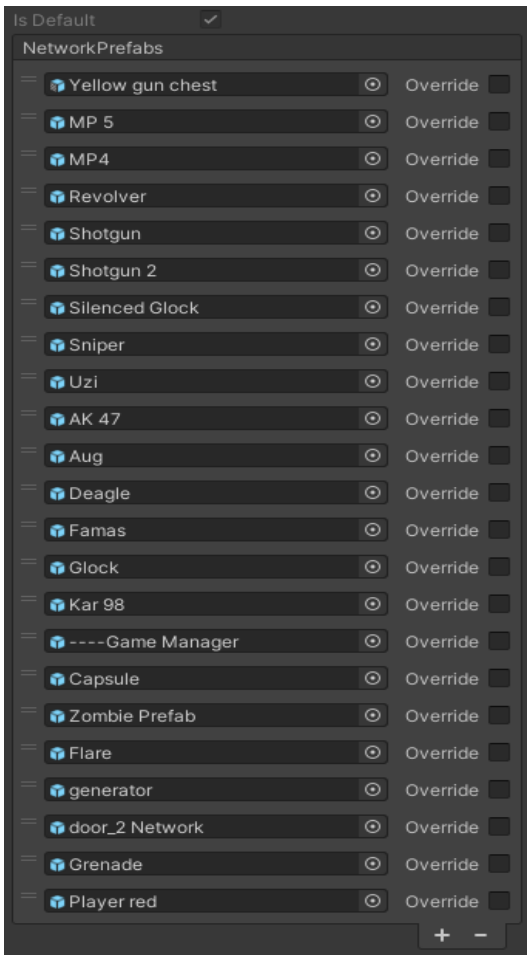
Network Manager, prikazan na slici 14, potrebna je komponenta za Netcode for GameObjects jer sadrži sve mrežne opcije. Mora biti aktivan u sceni igre kako bi NGO radio.

Za ovaj projekt potrebne su Network Prefabs Lists, Tick Rate i Enable Scene Management mrežne opcije. Tick Rate (stopa ažuriranja) je objašnjen u cjelini 7.1. dok će sada biti malo objašnjeni Network Prefabs Lists i Enable Scene Management.

Network Prefabs Lists je lista svih objekata koji se mogu slati preko veze. To omogućuje kreiranje i brisanje objekata po potrebi. U ovom projektu ta lista je prikazana na slici 15.



Slika 14. Prikaz NetworkManager komponente



Slika 15. Prikaz svih Network objekata

Svaki od tih objekata mora na sebi sadržavati Network Object komponentu i, po potrebi, Network Transform ili Client Network Transform komponentu.

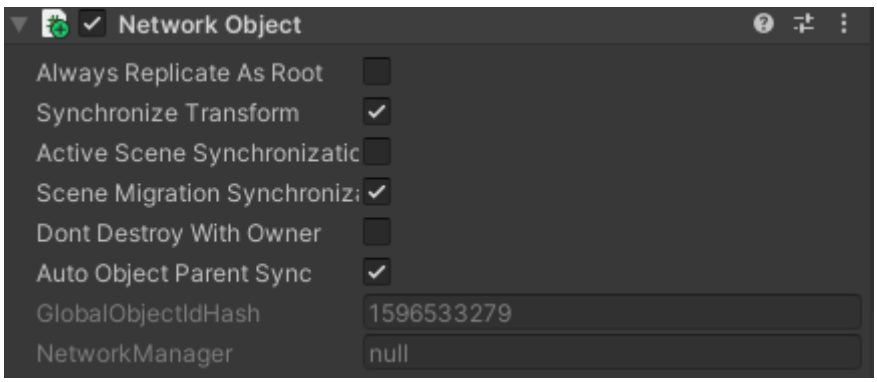
Ako je Enable Scene Management uključen, NGO će se brinuti o upravljanju scenama i sinkronizaciji klijenata. Kada je isključeno, developeri moraju sami stvoriti svoje skripte za upravljanje scenama i brinuti se o sinkronizaciji klijenata [4].

7.3. NetworkObject

Da bi se replicirale osobine svjesne Netcode-a ili poslali RPC-ovi, objekt mora imati NetworkObject komponentu, prikazanu na slici 16, i barem jednu NetworkBehaviour komponentu. Bilo koja Netcode povezana komponenta, poput NetworkTransforma ili NetworkBehavioura s NetworkVariablima ili RPC-ovima, zahtjeva NetworkObjekt komponentu na istom objektu (ili njegovom roditelju).

Prilikom stvaranja NetworkObjecta, NetworkObject.GlobalObjectIdHash identificira mrežni prefab koji će klijent instancirati kako bi stvorio lokalne kopije. Nakon instanciranja, svaki NetworkObject dobiva NetworkObjectId koji se koristi za identifikaciju objekta preko mreže. Na primjer, igrač može reći “Pošaljite ovaj RPC objekt s NetworkObjectIdom 103” i svi znaju na koji objekt se referira. NetworkObject se stvara na klijentu kada se instancira i dodjeli mu se jedinstveni NetworkObjectId [5].

Objekti u igri mogu biti posjedom servera (što je postavljeno kao zadano) ili se mogu dati u posjed nekom od klijenata. Netcode for GameObjects ima server- autoritativnu strukturu, što znači da server ima kontrolu nad procesima stvaranja i uništavanja NetworkObjecta i samo je on ovlašten za to. Ako igrač želi uništiti neki objekt onda mora poslati RPC poziv na server [5].



Slika 16. Prikaz NetworkObject komponente na objektu

7.4. NetworkBehaviour

NetworkBehaviour zamjenjuje standardni MonoBehaviour i dodaje dodatne opcije potrebne za mrežu. Jedna je od ključnih komponenta u Netcode for GameObjects okruženju koja omogućuje sinkronizaciju stanja i slanje poruka preko mreže. Da bi se slale varijable ili RPC-jevi putem mreže, objekt u igri mora imati komponentu NetworkObject i barem jednu komponentu NetworkBehaviour. Ako se na objekt dodjeli samo NetworkBehaviour komponenta Netcode će automatski dodati komponentu NetworkObject na taj objekt.

7.4.1. RPC funkcije i NetworkVariables

NetworkBehaivouri mogu sadržavati RPC funkcije i NetworkVariable. RPC funkcije mogu biti ClientRpc i ServerRpc. Prilikom kreiranja takva funkcija mora imati [ClientRpc] ili [ServerRpc] atribut te moraju imati takav isti sufiks. Kod ServerRpc možemo dodati RequireOwnership varijablu koja označava ima li igrač pravo pozvati tu funkciju ili ju samo server može pozvati.

```
[ServerRpc(RequireOwnership = false)]
1 reference
public void TakeDmgServerRpc(int dmgTaken){
    if(IsServer)
        hp.Value -=dmgTaken;
}
```

Slika 17. TakeDmgServerRpc funkcija iz Stats skripte

Razlika između RPC poziva:

- ClientRpc (Client Remote Procedure Call): Metoda koja se izvršava na klijentskoj strani, ali se pokreće s poslužitelja (servera)
- ServerRpc (Server Remote Procedure Call): Metoda koja se izvršava na serveru i koristi se za komunikaciju od klijenta prema serveru. Server tada može poslati ClientRpc poziv na sve klijente i njima dati nove informacije

NetworkVariable je vrsta varijable koja omogućava vrijednostima da budu sinkronizirane između klijenata i poslužitelja kako bi se održalo dosljedno stanje igre kod svih igrača. Kada se vrijednosti promjene, svi klijenti koji prate tu promjenu bit će obaviješteni o novoj vrijednosti. Prilikom izrade varijable postoje opcije koje omogućuju da se postavi vrijednost varijable, tko može čitati tu varijablu i tko može pisati na tu varijablu. Primjer NetworkVariable koja označava igračeve životne bodove prikazan je slici 18.

```
public NetworkVariable<int> hp = new NetworkVariable<int>(100,
                                                          NetworkVariableReadPermission.Everyone,
                                                          NetworkVariableWritePermission.Server);
```

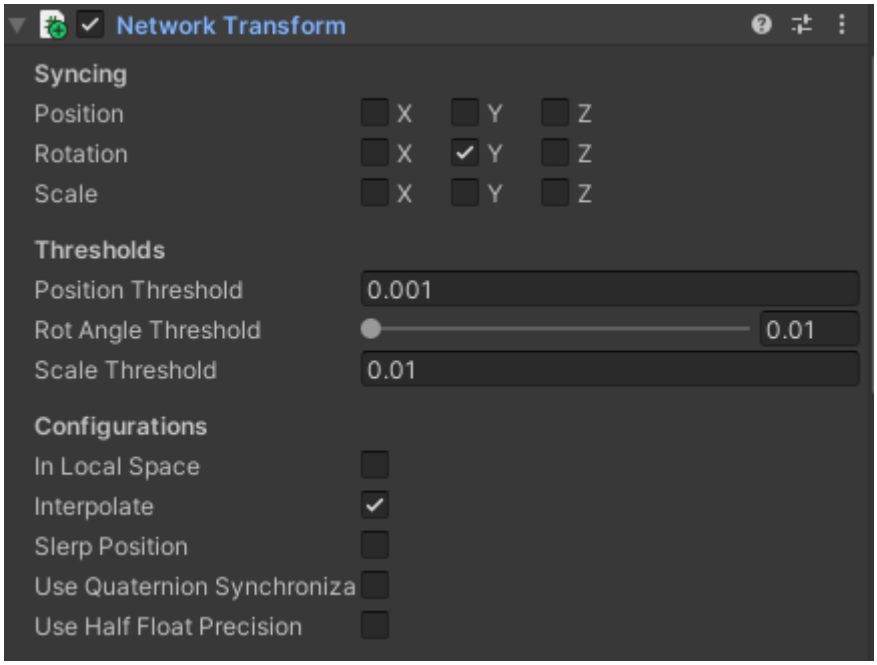
Slika 18. Prikaz NetworkVariable hp iz Stats skripte

7.5. NetworkTransform

Sinkronizacija pozicije, rotacije i veličine objekta jedan je od najčešćih zadataka u današnjim multiplayer igrama. Koncept je jednostavan:

- Određuje se koje se osi žele sinkronizirati
- Vrijednosti se serijaliziraju
- Serijalizirane vrijednosti se šalju kao poruke svim drugim povezanim klijentima
- Poruke se obrađuju i vrijednosti deserijaliziraju
- Deserijalizirane vrijednosti se primjenjuju na odgovarajuće osi pozicije

Zato NGO pruža NetworkTransform komponentu, prikazanu na slici 19, koja rješava aspekt sinkronizacije pozicije, rotacije i veličine objekta.



Slika 19.NetworkTransform komponenta na objektu

Kada se dodaje NetworkTransform komponenta na objekt, njegov objekt ili roditelj mora imati NetworkObject komponentu. NetworkTransform komponenta može se nalaziti na bilo kojoj razini potomaka glavnog objekta. Samo glavni objekt mora imati NetworkObject komponentu.

Prema zadanim postavkama sinkronizacija pozicije objekta radi se u globalnom prostoru. Uključivanjem postavke In Local Space omogućuje se promjena sinkronizacije u lokalni prostor, to jest, u odnosu na roditelja objekta.

Interpolacija (Interpolate) je opcija koja omogućuje glatke prijelaze između ažuriranja transformacije na instancama koje nisu autoritativne. Gleda dolazna stanja i uvodi blagu odgodu između instanci. Ako je ta opcija isključena, može doći do vizualnog treperenja ili prividnog skakanja na novo primijenjena ažurirana stanja kada je latencija visoka.

7.6. NetworkAnimator

Pruža sinkronizaciju animacija tijekom mrežne igre. Stanja animacija sinkroniziraju se s igračima koji se pridružuju postojećoj igri i bilo kojim klijentima koji su već povezani prije promjene stanja animacije.

NetworkAnimator mora biti stavljen na objekte čije animacije želimo da se sinkroniziraju. Sva Animator svojstva (osim okidača), mogu se postavljati direktno preko Animator komponente te će biti sinkronizirana.

7.7. ClientNetwork komponente

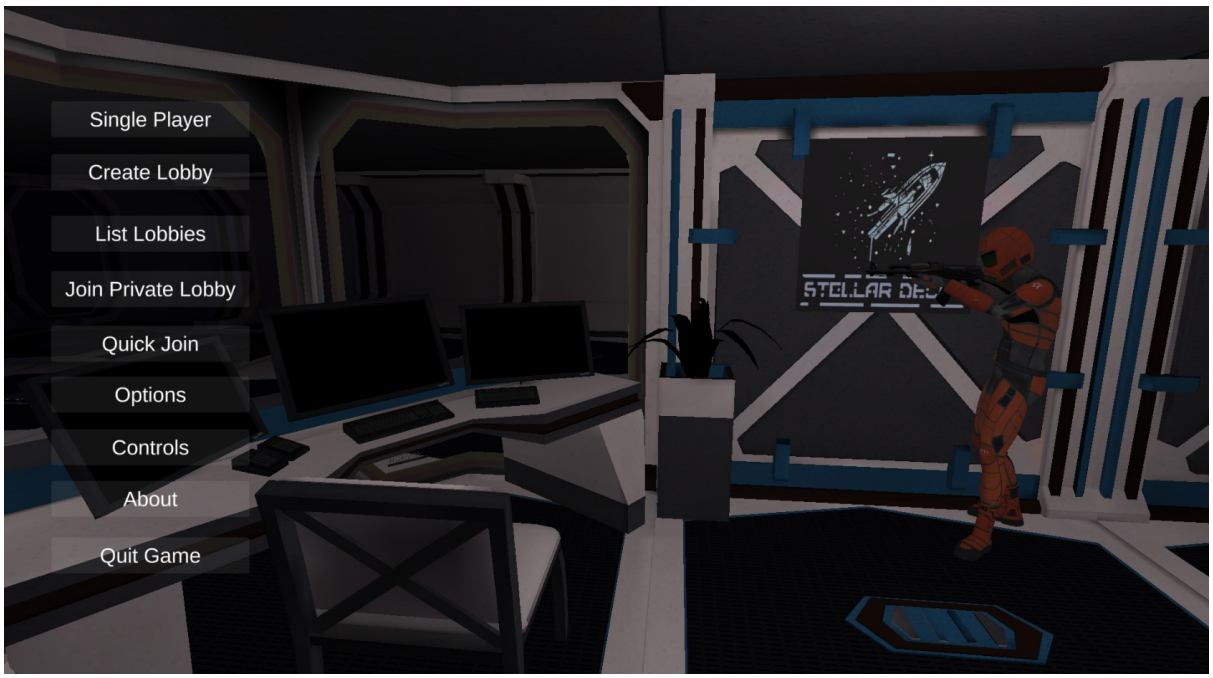
Rade isto što i obične network komponente uz razliku da one imaju autoritet za slanje podataka prema drugim igračima, to jest, ne čekaju dopuštenje servera smiju li nešto napraviti, već naknadno šalju podatke serveru što su točno napravili. Kao što postoje NetworkTransform i NetworkAnimator komponente, tako postoje i ClientTransform i ClientAnimator komponente.

8. Glavni izbornik

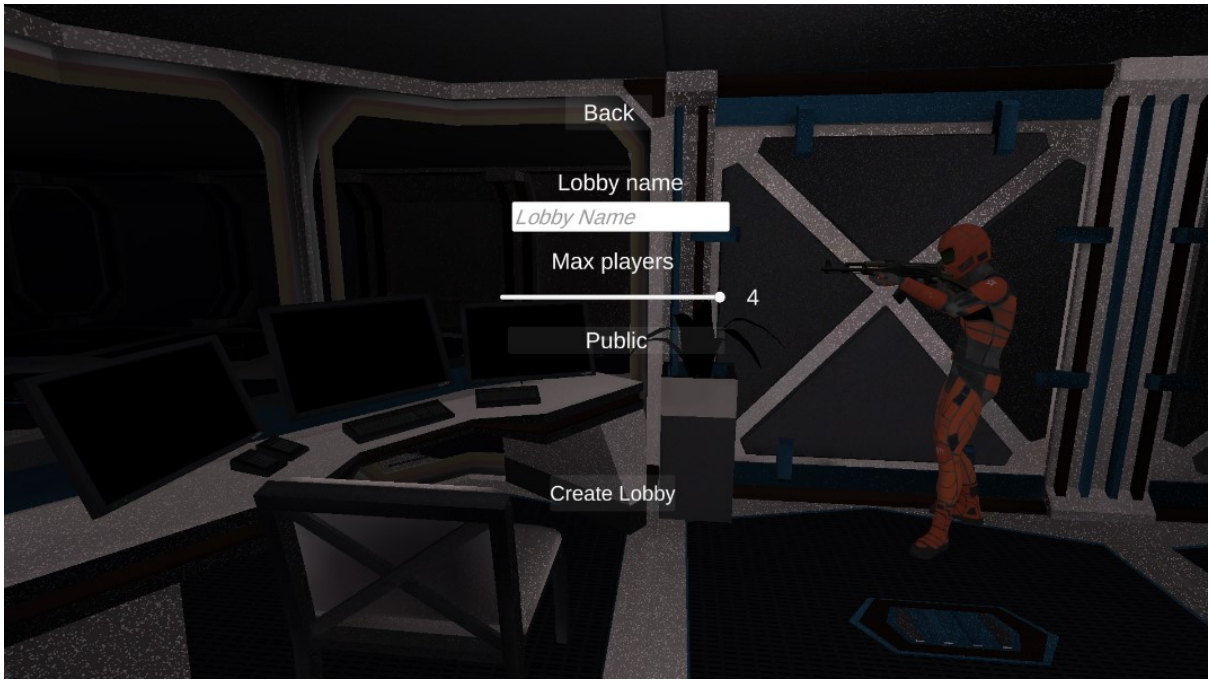
Prije izbornika igraču dolazi upit za unošenje imena prikazan na slici 20. Nakon toga se otvara glavni izbornik prikazan na slici 21.

Slika 20. Prikaz izbornika za upis imena

U glavnom izborniku igrač ima opciju pokrenuti igru, tako da igra sam, kreirati svoju sobu u koju se mogu pridružiti drugi igrači, prikazano na slikama 22 i 23, ili se može pridružiti nekoj od postojećih soba ako pritisne List Lobbies tipku, prikazanu na slici 24. Ako je soba privatna može joj se pridružiti pomoću koda kojeg mu host ili neki drugi igrač iz sobe može dati. Upis koda prikazan je na slici 25. Postoji i mogućnost pronalaska neke slučajne sobe kojoj će se igrač pridružiti pritiskom na Quick Join tipku.



Slika 21. Prikaz glavnog izbornika



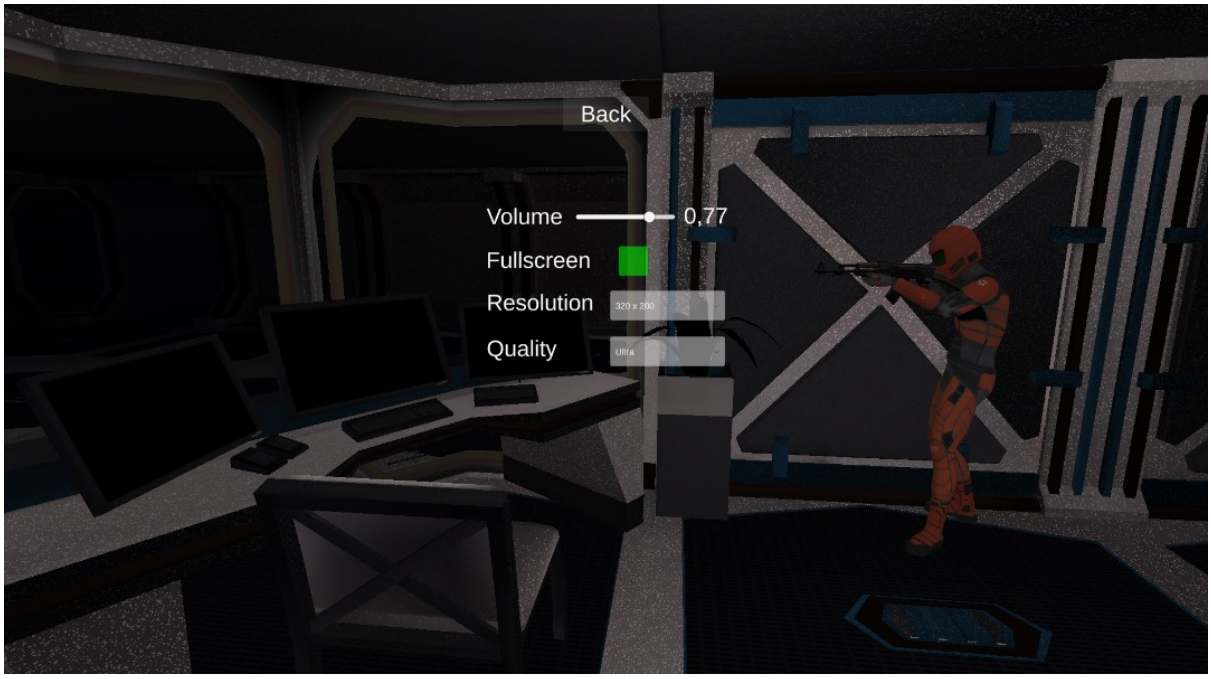
Slika 22. Prikaz kreiranja nove sobe

Slika 23. Prikaz izgleda sobe

Slika 24. Prikaz aktivnih soba

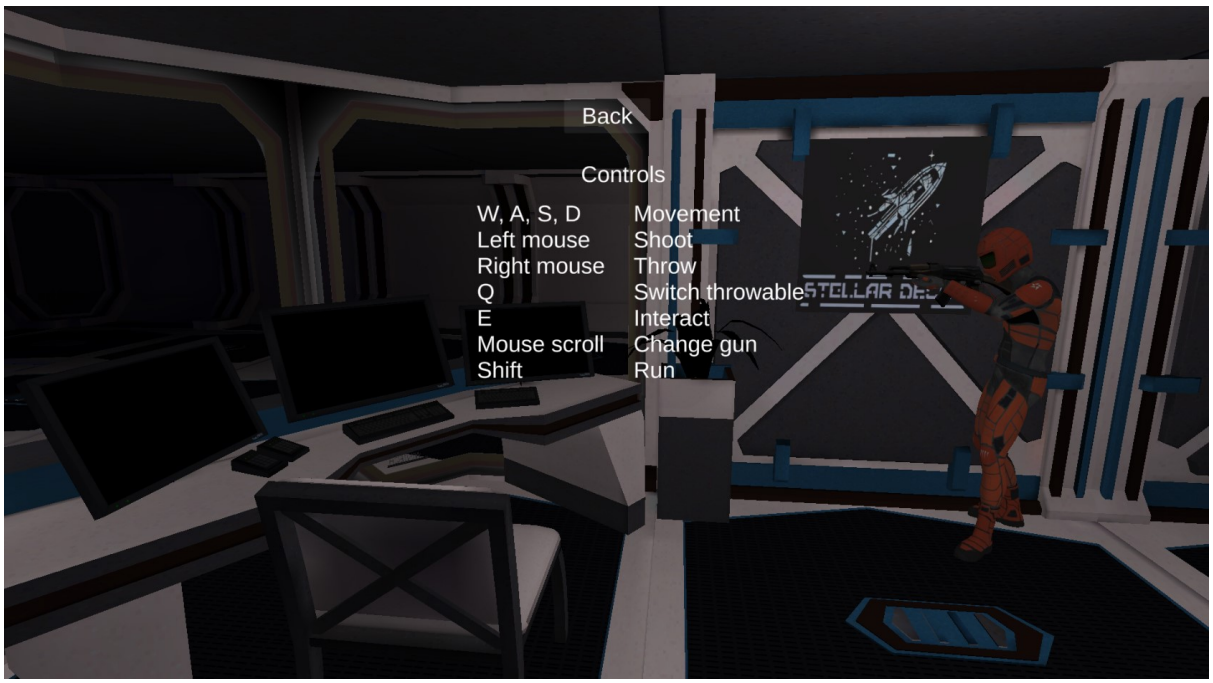
Slika 25. Prikaz upisa koda za pridruživanje privatnoj sobi

Igrač ima mogućnost promijeniti neke opcije igre. Pritiskom na Options tipku otvara se novi izbornik s opcijama prikazanim na slici 26. Igrač ima na raspolaganju opcije poput promjene rezolucije, promjene glasnoće zvuka, promjene punog ekrana ili prozora te promjene grafičke kvalitete igre.



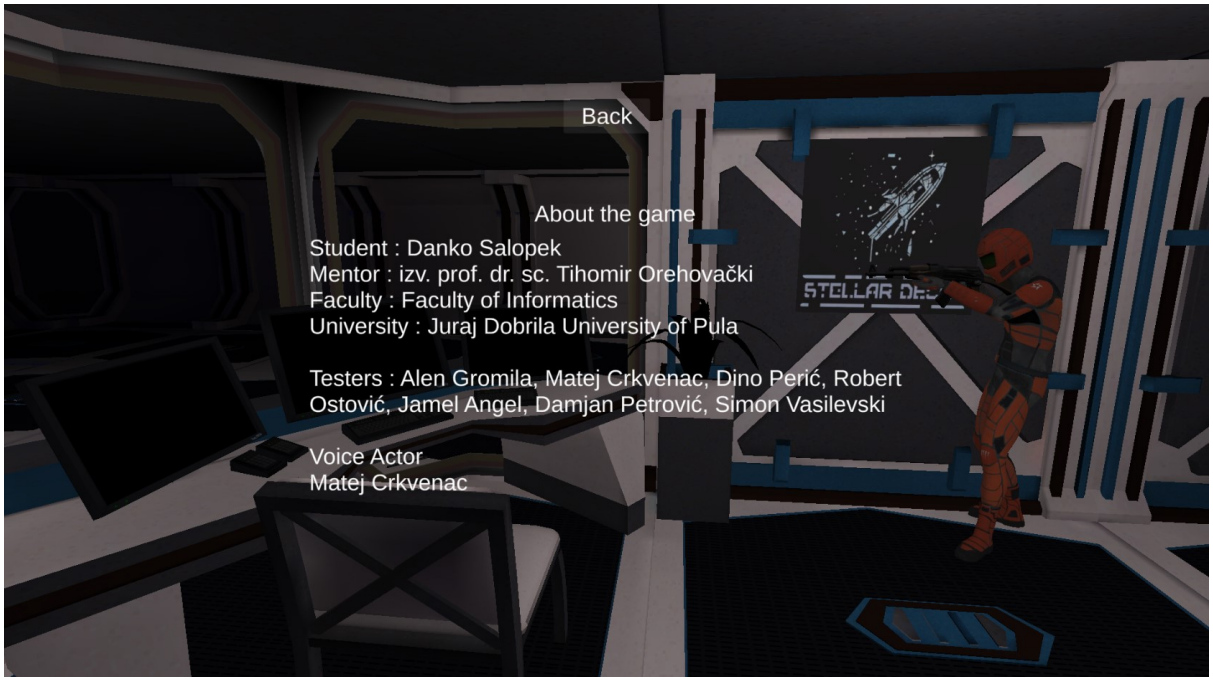
Slika 26. Prikaz izbornika za opcije

Igrač također ima mogućnost da pogleda kontrole potrebne za igranje, prikazane na slici 27.



Slika 27. Prikaz kontrola igrača tijekom igre

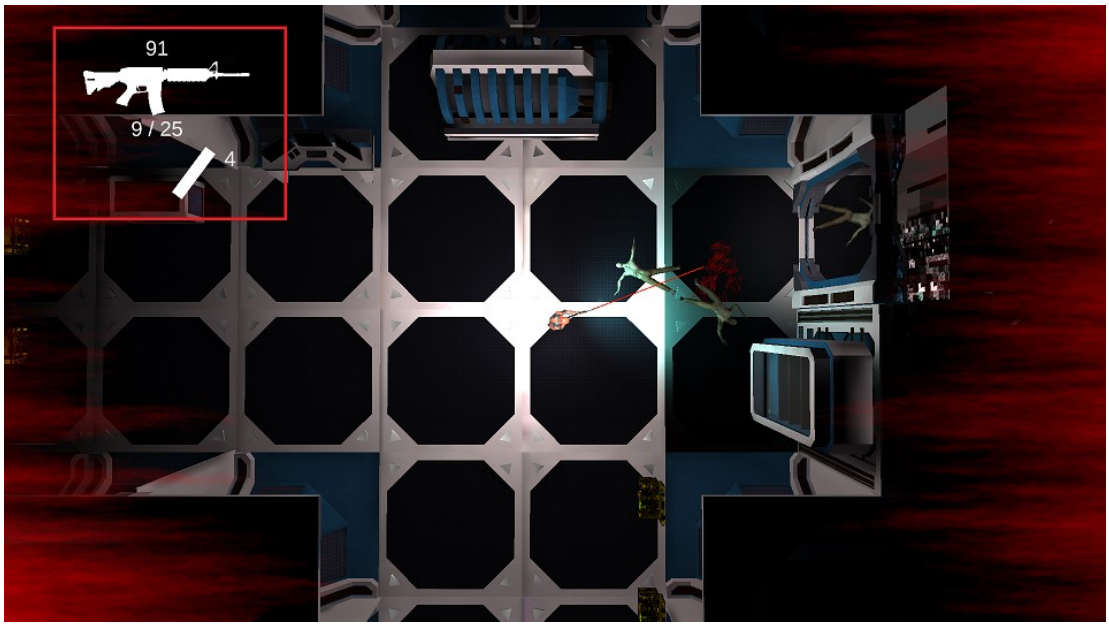
Slika 28 prikazuje informacije o igrici. Prozor s informacijama o igri se može otvoriti pritiskom na About tipku u glavnom izborniku.



Slika 28. Prikaz informacija o igri

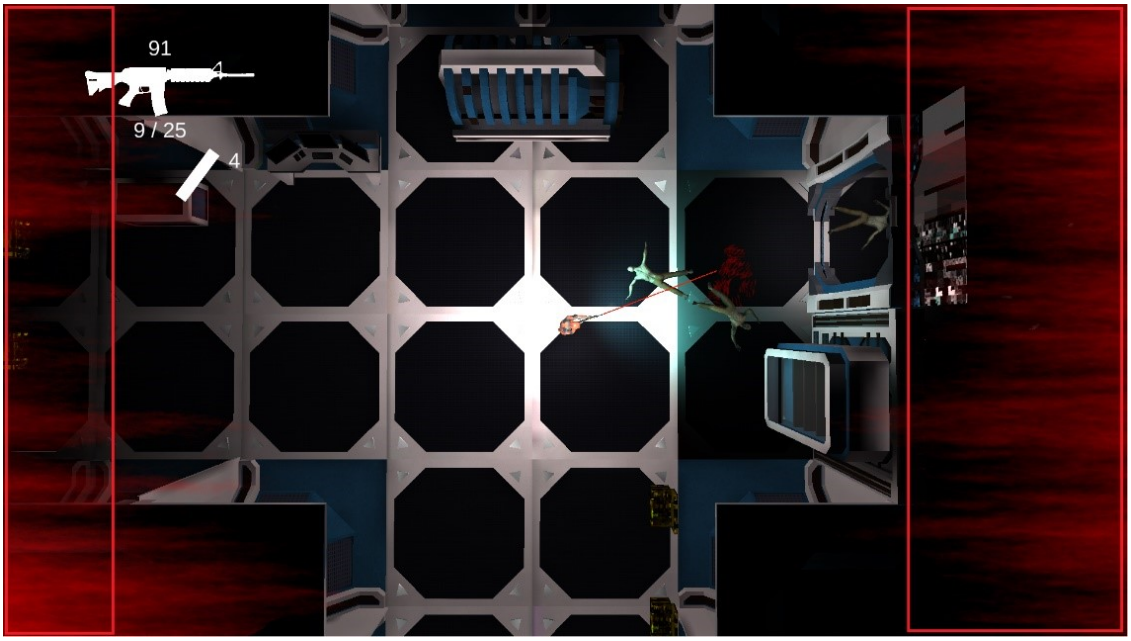
9. Korisničko sučelje u igri

Igra ima minimalističko korisničko sučelje. Na slici 29 označeno je trenutno oružje koje igrač koristi, broj trenutnih metaka u magazinu tog oružja i ukupan broj metaka preostalih za to oružje. Ispod ikone oružja prikazana je ikona baklje i broj koji prikazuje koliko ih igrač ima na raspolaganju. Tijekom igre igrač ima mogućnost promjene predmeta za bacanje te će se ikona ažurirati u ovisnosti koji se predmet za bacanje koristi.



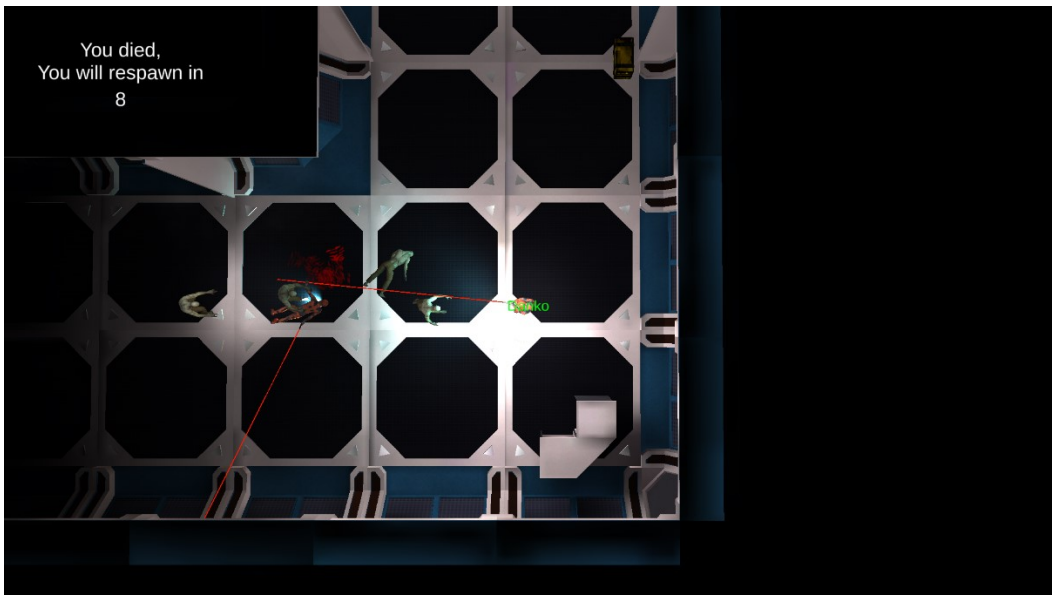
Slika 29. Prikaz korisničkog sučelja u igri

Na slici 30 označene su crvene linije koje reprezentiraju životne bodove igrača, što je ekran crveniji to je igrač bliže smrti. S obzirom na to koliko igrač ima preostalih životnih bodova pokreće se zvuk otkucaja srca koji postaje sve brži, tako da igrač zna da je blizu smrti.

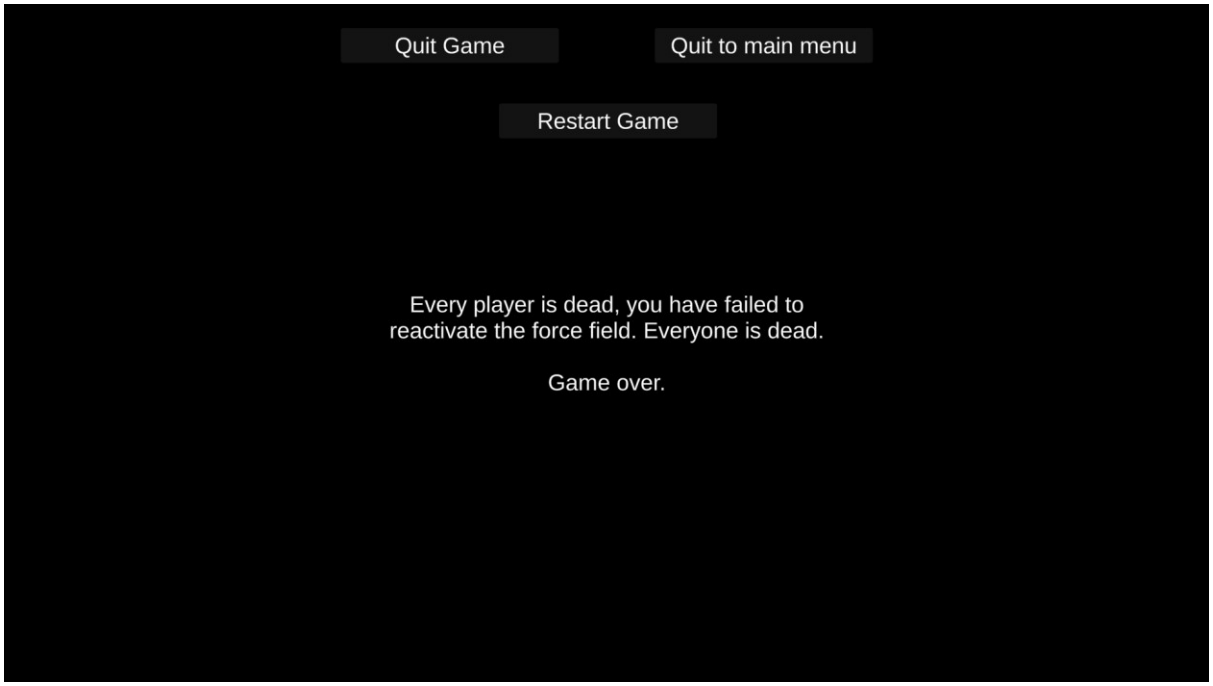


Slika 30. Prikaz životnih bodova igrača

Ako je igrač ubijen onda mu se prikazuje poruka da je mrtav i da će se uskoro ponovno stvoriti, kao što je prikazano na slici 31. To funkcionira samo kod više igrača, ako igrač igra sam onda je za njega igra gotova i pojavljuje mu se izbornik za kraj igre, prikazan na slici 32. Ako je u pitanju igra s više igrača onda će se kamera trenutnog igrača prebaciti i gledati nekog drugog dok se igrač ponovno ne stvori.

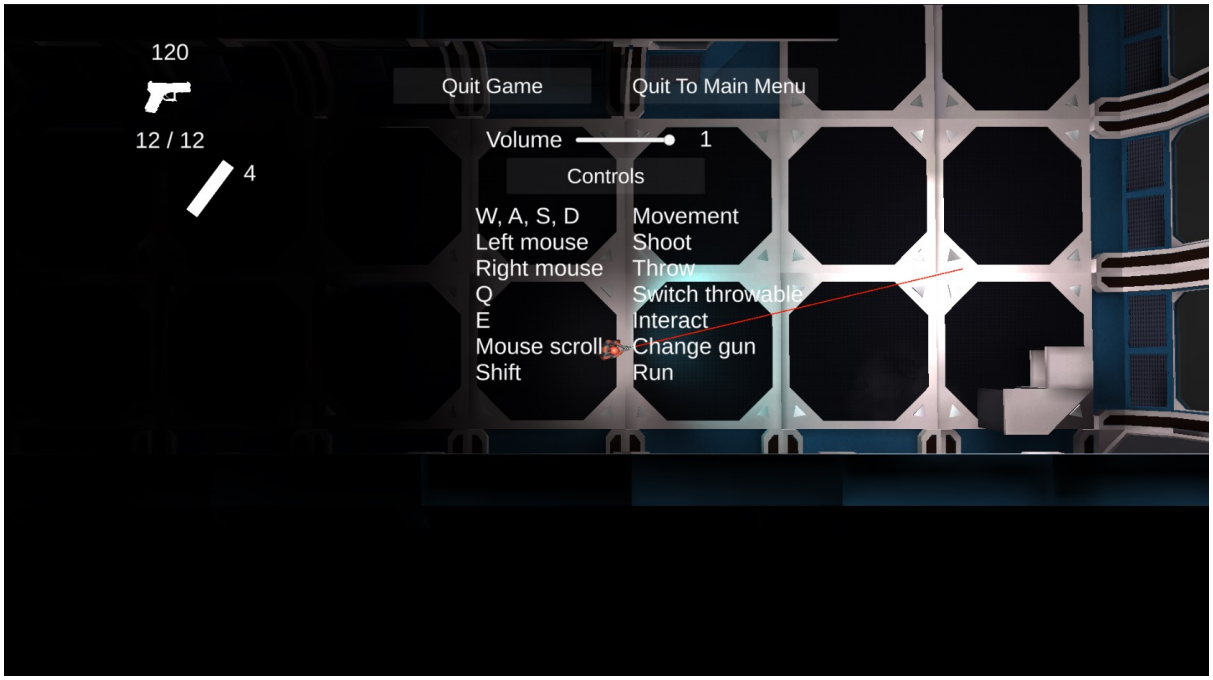


Slika 31. Sučelje nakon smrti igrača



Slika 32. Sučelje nakon smrti svih igrača

Igrač tijekom igre može pritisnuti “ESC” tipku na tipkovnici kako bi mu se otvorio izbornik prikazan na slici 33 gdje se prikazuju opcije za izlazak igre, smanjivanje zvuka u igri i kontrole igre. Pritiskom na Controls tipku igrač može otvoriti i zatvoriti prozor s kontrolama.
Ponovnim pritiskom “ESC” tipke taj se izbornik ugasi.



Slika 33. Prikaz sučelja za opcije tijekom igre

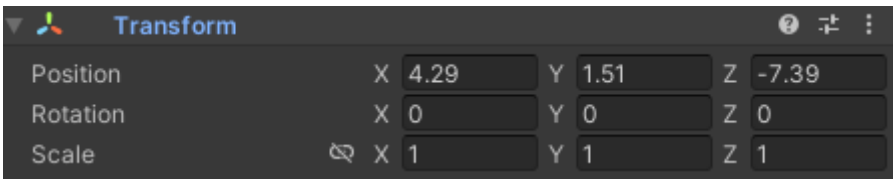
10. Igrač

Player model preuzet je s Unity Asset Stora [20]. Komponente na objektu igrača prikazane su na slici 34.

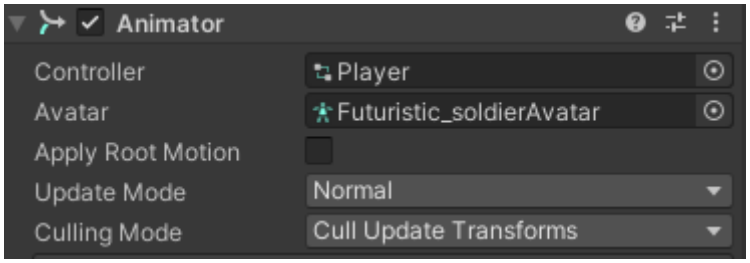
Transform komponenta, prikazana na slici 35, sadrži sve podatke o poziciji, rotaciji i veličini objekta. Svaki objekt ima na sebi transform komponentu.

Animator, prikazan na slici 36, služi za izvođenje animacija koje se nalaze u određenom kontroleru (Controller).

Slika 34. Prikaz svih komponenata na igraču

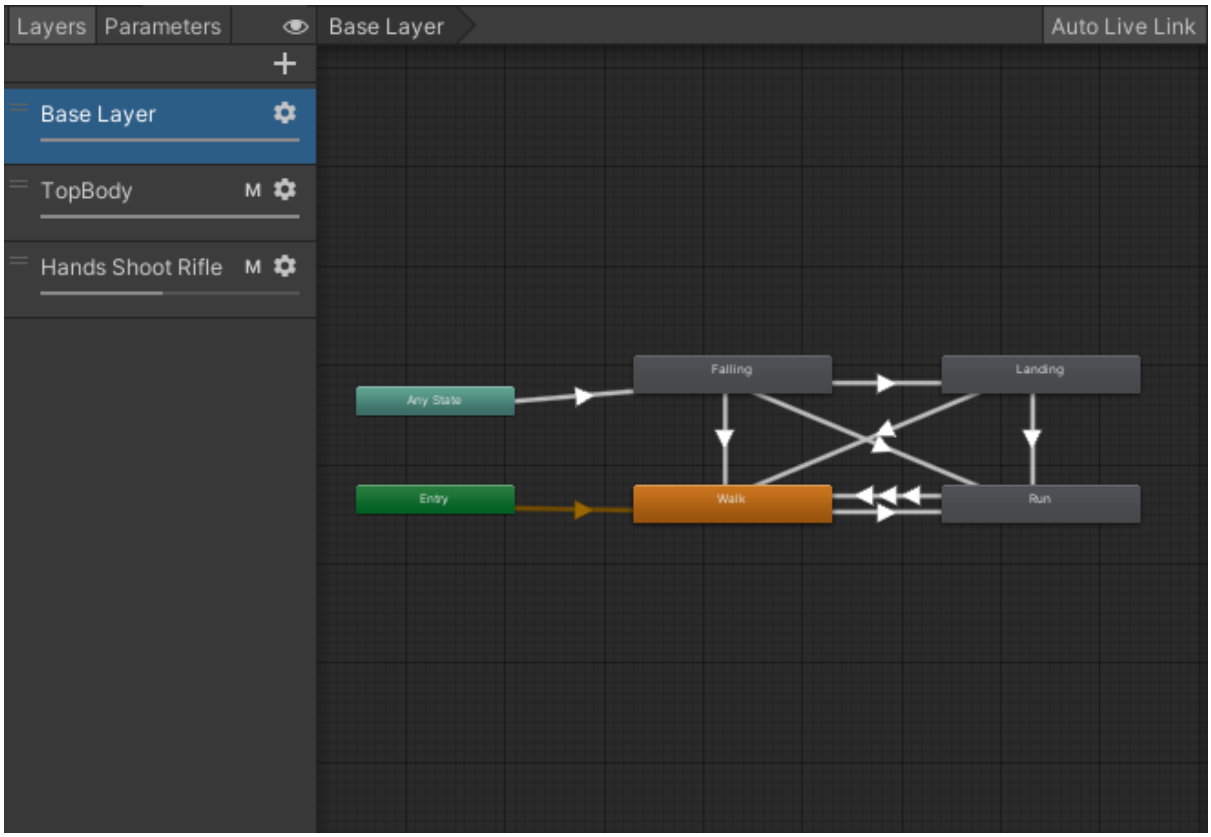


Slika 35. Prikaz Transform komponente na igraču

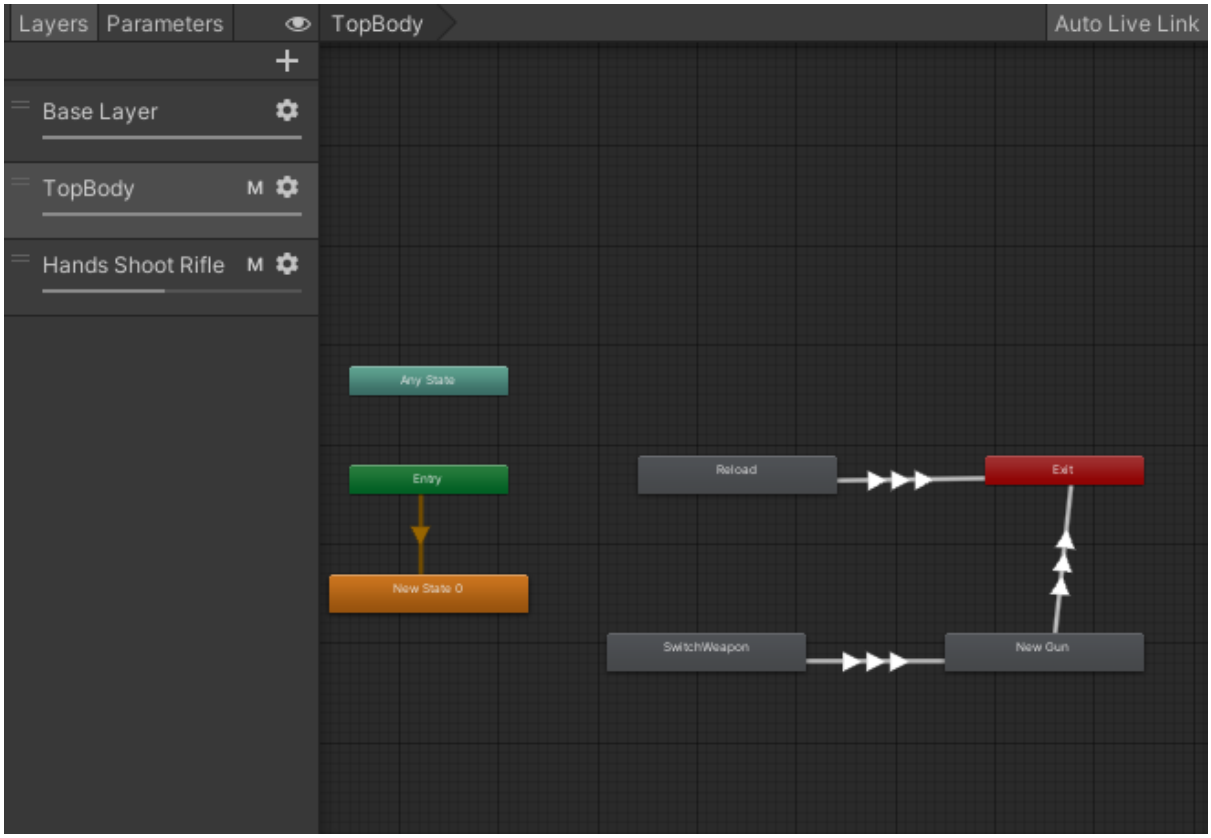


Slika 36. Prikaz Animator komponente na igraču

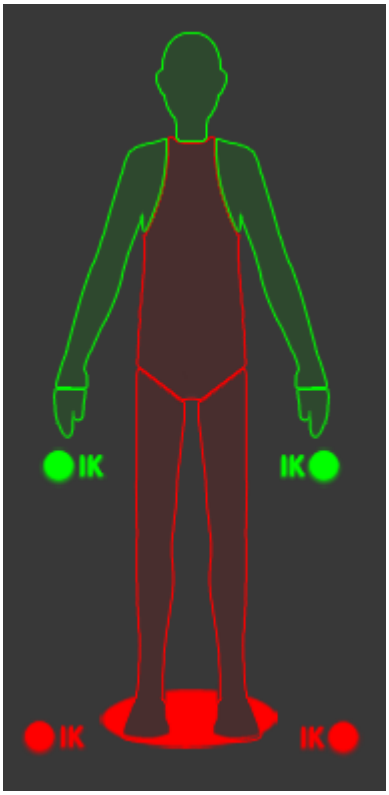
Player kontroler iz Animator komponente je raspodijeljen na tri sloja. Prvi je Base Layer, prikazan na slici 37, koji služi kao generalni sloj svih animacija za trčanje, skakanje i slično. Drugi sloj je Top Body, prikazan na slici 38, koji na sebi ima posebnu masku prikazanu na slici 39 koja limitira utjecaj animacije na pojedine dijelove tijela. Za ovaj sloj potrebne su samo ruke i glava tako da su samo oni označeni. Koristi se za izvođenje generalnih animacija poput promjene oružja i promjene magazina oružja. Slika 40 prikazuje treći sloj pod nazivom Hands Shoot Rifle, koji služi za stvaranje malog trzaja prilikom pucanja puške. Koristi istu masku kao i Top Body.



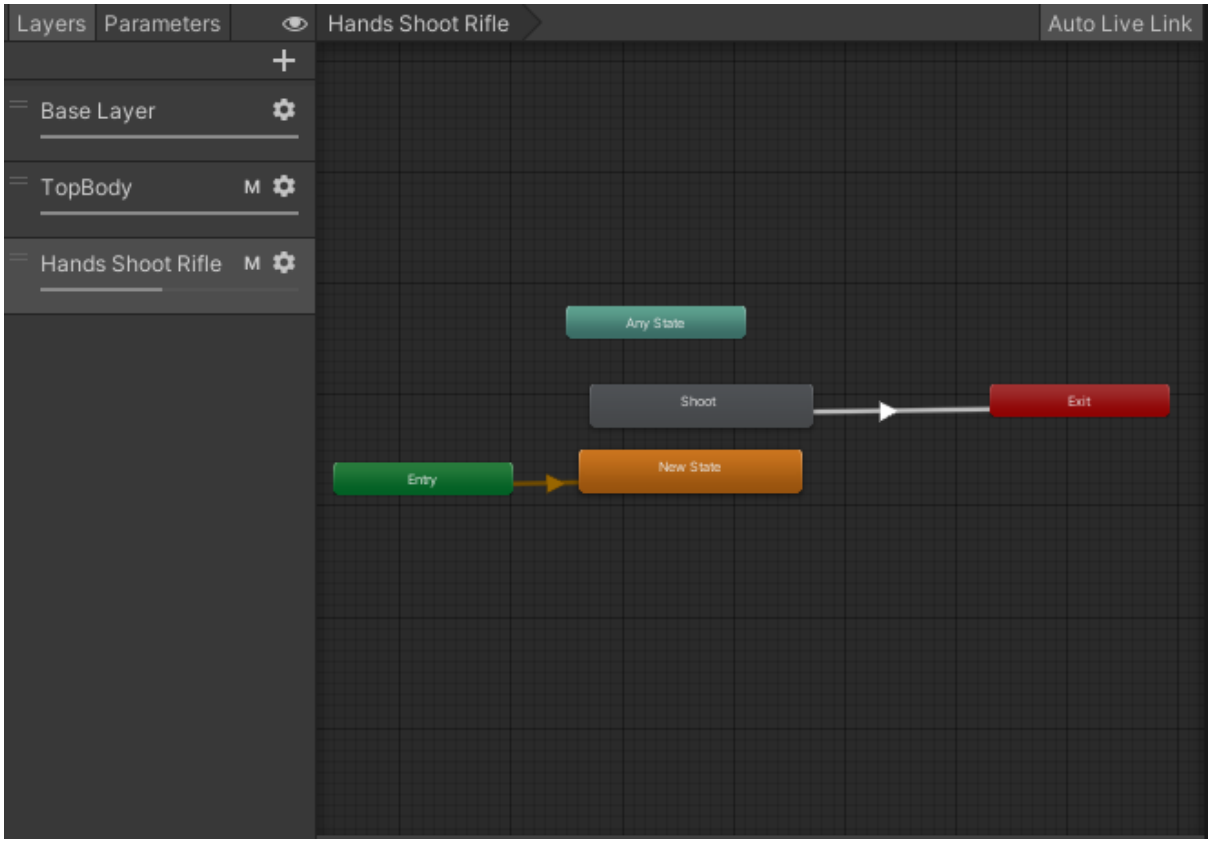
Slika 37. Prikaz Base Layer sloja



Slika 38. Prikaz Top Body Layer sloja

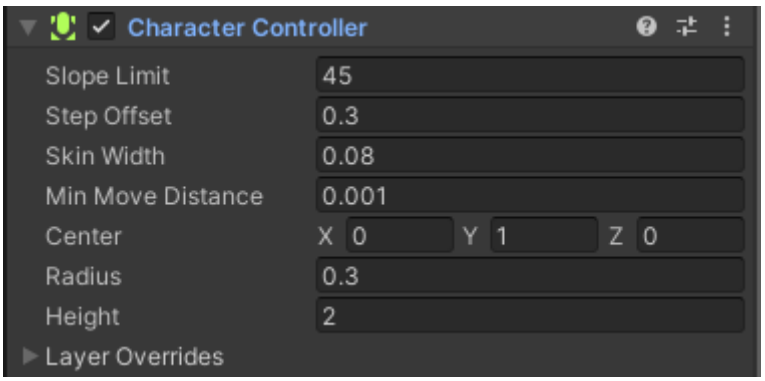


Slika 39. Prikaz maske koja se nalazi na Top Body i Hands Shoot Rifle slojevima



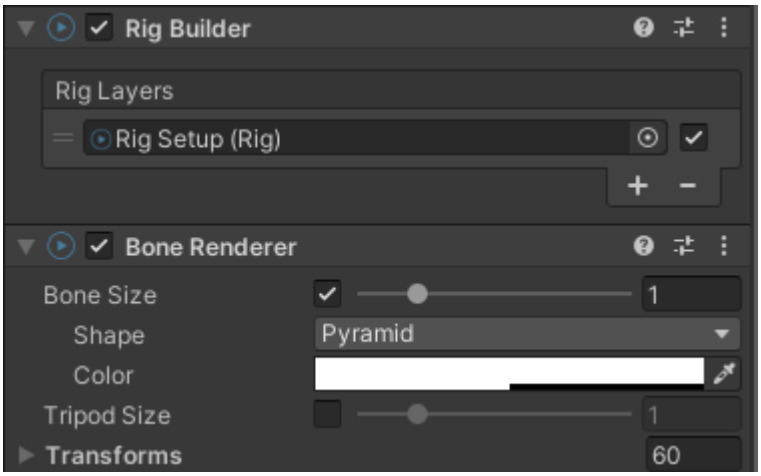
Slika 40. Prikaz Hands Shoot Rifle sloja

Character Controller, prikazan na slici 41, je Unity komponenta koja omogućuje kontroliranje kretanja likova. Omogućuje kretanje po tlu, automatski primjenjuje gravitaciju, detektira kolizije s drugim objektima u sceni i automatski ih rješava.



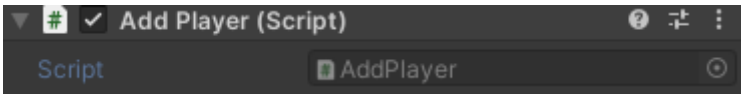
Slika 41. Prikaz Character Controller komponente s Player objekta

Slika 42 prikazuje Rig Builder i Bone Renderer komponente, oni su dijelovi Unityjevog dodatka poznatog kao Animation Rigging sustav. Ovaj sustav omogućuje precizno upravljanje dijelovima likova tijekom animacije. Na primjer, u igri se koristi za ruke igrača, omogućavajući postavljanje različitih pozicija ruku kako bi se pravilno prikazalo pravilno držanje različitih oružja.



Slika 42. Prikaz Rig Builder i Bone Renderer komponenata s Player objekta

AddPlayer skripta postavlja igrača u listu živih igrača kako bi zombiji znali prema kome se mogu kretati. Prikazana je na slici 43.



Slika 43. Prikaz Add Player komponente s Player objekta

10.1. Player skripta

Player skripta upravlja igračevim kretanjem i rotacijom te pokreće animacije koje su vezane uz kretanje.

Prilikom stvaranja igrača Player dohvaća PlayerCameraFollow skriptu i poziva njenu funkciju FollowPlayer, koja postavlja igrača kao cilj praćenja kamere, tako da će kamera uvijek pratiti i gledati igrača.

U funkciji Update, prikazanoj na slici 44 te koja se izvršava svaku sličicu, provjerava se je li trenutni igrač vlasnik ovog objekta. Ako nije vlasnik, onda se funkcija ne izvršava. Ako je vlasnik, provjerava se izvršava li se animacija slijetanja. Zatim se provjerava trči li igrač (RunInputCheck funkcija prikazana na slici 45) i u kojem se smjeru igrač kreće (MovementInput funkcija prikazana na slici 45).

```
protected override void Update()
{
    if(!IsOwner)
        return;
    if( AnimationPlayingCheck("Landing")){
        return;
    }

    RunInputCheck();
    MovementInput();
    Movement();
}
```

Slika 44. Update funkcija iz Player skripte

```
void MovementInput(){
    movement.x = Input.GetAxis("Horizontal");
    movement.z = Input.GetAxis("Vertical");
    relative = transform.InverseTransformDirection(movement);
}


1 reference
void RunInputCheck()
{
    if (Input.GetKey(KeyCode.LeftShift))
    {
        if(run == false){
            run = true;
            anim.SetBool("Running", true);
        }
    }
    else
    {
        if(run ){
            run = false;
            anim.SetBool("Running", false);
        }
    }
}
```

Slika 45. MovementInput i RunInputCheck funkcije iz Player skripte

Slika 46 prikazuje Movement funkciju koja postavlja parametre animatora, te zatim pokreće igrača. Move funkcija, prikazana na slici 47, pokreće igrača te ga zatim rotira prema poziciji miša.

```
protected override void Movement()
{
    SetAnimationParameters();

    Move();

}
```

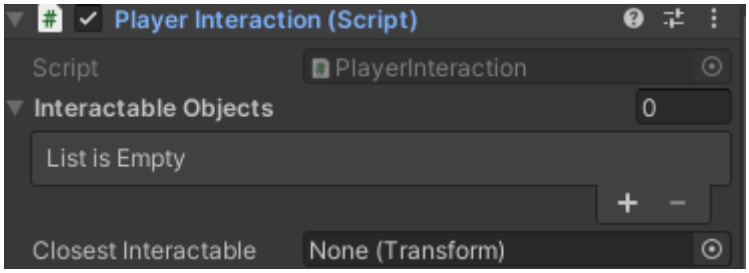
Slika 456. Movement funkcija iz Player skripte

```
void Move(){

    if(!grounded || AnimationPlayingCheck("Falling")){
        fallingTime += Time.deltaTime;
        MoveController(moveSpeedWhileFalling);
        FaceMouse();
        return;
    }

    if(run){
        MoveController(runSpeed);
        if(movement.magnitude <=0.05f)
            FaceMouse();
        else
            FaceDirection();
        return;
    }

    FaceMouse();
    MoveController(moveSpeed);
}
```

Slika 46. Move funkcija iz Player skripte

10.2. PlayerInteraction skripta

Slika 48 prikazuje PlayerInteraction skriptu kao komponentu na Player objektu. Update funkcija iz PlayerInteraction skripte prikazana je na slici 49 te omogućuje igraču interakciju s objektima u okolini. Sadrži listu objekata s kojima igrač može imati interakciju. Ako postoji više objekata, odabrat će objekt najbliži igraču koristeći SelectInteractableObject funkciju. Interakcija se pokreće pritiskom na tipku “E”. Kroz Interact funkciju, prikazanu na slici 50, objekti se dodaju i uklanjaju iz liste, ovisno o tome jesu li dostupni za interakciju ili ne. Interact funkcija poziva se s drugih objekata kojima je igrač došao dovoljno blizu te omogućio interakciju s njima. Također, ova skripta prikazuje sučelje za interakciju te pritom interakcije stvara zvukove.



Slika 47. PlayerInteraction komponenta na Player objektu

```
private void Update()
{
    if (interactableObjects.Count != 0)
    {
        SelectInteractableObject();

        Debug.Log("Closest interactable " + closestInteractable.name);

        if (Input.GetKeyDown(KeyCode.E))
        {
            ButtonClickSFX.Instance.PlayInteractionSound();
            closestInteractable.GetComponent<IInteractable>().Interact(transform);
        }
    }
    else
    {
        closestInteractable = null;
    }
}
```

Slika 48. Update funkcija iz PlayerInteract skripte

```
public void Interact(Transform thisInteracted)
{
    if (interactableObjects.Contains(thisInteracted))
        interactableObjects.Remove(thisInteracted);
    else
        interactableObjects.Add(thisInteracted);
}
```

Slika 49. Interact funkcija iz PlayerInteract skripte

10.3. AudioPlayer skripta

Funkcije AudioPlayer skripte pozivaju se prilikom izvođenja animacija. Funkcije koje su prikazane na slici 51 rade zvukove hodanja s obzirom na to koja je noga u animaciji došla do poda.

```
public void PlayLeftLegSound(string runOrWalk){
    if(runOrWalk == "Run"){
        leftLeg.clip = RandomAudio(allAudiosForCurrentFloorRun);
    }
    else{
        leftLeg.clip = RandomAudio(allAudiosForCurrentFloorWalk);
    }
    leftLeg.Play();
    //Debug.Log("LeftLeg Sound");
}

0 references
public void PlayRightLegSound(string runOrWalk){
    if(runOrWalk == "Run"){
        rightLeg.clip = RandomAudio(allAudiosForCurrentFloorRun);
    }
    else{
        rightLeg.clip = RandomAudio(allAudiosForCurrentFloorWalk);
    }
    rightLeg.Play();
    //Debug.Log("RightLeg Sound");
}
```

Slika 50. Prikaz funkcija za kreiranje zvuka iz AudioPlayer skripte

Skripta također pruža opciju pravljenja zombi zvukova. Prilikom kreiranja skripte, provjerava se je li objekt zombi i ako je poziva se Invoke funkcija. Ona omogućuje pozivanje neke funkcije nakon nekog određenog vremena te se tako pokreće rekurzivna funkcija PlayZombieNoise koja kreira zombi zvuk kao što je prikazano na slici 52.

```
0 references
private void PlayZombieNoise(){
    Vector3 spawnPosition = new Vector3(transform.position.x, transform.position.y+1, transform.position.z
    Instantiate(zombieNoises[Random.Range( 0,zombieNoises.Count)],spawnPosition,Quaternion.identity,transf
    Invoke("PlayZombieNoise", Random.Range(minRepeatTime,maxRepeatTime));
}
```

Slika 51. PlayerZombieNoise funkcija iz AudioPlayer skripte

10.4. Stats skripta

Ova skripta omogućuje praćenje zdravlja igrača ili nekog drugog objekta na mrežnoj igri, ažurira vizualni prikaz ukoliko je u pitanju igrač i upravlja zvučnim efektom ovisno o stanju zdravlja. Također osigurava da se promjene zdravlja sinkroniziraju između svih klijenata i servera.

Slika 53 prikazuje dvije mrežne varijable hp i playerName. Hp i playerName ažuriraju se na svim klijentima i serveru zbog toga što su mrežne varijable. NetworkVariable ne podržava stringove jer su oni u C# nepromjenjivi tipovi, što ih sprječava da se deserijaliziraju na licu mjesta. Tako da bi svako ažuriranje string tipa podatka dovelo do alokacije nove string varijable, što bi moglo dovesti do problema s preformansom. Zbog toga Unity nudi svoje tipove varijabli FixedStrings. Potrebno je kreirati novu strukturu s INetworkSerializable sučeljem pod nazivom PlayerNameStruct koja omogućuje korištenje FixedString64Bytes tipa podatka te serijalizaciju same varijable _playerName.

```
4 references
public NetworkVariable<int> hp = new NetworkVariable<int>(100,
                                        NetworkVariableReadPermission.Everyone,
                                        NetworkVariableWritePermission.Server);


4 references
public NetworkVariable<PlayerNameStruct> playerName = new NetworkVariable<PlayerNameStruct>(new PlayerNameStruct{
                                                            _playerName = ""
                                                        },
                                                        NetworkVariableReadPermission.Everyone,
                                                        NetworkVariableWritePermission.Owner);

6 references
public struct PlayerNameStruct : INetworkSerializable
{
    6 references
    public FixedString64Bytes _playerName;

    0 references
    public void NetworkSerialize<T>(BufferSerializer<T> serializer) where T : IReaderWriter{
        serializer.SerializeValue(ref _playerName);
    }
}
```

Slika 52. Mrežne varijable hp i playerName iz Stats skripte

Slika 54 prikazuje funkciju koja se izvršava prilikom kreiranja objekta. Varijabla hp se postavlja na maxHp te se pretplaćuje na događaj hp.OnValueChanged koji će se pozivati svaki put kada se životni bodovi objekta promjene. U ovom slučaju prilikom promjene životnih bodova taj event omogućuje da pozovemo funkciju za ažuriranje vizualnog prikaza i provjeru smrti igrača. Poslije toga se pretplaćuje na playerName.OnValueChanged koji mijenja sadržaj teksta koji služi za prikazivanje imena igrača ostalim igračima.

```
public override void OnNetworkSpawn()
{
    maxHP = hp.Value;
    hp.OnValueChanged += (int oldHP, int newHP) =>
    {

        UpdateHPImage(newHP);


        if (newHP <= 0)
        {
            if (heartbeatAudioSource != null)
                heartbeatAudioSource.Stop();

            GetComponent<IDied>().Died();
        }
    };

    playerName.OnValueChanged += (PlayerNameStruct oldName, PlayerNameStruct newName) =>
    {
        nameText.text = newName._playerName.ToString();

        if (IsOwner)
        {
            nameText.gameObject.SetActive(false);
        }
    };

}
```

Slika 53. OnNetworkSpawn funkcija iz Stats skripte

Pozivanjem UpdateHpImage funkcije, prikazanoj na slici 55, ažurira se vizualni prikaz zdravlja i uključuje zvuk otkucaja srca ako je igrač blizu smrti.

```
void UpdateHPImage(int newHP){
    Debug.Log("Shake");
    if(!IsOwner)
        return;
    if(gameObject.CompareTag("Player")){
            Debug.Log("Changing image hp");
            CameraShake.Instance.Shake(3f,0.1f);
            PlayerUIManager.Instance.UpdateHpImage(newHP, maxHP);

            float precentageHpLeft = (float) newHP/maxHP;
            if( precentageHpLeft <=0.3f ){


                if(!heartbeatAudioSource.isPlaying)
                    heartbeatAudioSource.Play();
                if(precentageHpLeft<0.15f)
                    heartbeatAudioSource.pitch = 1.4f;
                else
                    heartbeatAudioSource.pitch = 1f;
            }
            else
                heartbeatAudioSource.Stop();
        }
}
```

Slika 54. UpdateHpImage funkcija iz Stats skripte

Stats skripta implementira IDamagable sučelje koje u sebi ima samo jednu funkciju TakeDmg. Stats implementira tu funkciju i šalje količinu životnih bodova koju je potrebno oduzeti s igrača ili nekog drugog objekta kao što je prikazano na slici 56.

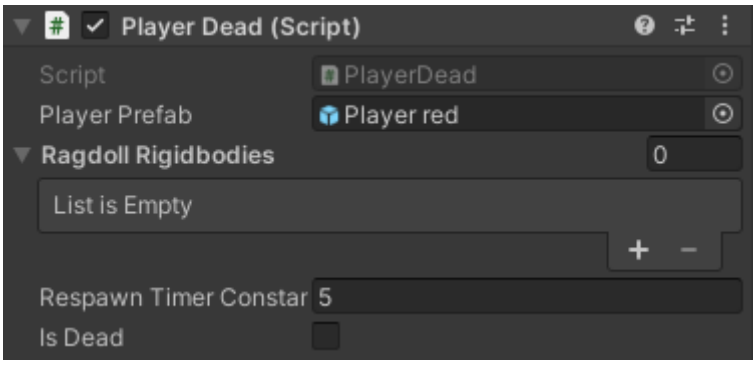
Za objekte koji mogu napadati neke druge objekte nije bitno ima li taj drugi objekt Stats skriptu na sebi, već će tražiti funkciju TakeDmg od IDamagable sučelja.

```
public void TakeDmg(int dmgTaken){
    TakeDmgServerRpc(dmgTaken);
}

[ServerRpc(RequireOwnership = false)]
1 reference
public void TakeDmgServerRpc(int dmgTaken){
    if(IsServer)
        hp.Value -=dmgTaken;
}
```

Slika 55. Funkcije za slanje štete iz Stats skripte

10.5. PlayerDead skripta

Slika 57 prikazuje PlayerDead skriptu kao komponentu na Player objektu. Prilikom pokretanja igre PlayerDead skripta dohvaća sve Rigidbody komponente s igrača, postavlja ih da su deaktivirani pomoću DisableRagdoll funkcije, prikazane na slici 58, i aktivira potreban vizualni prikaz ako igrač upravlja objektom.



Slika 56. Prikaz PlayerDead komponente na Player objektu

```
private void Start()
{
    Rigidbody[] ragdoll = GetComponentsInChildren<Rigidbody>();
    ragdollRigidbodies = ragdoll.ToList();

    DisableRagdoll();
    if(IsOwner)
        ActivateUI();

}

1 reference
void ActivateUI(){
    PlayerUIManager.Instance.PlayerAliveUI();
}

1 reference
void DisableRagdoll()
{
    foreach (Rigidbody rig in ragdollRigidbodies)
    {
        rig.isKinematic = true;
    }
}
```

Slika 57. Start, ActivateUI i DisableRagdoll funkcije iz PlayerDead skripte

Ako igrač umre poziva se Died funkcija prikazana na slici 59. Koja onesposobi sve kontrole igrača i sve ostale komponente nad kojima igrač ima kontrolu ili koje bi mogle utjecati na ragdoll efekt. Te zatim aktivira ragdoll efekt i poziva ga na svim klijentima. Zatim, ako je igrač vlasnik objekta poziva mu se RespawnAfter korutina koja će ga opet stvoriti na nekom od ostalih živih igrača.

```
public void Died()
{
    DisableComponents();
    ActivateRagdoll();
    AcivateRagdollClientRpc();

    if (!IsOwner )
        return;
    Debug.LogWarning("Indeed it is the owner");
    StopAllCoroutines();
    StartCoroutine(RespawnAfter(respawnTimerConstant));
}

2 references
void DisableComponents()
{
    GetComponent<Animator>().enabled = false;
    GetComponent<CharacterController>().enabled = false;
    GetComponent<Rigidbody>().isKinematic = true;
    GetComponent<PlayerGuns>().enabled = false;
    GetComponent<Player>().enabled = false;
    GetComponent<Stats>().enabled = false;
    GetComponent<PlayerInteraction>().enabled = false;
    GetComponentInChildren<Rig>().weight = 0;
    GameManager.Instance.allPlayers.Remove(gameObject);
    if(IsOwner){
        PlayerUIManager.Instance.PlayerDeadUI();
        if(GameManager.Instance.allPlayers.Count != 0)
            PlayerCameraFollow.Instance.FollowPlayer(GameManager.Instance.allPlayers[Random.Range(0, GameManager.Instance.al
    }
}

2 references
void ActivateRagdoll()
{
    foreach (Rigidbody rig in ragdollRigidbodies)
    {
        if (rig == null)
            continue;
        rig.isKinematic = false;
    }
    //after 5-10s become kinematic
}
```

Slika 58. DisableComponents, Dead i ActivateRagdoll funkcija iz PlayerDead skripte

Slika 60 prikazuje RespawnAfter korutinu koja će s while petljom postepeno smanjivati tajmer za ponovno kreiranje igrača te će to vrijeme prikazati na ekranu. Kada vrijeme bude manje od 0, šalje se ServerRpc zahtjev serveru za ponovno kreiranje igrača uz slanje igračevog ID-a kako bi server mogao staviti vlasništvo novo kreiranog objekta na igrača te se prijašnje tijelo igrača makne iz igre.

```
IEnumerator RespawnAfter(int time){
    PlayerUIManager.Instance.UpdateRespawnTime(time);
    while(time>=0){
        yield return new WaitForSeconds(1);
        time--;
        PlayerUIManager.Instance.UpdateRespawnTime(time);
    }

    RespawnServerRpc(OwnerClientId);
    Invoke("DespawnInServerRpc", 2f);
}
[ServerRpc(RequireOwnership = false)]
1 reference
void RespawnServerRpc(ulong clientID){

    Debug.Log("New player spawining, if not WHY NOT???");
    GameObject newPlayer = Instantiate(GameManager.Instance.playerPrefab,
                                       GameManager.Instance.allPlayers[Random.Range(0,GameManager.Instance.allPlayers.Count)].transform.position,
                                       Quaternion.identity);
    newPlayer.GetComponent<NetworkObject>().SpawnWithOwnership(clientID);

}
```

Slika 59. RespawnAfter korutina iz PlayerDead skripte

10.6. PlayerGuns skripta

PlayerGuns skripta upravlja funkcionalnostima oružja koje igrač koristi. Prati trenutno oružje s kojim igrač igra, omogućuje ciljanje, pucanje i akcije poput bacanja granata. Također, upravlja animacijama igrača te osim toga omogućuje igraču da prebacuje između različitih oružja, kao i ponovno punjenje oružja. Po potrebi, mijenja jačinu utjecaja Animation Rigging komponenata.

S obzirom na to kako je igrač objekt napravljen sve puške koje su u igri se nalaze kao djeca desne ruke, samo nisu aktivni. Tijekom igre igrač, skupljanjem novog oružja, aktivira i deaktivira potrebne objekte te iste koristi.

U Start funkciji, prikazanoj na slici 61, prolazi se kroz sva oružja koja su pod igračem te se sva deaktiviraju. Nakon toga aktivira se oružje za početak te se promjeni vizualni prikaz. Zatim se pomoću SwitchWeight funkcije mijenja utjecaj Animation Rigging komponente za lijevu ruku ovisno o tome je li oružje koje se koristi pištolj ili nije.

```
void Start()
{
    if (!IsOwner)
        return;

    grenadeRefilTimer = grenadeRefilTimerConstant;
    flareRefilTimer = flareRefilTimerConstant;
    throwAmountGrenadeCurrent = throwAmountGrenadeMax;
    throwAmountFlareCurrent = throwAmountFlareMax;
    UpdatePlayerUIThrowable();

    AudioListener.volume = PlayerPrefs.GetFloat("masterAudioVolume");
    MenuManager.Instance.ActivatePlayerUI();
    foreach (Weapon weapon in GetComponentsInChildren<Weapon>())
    {
        allWeapons.Add(weapon);
        foreach (GameObject weaponGO in startWeapon)
            if (weapon.name == weaponGO.name)
            {
                equipedWeapons.Add(weapon);
                break;
            }
        weapon.isEnabled.Value = false;
    }
    usingWeapon = equipedWeapons[0];
    usingWeapon.isEnabled.Value = true;

    PlayerUIManager.Instance.UpdateWeaponIcon(usingWeapon.weaponIcon);
    PlayerUIManager.Instance.UpdateAmmoMagazineText(usingWeapon.curentMagazineAmmoCount, usingWeapon
    PlayerUIManager.Instance.UpdateAmmoRemainingText(usingWeapon.curentAmmoCount);

    SwitchWeight();
    Invoke("SetAimTarget", 1f);
}
```

Slika 60. Start funkcija iz PlayerGuns skripte

S obzirom na to da je update funkcija dosta velika bit će podijeljena u manje dijelove tako da će sljedeći dio opisivati update funkciju.

Korištenjem raycasta dobivamo točku na koju igrač cilja s mišem na ekranu te na koju se aimTarget postavlja, kao što je prikazano na slici 62. U igri je aimTarget pozicija gdje igračevo oružje cilja, laser oružja pokazuje smjer u kojem se taj cilj nalazi.

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit;
if (Physics.Raycast(ray, out hit, Mathf.Infinity))
{

    if (hit.point.y > transform.position.y + 1.5f)
        aimTarget.position = new Vector3(hit.point.x, transform.position.y + 1.5f, hit.point.z);
    else
        aimTarget.position = hit.point;
}
```

Slika 61. Prvi dio Update funkcije iz PlayerGuns skripte

Slika 63 prikazuje provjeru igračevog unosa za granate gdje igrač pritiskom na tipku Q može promijeniti granatu. Pritiskom desnog klika miša igraču se daje mogućnost bacanja granate s obzirom na to koliko mu ih je preostalo. Ako se granata može baciti onda se igraču ažurira vizualni prikaz ekrana. Te se šalje poziv serveru za bacanje granate.

```
if (Input.GetKeyDown(KeyCode.Q))
{
    SelectNewGrenade();
}

//throw Granade
if (Input.GetKeyDown(KeyCode.Mouse1)
    && !AnimationPlayingCheck(new string[] { "Reload", "SwitchWeapon", "New Gun" }, 1)
    && !AnimationPlayingCheck(new string[] { "Run", "Falling", "Landing" }, 0))
{
    bool canThrow= false;
    if(throwablePrefabsList[curentThrowableIndex.Value].CompareTag("Flare")){
        if(throwAmountFlareCurrent>0){
            canThrow = true;
            throwAmountFlareCurrent--;
        }

    }

    if(throwablePrefabsList[curentThrowableIndex.Value].CompareTag("Grenade")){
        if(throwAmountGrenadeCurrent>0){
            canThrow = true;
            throwAmountGrenadeCurrent--;
        }
    }

    if(canThrow){
        UpdatePlayerUIThrowable();
        ThrowObjectServerRpc(curentThrowableIndex.Value);
    }
}
```

Slika 62. Drugi dio Update funkcije iz PlayerGuns skripte

Slika 64 prikazuje unose s tipkovnice koje igrač ima mogućnost koristiti. Pritiskom na lijevi klik miša igrač ima mogućnost pucanja, osim ako ne radi neke druge animacije pod kojima pucanje nije moguće. Pritiskom na R tipku igrač može napuniti magazin oružja, ali mora pričekati da se animacija izvrši jer se punjenje radi pomoću animacijskog događaja. Igrač može koristiti kotačić na mišu za promjenu oružja gdje se isto kao i kod punjenja pokreće animacija koja mora doći do kraja te se tek onda dogodi promjena oružja.

```
if (Input.GetKey(KeyCode.Mouse0)
    && usingWeapon.fireRate <= 0
    && !AnimationPlayingCheck(new string[] { "Reload", "SwitchWeapon", "New Gun" }, 1)
    && !AnimationPlayingCheck(new string[] { "Run", "Falling", "Landing" }, 0))
{

    usingWeapon.Shoot();
    PlayerUIManager.Instance.UpdateAmmoMagazineText(usingWeapon.curentMagazineAmmoCount,
                                                    usingWeapon.maxMagazineCount);
}

//reload
if (Input.GetKeyDown(KeyCode.R)
    && !AnimationPlayingCheck(new string[] { "SwitchWeapon", "New Gun" }, 1)
    && !AnimationPlayingCheck(new string[] { "Run", "Falling", "Landing" }, 0))
{

    anim.Play("Reload", 1);
}

if (Input.mouseScrollDelta.y != 0
    && !AnimationPlayingCheck(new string[] { "Falling", "Landing" }, 0)
    && equipedWeapons.Count == 2)
{
    scrollDirection = Input.mouseScrollDelta.y;
    //Debug.Log(scrollDirection);
    //Debug.Log(Input.mouseScrollDelta);
    anim.Play("SwitchWeapon", 1);
}
```

Slika 63. Treći dio Update funkcije iz PlayerGuns skripte

Ruke tijekom svih animacija ne mogu biti na istim pozicijama. Tako da dio koda prikazan na slici 65 provjerava koje se animacije rade te postepeno mijenja utjecaj Rig komponente nad animacijom. To je potrebno napraviti kako ne bi došlo do instantnog prebacivanja s 0 na 1 utjecaja Rig komponente već da dobijemo glatku tranziciju.

```
if (AnimationPlayingCheck(new string[] { "SwitchWeapon", "New Gun", "Reload" }, 1)
    || AnimationPlayingCheck(new string[] { "Falling", "Landing", "Run" }, 0))
{
    if (rig.weight > 0.2)
    {
        rig.weight -= Time.deltaTime * 10;
        net_rigWeight.Value = rig.weight;
    }
    else
    {
        rig.weight = 0;
        net_rigWeight.Value = rig.weight;
    }
    //Debug.Log("Some animations playing " + rig.weight);
}
else
{
    if (rig.weight < 0.8)
    {
        rig.weight += Time.deltaTime * 5;
        net_rigWeight.Value = rig.weight;
    }
    else
    {
        rig.weight = 1;
        net_rigWeight.Value = rig.weight;
        SetAimTarget();
    }
    //Debug.Log("Some animations playing " + rig.weight);
}
```

Slika 64. Četvrti dio Update funkcije iz PlayerGuns skripte

Prilikom prikupljanja novog oružja koristi se funkcija NewWeapon prikazana na slici 66. Ona provjerava postoji li već oružje među oružjima koje igrač ima na raspolaganju i ako da samo mu napuni municiju. Ako novo oružje nije na raspolaganju, onda provjerava ima li igrač maksimalan broj oružja koje može koristiti. Ukoliko igrač nema maksimalan broj oružja koje može koristiti, funkcija mu postavlja novo oružje na tu poziciju, a ukoliko igrač ima maksimalan broj oružja koje može koristiti onda zamjenjuje trenutno oružje s novim.

```
public void NewWeapon(Weapon newWeapon)
{
    bool isBeingUsed = false;
    //ako postoji gun napuni metke
    foreach (Weapon weapon in equipedWeapons)
    {
        if (weapon.name == newWeapon.name)
        {
            //if exists replanishAmmo
            isBeingUsed = true;
            weapon.ReplanishAmmo();
            break;
        }
    }
    //ako ne postoji gun treba se aktivirati
    if (!isBeingUsed)
    {
        Debug.Log("Old weapon sould be disabled");
        if(maxWeaponCarry== equipedWeapons.Count)
            equipedWeapons.Remove(usingWeapon);
        usingWeapon.isEnabled.Value = false;
        foreach (Weapon weapon in allWeapons)
        {
            if (newWeapon.name == weapon.name)
            {
                usingWeapon = weapon;
                break;
            }
        }
    }
    equipedWeapons.Add(usingWeapon);
    Debug.Log(usingWeapon.name);
    usingWeapon.isEnabled.Value = true;
    UpdatePlayerUI();
    Invoke("SwitchWeight", 0.2f);
    Debug.Log("New weapon added");
    newWeapon.GetComponentInParent<IDestroy>().DestroyObject();
}
```

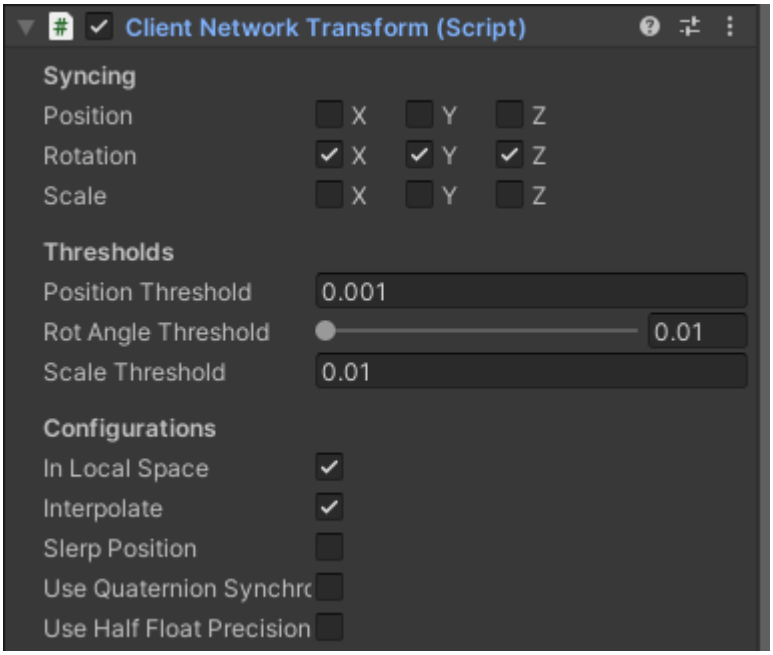
Slika 65. NewWeapon funkcija iz PlayerGuns skripte

11. Oružje

Modeli za oružje su preuzeti s Unity Asset Stora [21]. Svako oružje na sebi ima 3 glavne komponente potrebne za funkcionalnost. To su Weapon skripta, Line Renderer (koja služi za prikaz lasera u igri) i ClientNetworkTransform.

LineRenderer komponenta koristi niz od najmanje dvije točke u 3d prostoru i crta ravnu liniju između svake od njih. Može se koristiti za crtanje različitih oblika, od jednostavne ravne linije do složenih spirala [13].

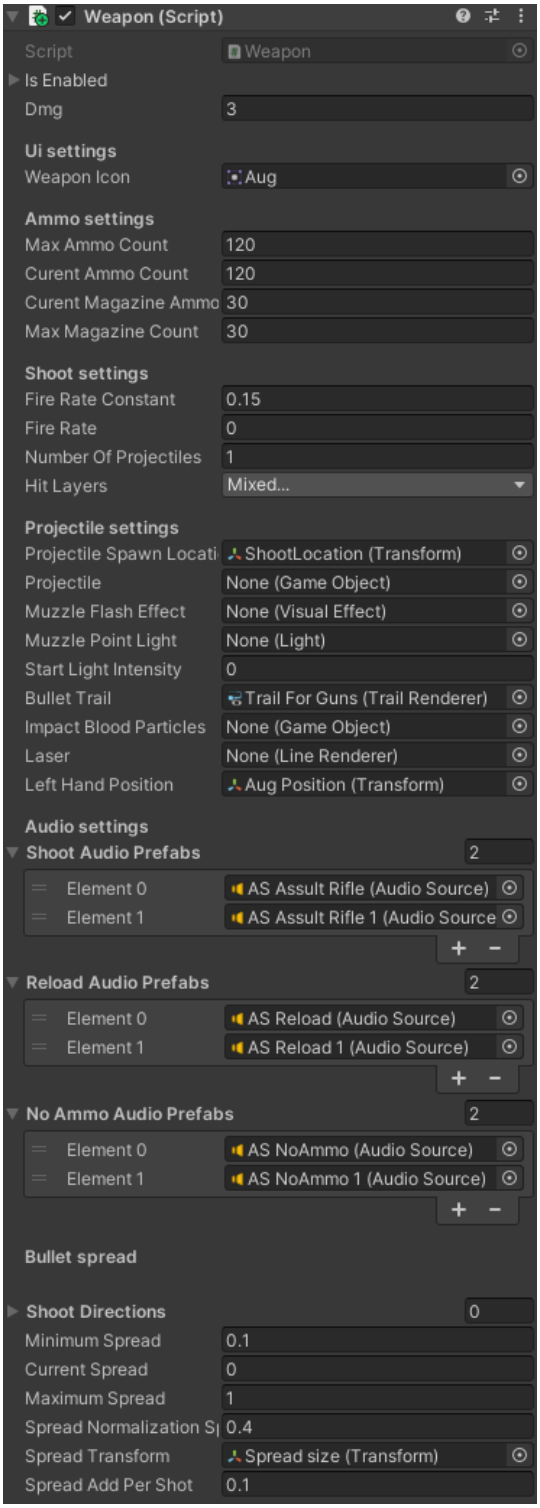
ClientNetworkTransform komponenta objašnjena je u sekciji 7.7. Ona na oružju mora sinkronizirati samo rotaciju i to u lokalnom prostoru, odnosno, u usporedbi na svog roditelja. ClientNetworkTransform komponenta prikazana je na slici 67.



Slika 66. Prikaz ClientNetworkTransform komponente na oružju

11.1. Weapon skripta

Weapon skripta upravlja svim sposobnostima oružja. Slika 68 pokazuje Weapon skriptu kao komponentu na oružju.



Slika 67. Prikaz Weapon komponente na oružju

Slika 69 prikazuje Update funkciju koja upravlja Line Rendererom u Laser funkciji, smanjuje vrijeme koje je potrebno do sljedeće mogućnosti pucanja te vraća lokalnu veličinu objekta za raspršivanje metaka na minimum.

Sam objekt za raspršivanje metaka na sebi također ima ClientNetworkTransform komponentu koja gleda lokalnu veličinu objekta tako da se sinkronizira svim igračima.

```
void Update()
{

    Laser();
    if (fireRate > 0)
        fireRate -= Time.deltaTime;

    if (IsOwner)
    {
        if (currentSpread > minimumSpread)
        {
            if (currentSpread > maximumSpread)
                currentSpread = maximumSpread;
            currentSpread -= Time.deltaTime * spreadNormalizationSpeed;
            spreadTransform.localScale = new Vector3(currentSpread,
                                                     currentSpread,
                                                     spreadTransform.localScale.z);
        }
    }
}
```

Slika 68. Update funkcija iz Weapon skripte

Slika 70 prikazuje Shoot funkciju koja se poziva u PlayerGuns skripti. Shoot funkcija provjerava je li igraču dozvoljeno da puca. Ako mu je dozvoljeno da puca, odabire jednu od pozicija koje se nalaze u objektu za širenje te puca metak kod igrača, kreira prikladan zvuk i zatim šalje poziv na server da je potrebno pucati.

```
public void Shoot()
{

    if (fireRate > 0)
    {
        return;
    }
    if (curentMagazineAmmoCount <= 0)
    {
        Debug.Log("No ammo sound");
        fireRate = 0.4f;
        SpawnAudio("NoAmmo");
        if (IsHost)
        {
            SpawnAudioClientRpc("NoAmmo");
        }
        else
        {
            SpawnAudioServerRpc("NoAmmo");
        }
        return;
    }
    currentSpread += spreadAddPerShot;

    curentMagazineAmmoCount--;
    Debug.Log("Shoot");


    bool useAudio = true;


    for (int i = 0; i < numberOfProjectiles; i++)
    {
        int shootAtIndex = Random.Range(0, shootDirections.Count);
        ShootRaycast(true, shootAtIndex, useAudio);

        if (IsHost)
            ShootSyncClientRpc(shootAtIndex, useAudio);
        else
            ServerShootServerRpc(shootAtIndex, useAudio);


        useAudio = false;
    }


    fireRate = fireRateConstant;
}
```

Slika 69. Shoot funkcija iz Weapon skripte

ShootRaycast funkcija, prikazana na slici 71, upravlja procesom ispaljivanja metka iz oružja u igri. Kada igrač puca provjerava je li potreban zvuk te pokreće funkciju za kreiranje zvuka i funkcije za kreiranje efekta pucanja ako zvuk jest potreban. Zatim korištenjem Physics.Raycast provjerava je li metak pogodio neki objekt, stvara novi trag metka i pokreće korutinu ovisno o tome može li metak nanositi štetu.

ServerShootServerRpc kreira trag metka na serveru te šalje informaciju ostalim klijentima da igrač puca s ShootSyncClientRpc.

```
void ShootRaycast(bool canDmg, int _shootAtIndex, bool _useAudio)
{
    if (_useAudio)
    {
        SpawnAudio("");
        muzzleFlashEffect.Play();
        StopCoroutine(MuzzleLightFlash());
        StartCoroutine(MuzzleLightFlash());
    }

    RaycastHit hit;

    Vector3 direction = shootDirections[_shootAtIndex].position - projectileSpawnLocation.position;

    if (Physics.Raycast(projectileSpawnLocation.position, direction, out hit, Mathf.Infinity, hitLayers))
    {
        TrailRenderer trail = Instantiate(bulletTrail, projectileSpawnLocation.position, Quaternion.identity);
        if (canDmg)
            StartCoroutine(SpawnTrailAndDMG(trail, hit));
        else
            StartCoroutine(SpawnTrail(trail, hit));
        Debug.Log("Did Hit");
    }
}

//salje hostu da puca i puca host tj server
[ServerRpc(RequireOwnership = false)]
1 reference
public void ServerShootServerRpc(int _shootAtIndex, bool _useAudio)
{
    //Debug.DrawRay(projectileSpawnLocation.position, Camera.main.ScreenPointToRay (Input.mousePosition).direct
    ShootRaycast(false, _shootAtIndex, _useAudio);
    ShootSyncClientRpc(_shootAtIndex, _useAudio);

}

//Server salje ostalima da player puca
[ClientRpc]
2 references
public void ShootSyncClientRpc(int _shootAtIndex, bool _useAudio)
{
    if (IsOwner) return;

    //Debug.DrawRay(projectileSpawnLocation.position, Camera.main.ScreenPointToRay (Input.mousePosition).direct
    ShootRaycast(false, _shootAtIndex, _useAudio);
}
```

Slika 70. ShootRaycast funkcija i dvije funkcije vezane za sinkronizaciju pucanja

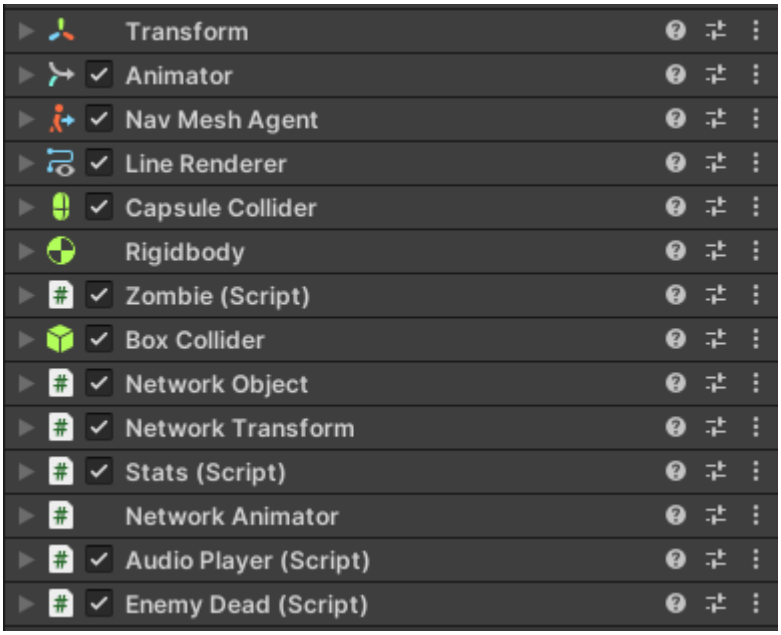
Slika 72 prikazuje SpawnTrailAndDmg funkcija koja pomiče kreirani trag metka od trenutne lokacije do pogođene lokacije u nekom malom određenom vremenu. Nakon toga pokušava oštetiti zadani objekt te kreirati efekt krvi ako je to potrebno.

```
IEnumerator SpawnTrailAndDMG(TrailRenderer trail, RaycastHit hit)
{
    float time = 0;
    Vector3 startPosition = trail.transform.position;
    while (time < 1)
    {
        trail.transform.position = Vector3.Lerp(startPosition, hit.point, time);
        time += Time.deltaTime / trail.time;
        yield return null;
    }
    trail.transform.position = hit.point;
    //instantiate impact particles
    //Instantiate(impactParticles, hit.point, Quaternion.LookRotation(hit.normal));

    Destroy(trail.gameObject, trail.time);

    try
    {
        hit.transform.GetComponent<IDamageable>().TakeDmg(dmg);

        //spawns blood
        if (hit.transform.CompareTag("Zombie"))
        {
            Instantiate(impactBloodParticles, hit.point, Quaternion.LookRotation(-hit.normal));
        }
    }
    catch
    {
        try
        {
            hit.transform.parent.root.GetComponent<IDamageable>().TakeDmg(dmg);
        }
        catch
        {
            Debug.Log("Object does not contain any IDamagable interface");
        }
    }

    Debug.Log("tEST");
}
```

Slika 71. SpawnTrailAndDMG funkcija iz Weapon skripte

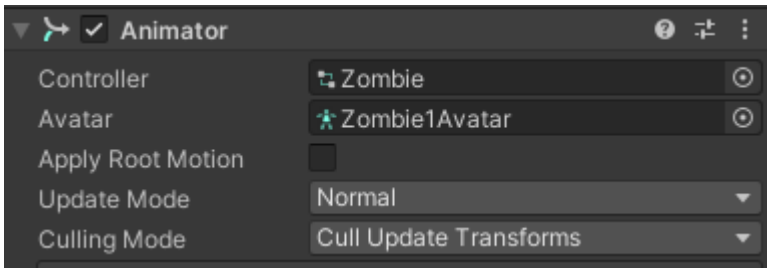
12. Zombi

Zombi model preuzet je s Unity Asset Stora [19]. Slično igračevom objektu, ima velik broj komponenata na sebi. NetworkObject, NetworkTransform i NetworkAnimator objašnjeni su u 7. poglavlju. Slika 73 prikazuje sve komponente na zombi objektu.



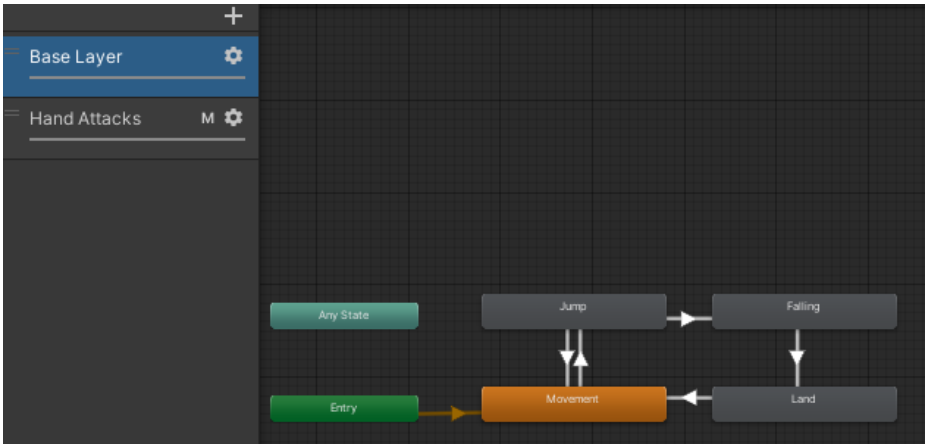
Slika 72. Prikaz svih komponenata na Zombi objektu

Slika 74 prikazuje Animator komponentu koja koristi zombie kontroler i zombie avatar.

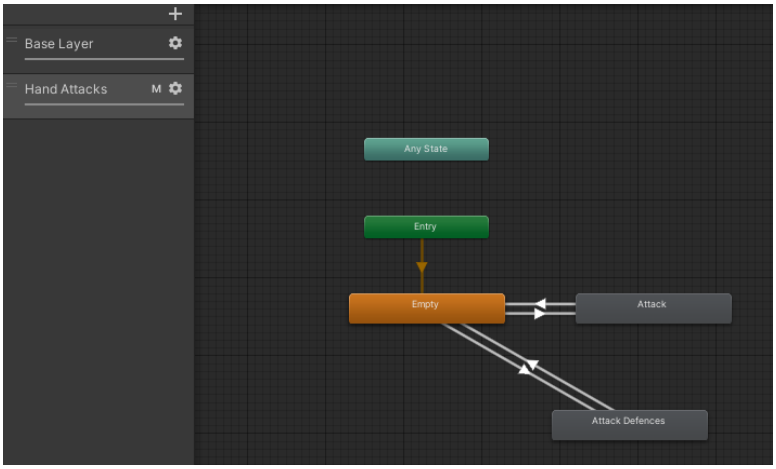


Slika 73. Prikaz Animator komponente na Zombi objektu

Zombie kontroler sadrži Base sloj, prikazan na slici 75, koji izvršava pojedine potrebne animacije, dok Hands Attacks sloj, prikazan na slici 76, upravlja određenim dijelovima zombija, u ovom slučaju gornjim dijelom tijela prilikom napadanja.



Slika 74. Prikaz Baze Layer sloja Zombie kontrolera



Slika 75. Prikaz Hands Attack sloja Zombie kontrolera

Rigidbody i Capsule Collider služe za detektiranje kolizija, dok je Box Collider okidač koji se koristi za kreiranje područja prilikom napada zombija. U tom se području provjerava postoji li igrač te mu zatim nanosi štetu.

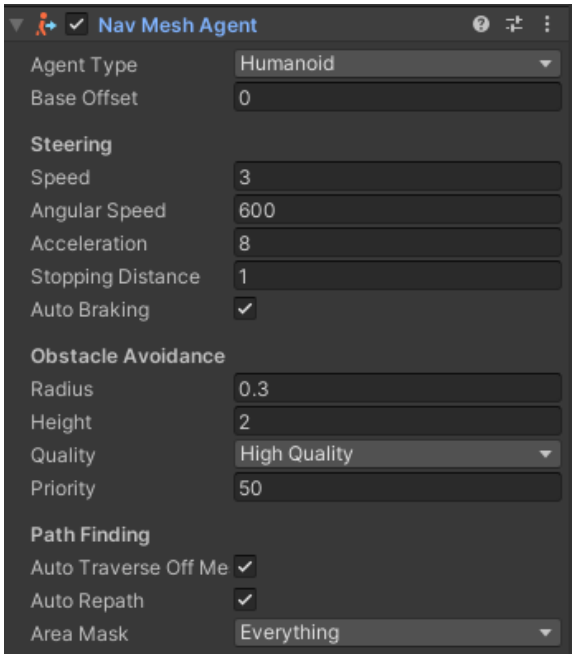
Stats skripta radi na istom principu kao i kod igrača, ali nema dodatne logike za aktiviranje zvučnih efekata pri niskom zdravlju i ne upravlja prikazom na ekran.

Enemy Dead skripta radi isto što i Player Dead skripta, jedina je razlika što deaktivira drugačije komponente.

Dok je u Audio Playeru, prikazanom na slici 77, omogućena Is Zombie varijabla koja omogućava kreiranje dodatnih zvukova tijekom igre. U ovom slučaju to su zvukovi zombija.

Slika 76. Prikaz AudioPlayer komponente sa Zombi objekta

Unity Navigacijski sustav omogućuje developerima da definiraju područja gdje se agenti (u ovom slučaju zombiji) smiju slobodno kretati putem NavMesh-a. Pomoću NavMesh-a moguće je kreirati zone na kojima se agenti mogu kretati. Koristi se za omogućavanje planiranja i izračuna optimalnih staza koje će agenti pratiti. Primarni alat za pokretanje agenata je NavMeshAgent prikazan na slici 78. NavMeshAgent omogućuje postavljanje ciljeva i automatsko kretanje. Osim jednostavnog kretanja, developeri mogu implementirati taktike poput izbjegavanja prepreka, grupnog kretanja i praćenja ciljeva.



Slika 77. Prikaz NavMeshAgent komponente sa Zombi objekta

12.1. Zombie skripta

Zombie skripta odgovorna je za ponašanje zombija u igri. Zombiji su sposobni pratiti i napadati igrača. Imaju određeni domet napada definiran varijablom attackRange. Također koriste NavMeshAgent komponentu za kretanje prema igračima.

U Start funkciji koristi se InvokeRepeating za funkciju FollowPlayer koja je prikazana na slici 79. FollowPlayer svake sekunde prolazi kroz listu živih igrača te postavlja najbližeg igrača kao cilj kretanja.

FixedUpdate funkcija konstantno postavlja destinaciju NavMeshAgent-a na cilj koji je postavljen u FollowPlayer funkciji.

```
void FollowPlayer()
{
    List<GameObject> players = GameManager.Instance.allPlayers;
    if (players.Count == 0)
        return;
    //Debug.Log(players.Count);
    float distance = 500;
    if (followTarget != null)
        distance = Vector3.Distance(transform.position, followTarget.transform.position);
    foreach (GameObject player in players)
    {
        if (player == followTarget || player == null)
            continue;

        float newDistance = Vector3.Distance(transform.position, player.transform.position);
        if (Vector3.Distance(transform.position, player.transform.position) < distance)
        {
            followTarget = player.transform;
            distance = newDistance;
        }
    }

}
```

Slika 78. FollowPlayer funkcija iz Zombie skripte

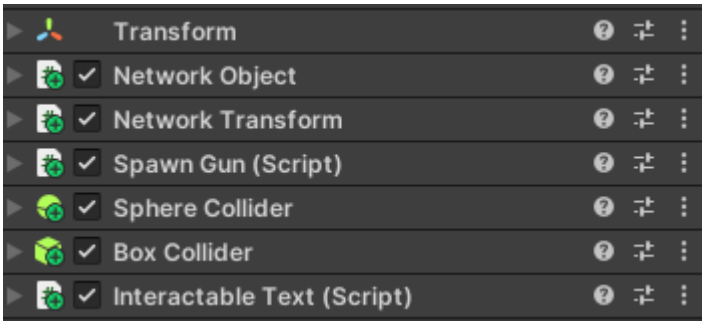
U funkciji Update prikazanoj na slici 80, zombi provjerava je li u trenutnoj animaciji zvanoj Land, odnosno, animaciji koja se izvodi kada zombi slijeće s neke visine. Također, zombi provjerava svoju poziciju u odnosu na određene navigacijske komponente OffMeshLink, koje označavaju područja za skakanje ili prijelaz preko prepreka. Ako je zombi blizu igrača, koristi se funkcija AttackCheck za utvrđivanje je li dovoljno blizu igrača kako bi izveo napad.

```
protected override void Update()
{
    if(!IsServer)
        return;
    if (followTarget == null)
    {
        Debug.LogWarning("No Player");
        return;
    }


    if (anim.GetCurrentAnimatorStateInfo(0).IsName("Land"))
    {
        navAgent.isStopped = true;
        Debug.Log("Sould be stopped");
    }
    else if (!anim.GetCurrentAnimatorStateInfo(0).IsName("Land") &&
            navAgent.isStopped)
    {
        navAgent.isStopped = false;
    }

    ChangeAgentPriority();

    anim.SetFloat("Move Speed", currentSpeed);

    currentSpeed = navAgent.velocity.magnitude;

    if (anim.GetBool("Off Mesh Link") != navAgent.isOnOffMeshLink)
    {
        anim.SetBool("Off Mesh Link", navAgent.isOnOffMeshLink);
        if (navAgent.currentOffMeshLinkData.startPos.y != navAgent.currentOffMeshLinkData.endPos.y)
        {
            anim.SetBool("Falling", true);
        }
        else
        {
            anim.SetBool("Falling", false);
        }
    }
    else
    {
        AttackCheck();
    }
    GroundCheck();
}
```

Slika 79. Update funkcija iz Zombie skripte

13. Škrinja s oružjem

Model škrinje preuzet je s Unity Asset Stora [18]. Škrinja služi za kreiranje oružja koje igrač može pokupiti i koristiti tijekom igre. Slika 81 prikazuje sve komponente koje se nalaze na škrinji.



Slika 80. Prikaz svih komponenti na škrinji za puške

Škrinja je opremljena mrežnim komponentama jer je potrebno osigurati da se njezina pozicija sinkronizira kod svih igrača. Tijekom otvaranja škrinje, animacija njezinog gornjeg dijela također mora biti sinkronizirana kod svih igrača. Ova škrinja nema vlastiti animator, već se otvaranje škrinje kontrolira putem skripte.

13.1. SpawnGun skripta

Prilikom dolaska igrača blizu škrinje aktivira se Sphere Collider okidač kojemu je cilj dati igraču reference na ovaj objekt te maknuti reference s igrača ako odu izvan dometa. Slika 82 prikazuje funkcije koje koristi Sphere Collider okidač prilikom ulaska i izlaska igrača iz samog okidača.

```
0 references
private void OnTriggerEnter(Collider other)
{
    //display UI for possible interaction
    if(!other.CompareTag("Player")){
        return;
    }
    other.GetComponent<IInteractable>().Interact(transform);
    Debug.Log("Can Interact");
}
0 references
private void OnTriggerExit(Collider other)
{
    //remove UI display for possible interaction
    if(!other.CompareTag("Player")){
        return;
    }
    other.GetComponent<IInteractable>().Interact(transform);
    transform.GetComponent<IText>().HideText();
    Debug.Log("Cant Interact anymore");
}
```

Slika 81. Trigger funkcije iz SpawnGun skripte

Ako se hoće napraviti interakcija s objektom, serveru se šalje funkcija OpenCrateServerRpc prikazana na slici 83.

```
public void Interact(Transform player){
    OpenCrateServerRpc();
}

[ServerRpc(RequireOwnership = false)]
1 reference
public void OpenCrateServerRpc()
{
    if(!IsServer)
        return;

    sphereRadius.Value = 0.1f;
    Debug.Log("Opening Crate");
    StartCoroutine(Rotate(1f));

}
```

Slika 82. Interact i OpenCrateServerRpc funkcije iz SpawnGun skripte

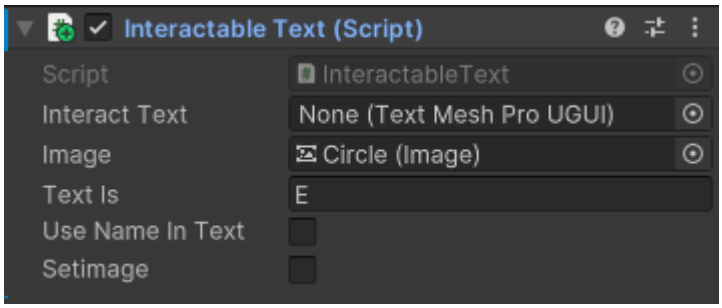
Otvaranje objekta vrši se samo na serveru. Gornji dio škrinje rotira se za 90 stupnjeva te se tada pozivanjem NewGunSpawned kreira novo oružje koje igrač može pokupiti kao što prikazano na slici 84.

```
IEnumerator Rotate(float inTime)
{
    var fromAngle = topPart.localRotation;
    var toAngle = Quaternion.Euler(new Vector3(angleOpend, 0,0));
    for (var t = 0f; t < 1; t += Time.deltaTime / inTime)
    {
        topPart.localRotation = Quaternion.Lerp(fromAngle, toAngle, t);
        yield return null;
    }
    NewGunSpawned();
}

1 reference
public void NewGunSpawned()
{
    if(!IsServer)
        return;

    GameObject newGun = gunsToSpawn[Random.Range(0,gunsToSpawn.Count)];
    newGun = Instantiate(newGun,transform.position,transform.rotation);
    newGun.GetComponent<NetworkObject>().Spawn();
}
```

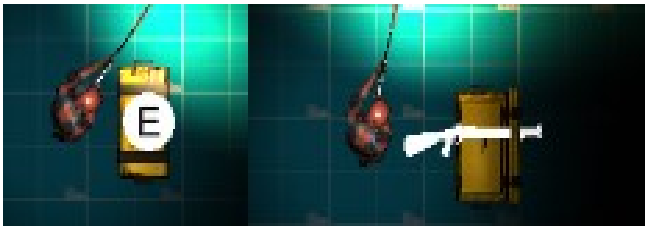
Slika 83. Rotate i NewGunSpawned funkcije iz SpawnGun skripte

13.2. InteractableText skripta

InteractableText skripta, prikazana na slici 85, služi za prikazivanje teksta tipke koju igrač mora pritisnuti kako bi izvršio interakciju s objektom. Može se koristiti slika ili ikona koja označuje interaktivnost, kao što je prikazano na slici 86. U slučaju oružja to bi bila ikona samog oružja.



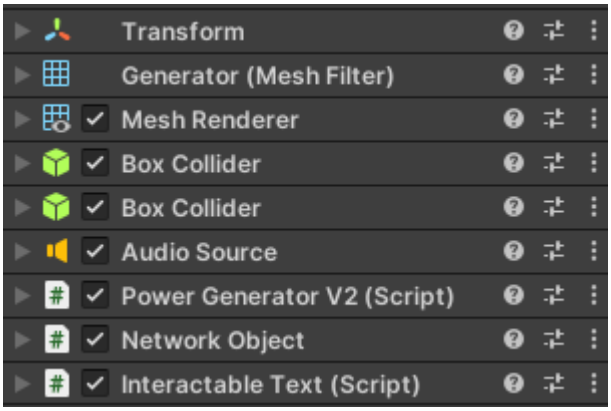
Slika 84. Prikaz InteractableText komponente na škrinji



Slika 85. Prikaz teksta za interakciju i ikone za interakciju tijekom igre

14. Generator

Glavna svrha generatora je aktivacija svjetala na dijelovima mape, otvaranje vrata te aktivacija objekata koji kreiraju zombije. Uz to aktivira i dodatne objekte koji su potrebni. Slika 87 prikazuje sve komponente na generator objektu.



Slika 86. Prikaz svih komponenata na generatoru

Poput škrinje, generator je objekt koji se mora sinkronizirati preko mreže.

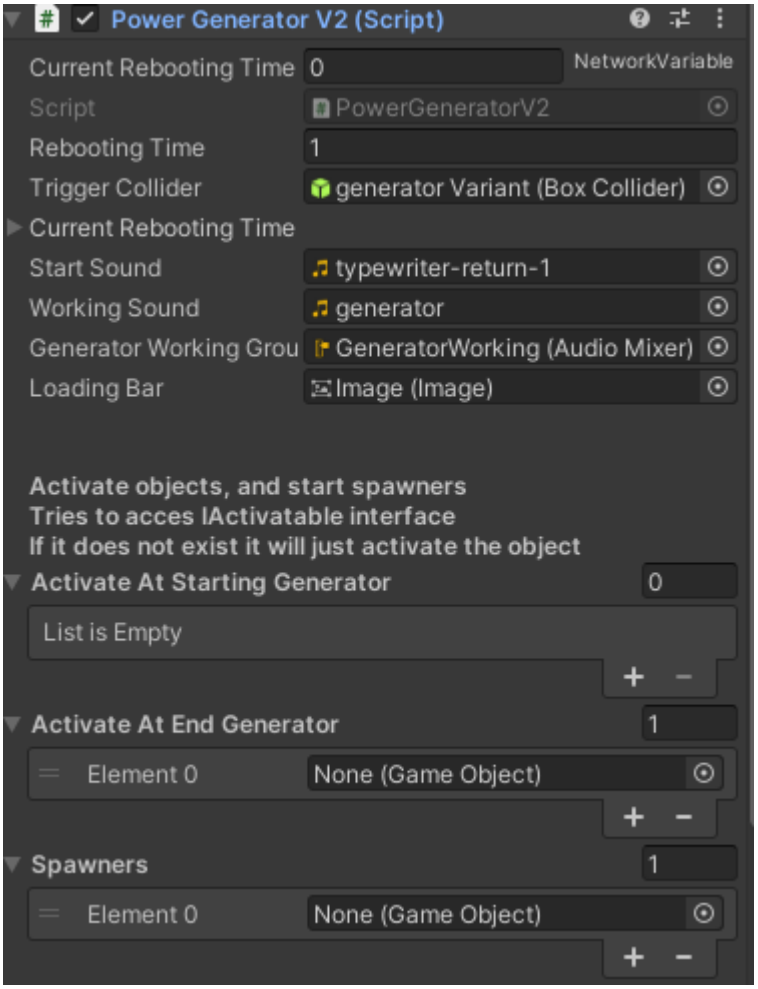
Generator na sebi ima Audio Source komponentu koja će se koristiti kada se generator pokrene i dok radi.

Na njemu se nalaze dvije BoxCollider komponente, od kojih je jedan normalan collider, dok drugi služi kao okidač.

Sadrži istu InteractableText komponentu kao i škrinja.

14.1. PowerGeneratorV2 skripta

PowerGeneratorV2 komponenta prikazana je na slici 88. Koristi OnTriggerEnter i OnTriggerExit funkcije prikazane na slici 89. Služe za dodavanje i micanje objekta iz liste objekata s kojima igrač može interaktirati.



Slika 87. Prikaz PowerGeneratorV2 komponente na generatoru

```
#region Trigger

0 references
private void OnTriggerEnter(Collider other)
{
    if(!other.CompareTag("Player")){
        return;
    }
    //display UI for possible interaction
    other.GetComponent<IInteractable>().Interact(transform);
    Debug.Log("Can Interact");
}
0 references
private void OnTriggerExit(Collider other)
{
    if(!other.CompareTag("Player")){
        return;
    }
    //remove UI display for possible interaction
    other.GetComponent<IInteractable>().Interact(transform);
    transform.GetComponent<IText>().HideText();
    Debug.Log("Cant Interact anymore");
}
#endregion
```

Slika 88. Trigger funkcije iz PowerGeneratorV2 skripte

Isto kao i kod škrinje, nakon što igrač obavi interakciju s generatorom šalje se StartGeneratorServerRpc poziv serveru, koji je prikazan na slici 90, nakon čega server pokreće generator.

```
public void Interact(Transform interactingPlayer){
    //later neki minigame
    //trenutno deaktivirati interaction kod svih

    StartGeneratorServerRpc();
}

[ServerRpc(RequireOwnership = false)]
1 reference
void StartGeneratorServerRpc(){
    xScaleTriggerCollider.Value = 1;
    StartCoroutine( GeneratorStarting());
}
```

Slika 89. Interact i StartGeneratorServerRpc funkcije iz PowerGeneratorV2 skripte

Slika 91 prikazuje GeneratorStarting funkciju koja šalje svim igračima poziv da pokrenu zvuk za pokretanje generatora. Prolazi kroz listu objekata za aktivaciju na startu te ih aktivira. U while petlji postepeno, svake sekunde, mijenja mrežnu varijablu currentRebootingTime. Koja prilikom promjene mijenja izgled punjenja trake iznad generatora. Nakon završetka pokretanja generatora aktiviraju se svi potrebni objekti te se pokreće zvuk rada generatora.

```
IEnumerator GeneratorStarting(){
    StartSoundClientRpc();

    ActivateAtStart();
    int timeToGO = (int) rebootingTime;
    while(timeToGO >=0){
        yield return new WaitForSeconds(1);
        currentRebootingTime.Value += 1;
        timeToGO--;
    }
    ActivateAtEnd();

}
```

Slika 90. GeneratorStarting korutina iz PowerGeneratorV2 skripte

15. Granata i baklja

Postoje dva objekta koje igrač može bacati. Granata i baklja (flare), modeli koji reprezentiraju granatu [16] i baklju [17] preuzeti su s Unity Asset Stora.

Granata sadrži Grenade skriptu koja sadrži Explode funkciju, prikazanu na slici 92, koja prilikom eksplozije poziva Physics.SphereCastAll funkciju koja vraća sve objekte koji su u nekom radijusu te se nalaze u sloju potrebnom detektirati. Vraćenim objektima pokušava skinuti životne bodove.

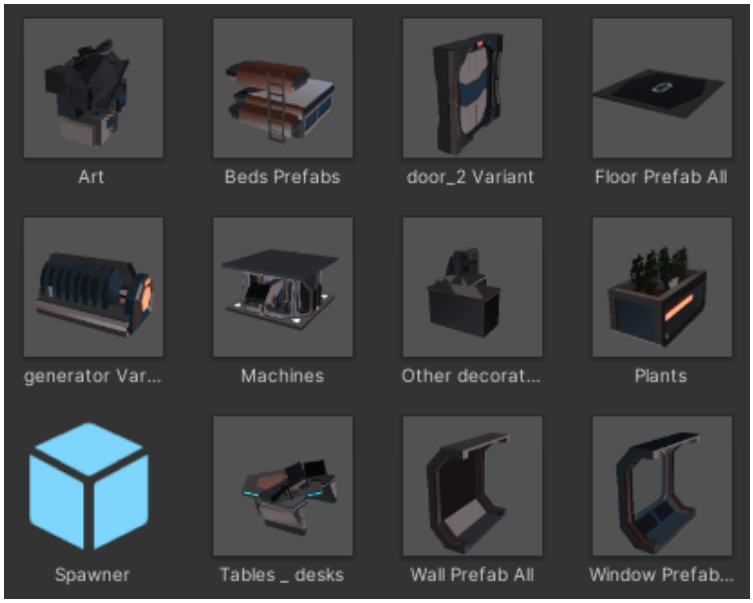
```
void Explode()
{
    SpawnParticleAndAudioClientRpc();
    RaycastHit[] all = Physics.SphereCastAll(transform.position, effectRadius, transform.up, 0f, layerMask);

    foreach (RaycastHit hit in all)
    {
        // Check if the hit object has the IDamagable interface
        IDamageable damagable = hit.collider.GetComponent<IDamageable>();
        if (damagable != null)
        {
            int dmgAmount = (int)(maxDmg * ((effectRadius - Vector3.Distance(transform.position, hit.transform.position))
            damagable.TakeDmg(dmgAmount);
            Debug.Log("Dmged target for " + dmgAmount);
        }
    }
}
```

Slika 91. Explode funkcija iz Grenade skripte

Baklja služi za osvjetljenje nekog područja na određeno vrijeme te se nakon tog vremena ugasi i uništi. Osvjetljenje se izvršava pomoću ParticleSystem komponente koja kreiranim česticama daje mogućnost osvjetljenja.

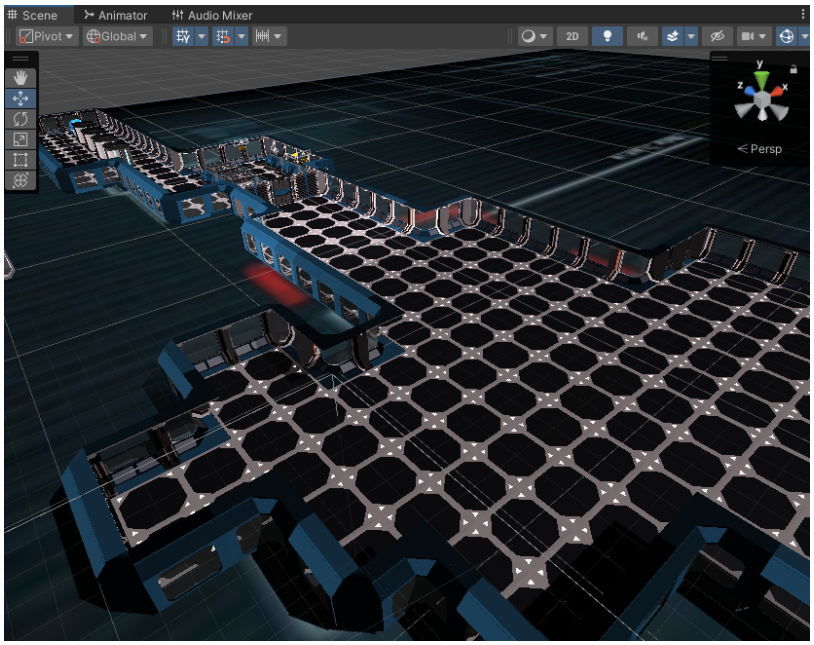
16. Izrada mape

Za izradu mape koriste se modeli iz Unity Asset Stora [22]. S obzirom na to da ovaj paket ima puno različitih modela za zidove, podove, dekoraciju i slično, napravljeni su novi objekti koji su skupina određene vrste objekta kao što je prikazano na slici 93.



Slika 92. Prikaz objekata koji sadrže sve skupine u jednom objektu

Izrada mape je jednostavna. U scenu se stavljaju odabrani objekti koji se zatim pomiču i dupliciraju po potrebi. Također se aktiviraju i deaktiviraju potrebna djeca. Slika 94 prikazuje izgled jednog dijela izgrađene mape.



Slika 93. Snimka ekrana, prikaz igrive mape

16.1. Osvjetljenje mape

Igra bi trebala imati pomalo zastrašujuć osjećaj te su stoga sva svjetla ugašena prilikom igranja. Igrač aktivacijom generatora aktivira pojedina područja i njihova svjetla.
Unity ima 3 vrste osvjetljenja:

- Osvjetljenje u stvarnom vremenu (Realtime Lighting) omogućuje dinamičko osvjetljenje koje se mijenja u stvarnom vremenu. To znači da svjetlost i sjene reagiraju na dinamičke objekte i njihove promjene u sceni
- Pečeno osvjetljenje (Baked Lighting) predstavlja statičko osvjetljenje koje se izračunava unaprijed i pohranjuje u teksture svjetla. Ono se ne mijenja tijekom igre i ne reagira na dinamičke objekte
- Mješovito osvjetljenje (Mixed Lighting) kombinira prednosti osvjetljenja u stvarnom vremenu i pečenog osvjetljenja. Omogućuje korištenje osvjetljenja u stvarnom vremenu za dinamičke objekte i pečenog osvjetljenja za statičke objekte u istoj sceni

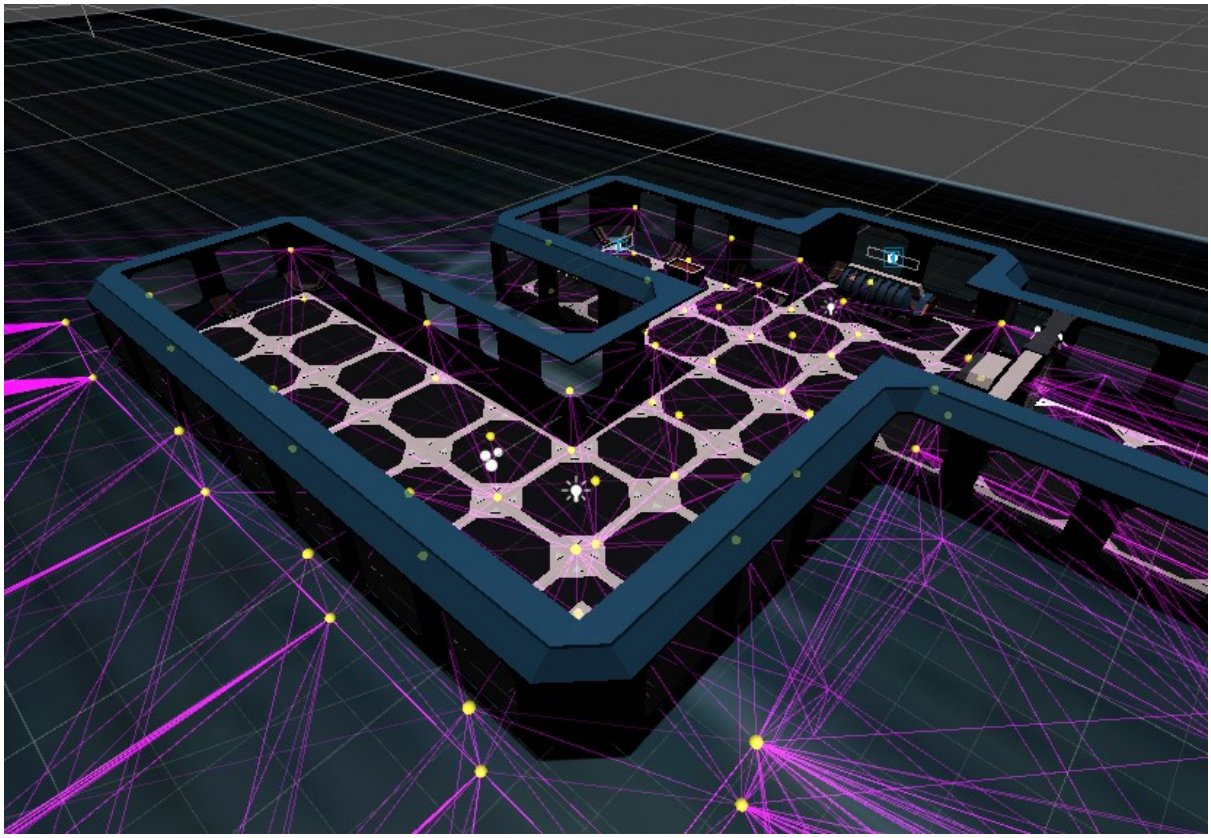
Za osvjetljenje velikog dijela mape korištena je tehnika pečenog osvjetljenja. Ta tehnika omogućuje prijevremenu kalkulaciju svjetla te to sprema kao podatke. Prilikom igranja Unity koristi te podatke kako bi osvijetlio mapu. Zato što su kompleksne kalkulacije napravljene prijevremeno, pečenje svjetla smanjuje troškove sjenčanja tijekom izvođenja i smanjuje troškove renderiranja sjena.

16.1.1. Svjetleće probe (Light Probes)

Svjetleće probe su ključna komponenta u Unityjevom sustavu osvjetljenja. One snimaju i koriste informacije o tome kako svjetlo prolazi kroz prazan prostor u sceni.

Dok se svjetlosnim mapama pohranjuju podaci o sceni gdje se svjetlo mjeri tijekom postupka pečenja svjetla, ove probe koriste se za približno određivanje neizravnog svjetla na dinamičke objekte na temelju njihove blizine najbližim probama [23]. Prikaz svjetlosnih proba u sceni nalazi se na slici 95.
Svjetleće probe imaju dvije primarne svrhe:

- Primarna svrha je pružiti visokokvalitetno osvjetljenje (uključujući neizravnu odbijenu svjetlost) na pokretnim objektima u sceni
- Sekundarna svrha je pružanje informacije o osvjetljenju za statične objekte koji koriste Unityjev sustav za razinu detalja (LOD)



Slika 94. Prikaz svjetlosnih proba u sceni

16.1.2. Promjena svjetla

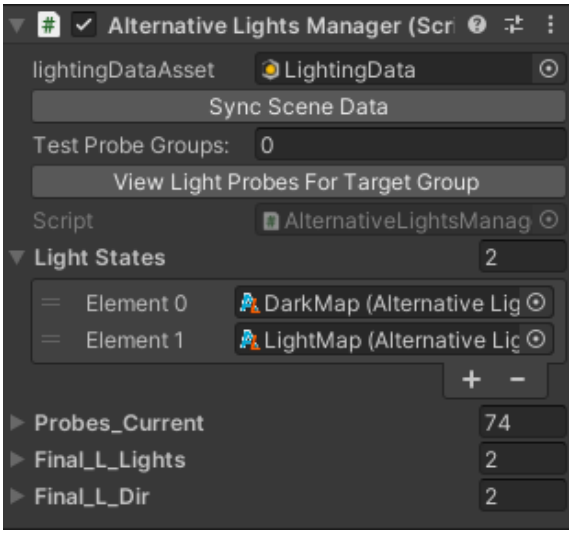
U Unityju ne postoji ugrađena opcija za promjenu pečenog osvjetljenja tijekom same igre. Međutim, pružena je mogućnost zamjene svjetlosnih mapa i podataka svjetlosnih proba kako bi se omogućila dinamička promjena ovih podataka.

Da bi se postigla ta dinamika, koristi se alat za zamjenu svjetlosnih mapa koji je napravio Birdmask Studio [24]. Taj alat omogućuje promjenu svjetlosnih mapa i podataka svjetlosnih proba prema potrebi.

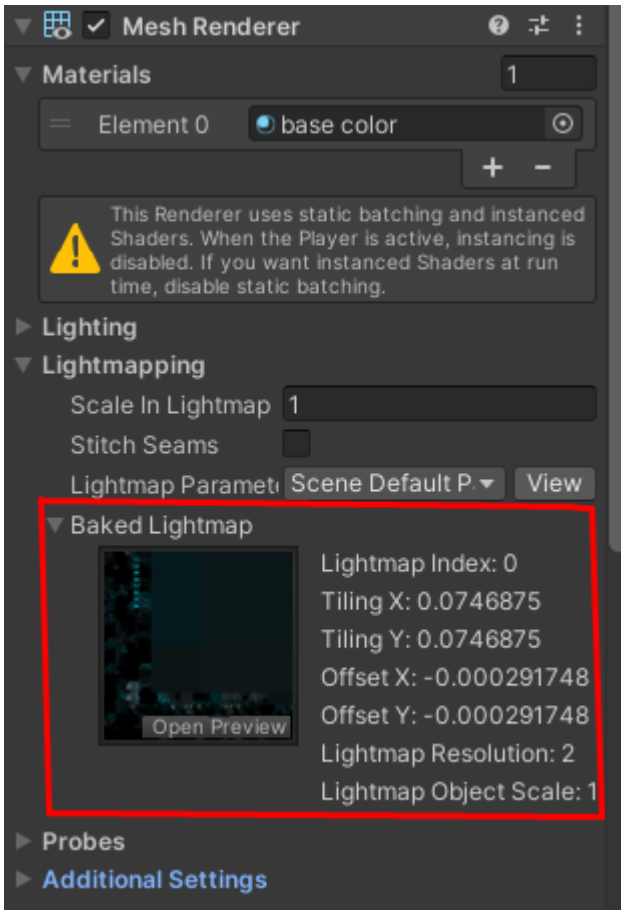
Ovisno o željama svjetlosnih efekata, igra zahtjeva različite podatke o osvjetljenju. Primjerice, za Stellar Decay potrebne su dvije vrste mapa, jedna za tamne uvjete i druga za svijetle. Tijekom igranja, igrači mogu dinamički mijenjati pojedine dijelove tih mapa iz tamnih u svijetle.

Alternative Light Manager skripta, prikazana na slici 96, sadrži informacije o svjetlosnim mapama koje želimo koristiti i broju svjetlosnih proba prisutnih u igri. Objekti u igri mogu imati komponentu MeshRenderer, prikazanu na slici 97, koja sadrži teksture pečenog osvjetljenja. Tijekom igranja može se mijenjati indeks tih tekstura. Problem je što Unity ima samo jedan skup podataka o osvjetljenju prilikom pokretanja igre. No, Alternative Light Manager stvara

listu tih podataka pri pokretanju i stavlja ih na definirane vrijednosti. Također zamjenjuje trenutne podatke o osvjetljenju s novom listom svjetlosnih podataka. To omogućuje promjenu indeksa svjetlosnih mapa, što rezultira promjenom tekstura objekta.



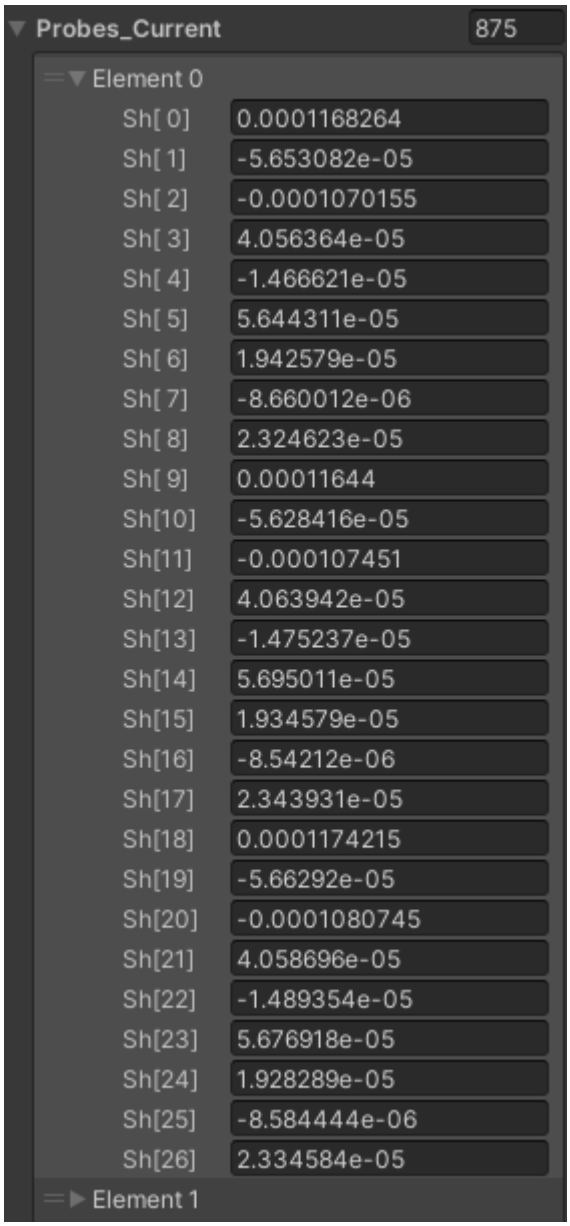
Slika 95. Prikaz Alternative Light Manager komponente



Slika 96. Označena trenutno korištena svjetlosna tekstura nekog objekta

Promjenom indeksa nekog objekta svjetlosne probe se neće promijeniti. One su samostalne komponente koje sadrže podatke o osvjetljenju. Alternative Light Manager ima sve podatke o tim probama, kao što je prikazano na slici 98, tako da se može poslati listu proba koje je potrebno izmijeniti te će se one izmijeniti ovisno o tome koja je svjetlosna opcija selektirana.

AlternateBakedLight je dodatna skripta koja se nalazi na objektima nad kojima se želi izvršiti promjena svjetla. Glavna funkcija ChangeLightState, prikazana na slici 99, omogućuje promjenu svjetlosnih tekstura na objektima. Ta se promjena postiže promjenom indeksa svjetlosne mape koja će se koristiti. Međutim, u originalnoj skripti ta je promjena bila ograničena samo na objekt nad kojim je bila ova skripta.



Slika 97. Prikaz jednog elementa iz liste svjetlosnih proba iz Alternative Light Manager komponente

```
public void ChangeLightState(int Value)
{
    //Make sure target lightmap texture is in the valid range
    Value = Mathf.Clamp(Value, 0, manager.maxStatesCount);
    currentLightState = Value;

    //Call the manager to change the selected lightprobes settings
    manager.AssignLightProbesSegment(linkedLightProbes, currentLightState);

    //Switch to a diffrent lightmap texture
    rend.lightmapIndex = currentLightState;
}
```

Slika 98. Orginalna funkcija za promjenu svjetlosnih proba i svjetlosnih tekstura

Za Stellar Decay bilo je potrebno mijenjati svjetlosne podatke na velikom broju objekata, tako da se ChangeLightState funkcija trebala prilagoditi kao što je prikazano na slici 100. Sada, umjesto da se mijenja svjetlo samo na trenutnom objektu, skripta prolazi kroz sve MeshRenderer komponente koje se nalaze u djeci objekta. Za svaku od tih komponenti, izračunava se odgovarajući indeks svjetlosne mape koji se mora koristiti te se zatim taj indeks postavlja kako bi se promijenila svjetlosna tekstura.

```
public void ChangeLightState(int Value)
{
    //Make sure target lightmap texture is in the valid range
    Value = Mathf.Clamp(Value, 0, manager.maxStatesCount);
    currentLightState = Value;

    //Call the manager to change the selected lightprobes settings
    manager.AssignLightProbesSegment(linkedLightProbes, currentLightState);

    //Switch to a diffrent lightmap texture

    int skipCount = manager.l_light.Length/manager.lightStates.Length;
    int startIndex = skipCount * currentLightState;

    foreach (Renderer rend in renderers)
    {
        int selectedIndex = (rend.lightmapIndex % skipCount)+ startIndex;

        rend.lightmapIndex = selectedIndex;
    }
}
```

Slika 99. Nova funkcija za promjenu svjetlosnih proba i svjetlosnih tekstura

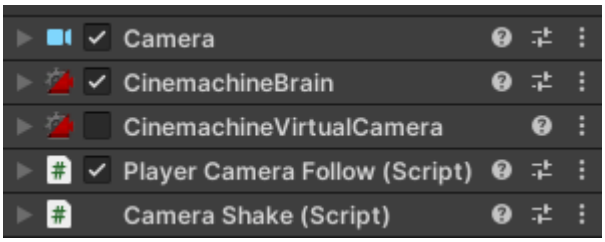
16.2. Kamera i praćenje igrača

Postoje dva različita objekta, jedan je prazan objekt kojemu je svrha da konstantno prati igrača i na njemu se nalazi PlayerFollow skripta. Slika 101 prikazuje Update funkciju od PlayerFollow skripte.

```
3 references
public Transform follow;

// Update is called once per frame
0 references
void Update()
{
    if(follow != null)
        transform.position = follow.position;
}
```

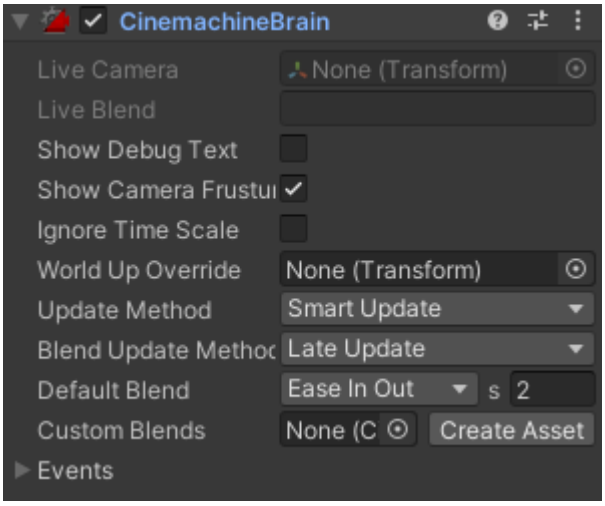
Slika 100. Update funkcija iz PlayerFollow skripte

Drugi objekt je sama kamera koja koristi Cinemachine sustav za upravljanje kamerama. Cinemachine rješava kompleksne matematičke i logičke izazove praćenja neke mete, u ovom slučaju igrača, ili prebacivanje između kadrova. Dizajniran je s ciljem značajnog smanjenja broja dugotrajnih ručnih manipulacija i revizija skripti koje se događaju tijekom razvoja [26]. Slika 102 prikazuje sve komponente koje se nalaze na glavnoj kameri.



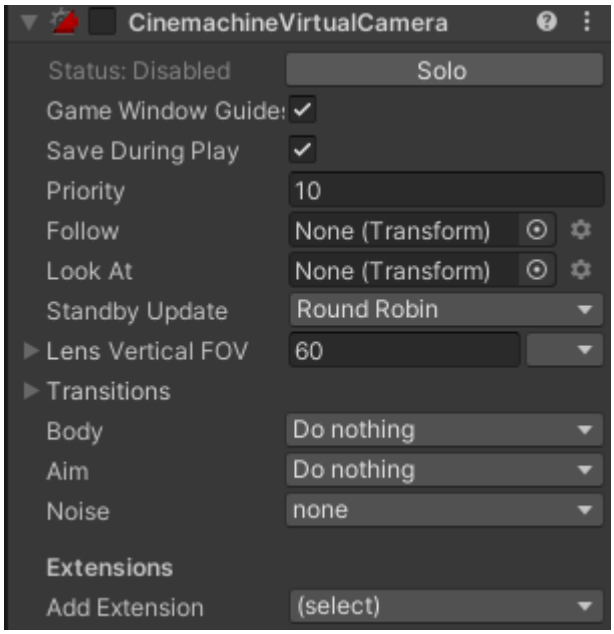
Slika 101. Prikaz komponenti na glavnoj kameri

Cinemachine kontrolira kameru kroz Cinemachine Brain komponentu prikazanoj na slici 103, koja se automatski dodaje na kameru kada se napravi Cinemachine kamera. Glavna poanta Cinemachine Brain komponente je da kontrolira glavnu kameru jer u sceni može biti više virtualnih kamera, ali one same po sebi ne rade ništa već je jedna od njih aktivna. Cinemachine Brain zna koja je virtualna kamera aktivna te prema njoj pomiče kameru.



Slika 102. Prikaz CinemachineBrain komponente na kameri

Virtualna kamera (Cinemachine Virtual Camera), prikazana na slici 104, je komponenta koja se dodaje na objekt. Korištenjem Aim, Body i Noise svojstava može se specificirati kako će virtualna kamera animirati poziciju, rotaciju i ostala svojstva. Kada je Virtualna kamera korištena, postavlja ta svojstva glavnoj kameri na kojoj se nalazi Cinemachine Brain.



Slika 103. Prikaz CinemachineVirtualCamera komponente na kameri

PlayerCameraFollow skripta, prikazana na slici 105, ima reference na Stats komponentu trenutnog igrača kojeg prati. Provjerava njegove životne bodove svake sekunde te pronalazni novog igrača kojeg treba pratiti ukoliko su životni bodovi trenutnog igrača manji ili jednaki 0.

```
private Stats followTargetStats;

0 references
private void Start()
{
    InvokeRepeating("CheckPlayerHP", 5f, 1f);
}

0 references
private void CheckPlayerHP()
{
    if (followTargetStats.hp.Value <= 0)
    {
        FollowPlayer(GameManager.Instance.allPlayers[Random.Range(0, GameManager.Instance.allPlayers.Count)].transform);
    }
}
```

Slika 104. Start i CheckPlayerHP funkcije iz PlayerCameraFollow skripte

Slika 106 prikazuje FollowPlayer funkciju koja se poziva prilikom smrti igrača ili prilikom ponovnog kreiranja igrača. U igri bi trebao biti samo jedan aktivan AudioListener, tako da funkcija pronalazi i deaktivira AudioListener koji se nalazi na umrlom igraču. Funkcija postavlja novu Stats referencu te uključuje AudioListener komponentu na trenutnom igraču kojeg prati. Zatim stavlja metu koju će PlayerFollow skripta pratiti te aktivira virtualnu kameru.

AudioListener komponenta se može zamisliti kao uši igrača, koje omogućuju da igrač čuje zvukove oko sebe.

```
public void FollowPlayer(Transform followTransform)
{
    try
    {
        followTargetStats.gameObject.GetComponentInChildren<AudioListener>().enabled = false;
    }
    catch
    {
        Debug.Log("No audio listener exists");
    }
    followTargetStats = followTransform.GetComponent<Stats>();
    try
    {
        followTargetStats.gameObject.GetComponentInChildren<AudioListener>().enabled = true;
    }
    catch
    {
        Debug.Log("No audio listener exists");
    }
    playerFollow.follow = followTransform;
    cinemachineVirtualCamera.Follow = playerFollow.transform;
    cinemachineVirtualCamera.LookAt = playerFollow.transform;
    cinemachineVirtualCamera.enabled = true;
}
```

Slika 105. FollowPlayer funkcija iz PlayerCameraFollow skripte

Zadnja komponenta na kameri je CameraShake skripta. Poziva se kada igrač primi neku štetu te stvara malo drhtanje kamere koje traje neko određeno vrijeme. Cinemachine kamera ima opciju noise koju ova skripta iskorištava za izradu efekta drhtanja kamere.

```
public void Shake(float intensity, float stopTime){

    cinemachineBasicMultiChannelPerlin.m_AmplitudeGain = intensity;

    CancelInvoke();
    Invoke("ShakeOver",stopTime);
}

0 references
private void ShakeOver(){
    cinemachineBasicMultiChannelPerlin.m_AmplitudeGain = 0;
}
```

Slika 106. Shake i ShakeOver funkcije iz CameraShake skripte

17. Zvukovi

Zvukovi u video igrama igraju ključnu ulogu u stvaranju dublje imerzije za igrače. Služe za uspostavljanje atmosfere, pružaju informacije igraču i daju mu upozorenja.

Kada igrač pokrene igru i dođe do glavnog izbornika čuje se glazba. Skladba koja svira u glavnom izborniku je The Door iz serije Chernobyl skladateljice Hildur Guðnadóttir [30].

U igri se nalazi mnoštvo UI elemenata koji se mogu pritisnuti. Prilikom pritiska igrača na bilo koji od ovih elemenata čuje se zvuk koji je preuzet s Unity Asset Store[29].

Tijekom igre akcije igrača ili neprijatelja popraćene su s raznoraznim zvukovima. Ove akcije uključuju kretanje igrača ili neprijatelja, aktiviranje generatora, pucanje iz raznoraznih oružja, ponovno punjenje oružja, pucanje s praznim magazinom i slično. Oružja koriste svoje jedinstvene zvukove kada pucaju. Zvukovi kretanja [31] i pucanja [32] preuzeti su s Unity Asset Store.

Kada igračevi životni bodovi padnu do određene razine počine svirati zvuk koji treba obavijestiti igrača da će uskoro umrijeti ako primi još par udaraca od neprijatelja. Zvuk koji označava skoru smrt je otkucaj srca [33].

Generatori koriste dva zvuka. Prvi zvuk koji generator koristi je zvuk pisaće mašine koji služi kao zvuk pokretanja generatora [34]. Kada generator završi s aktivacijom pokrene se drugi zvuk [35]. Zvukovi generatora se mogu čuti kada je igrač blizu generatora u igri.

Osim zvučnih efekata u Stellar Decay nalaze se i zvukovi snimljeni od glasovnih glumaca. Ovi zvukovi uključuju naraciju koja se pojavljuje u igri koja govori igraču što je potrebno napraviti i što se dogodi kada se generator aktivira i zvukove zombija.

18. Zaključak

Kroz ovaj rad opisana je izrada kooperativne pucačine iz ptičje perspektive u Unity okruženju. Opisana je većina elemenata koje je bilo potrebno izraditi prilikom izrade Stellar Decay igre, kao što su oružja koje igrač može koristiti, objekti s kojima igrač može interaktirati, neprijatelji koje igrač mora ubijati te objekti koje igrač mora pokrenuti kako bi napredovao u igri.

Opisane su tehnologije i alati koje je bilo potrebno koristiti za izradu kooperativne igre, kao i implementacije nekih od tih tehnologija i alata kao što su Unity Relay, Unity Netcode for GameObjects, Unity Lobby sistem. Opisani su cjevovodi renderiranja koji postoje u Unityju te je rečeno koji se cjevovod koristi u izradi Stellar Decayja. Opisan je glavni izbornik igre, izbornik tijekom igre i izbornik nakon što igrač umre. Detaljno je opisan igrač, njegova sposobnost, skripte i komponente koje se na njemu koriste. Zatim je opisan glavni neprijatelj igrača, rad komponenti koje se nalaze na njemu, kao i oružja i granate, njihov rad i njihova sposobnost promjene. Također su opisani generatori i škrinje. Nadalje, objašnjeno je kako se i zašto mijenja svjetlo tijekom igre. Na kraju je objašnjena kamera, njene komponente i njen način rada.

Stellar Decay se može poboljšati s dosta stvari koje bi igru činile zanimljivijom. Primjer jedne od tih stvari je raznovrsnost neprijatelja, trenutno postoji samo jedan neprijatelj koji napada igrača. Bilo bi zanimljivo dodati nove neprijatelje s drugačijim napadima. Također je igra trenutno previše linearna. To bi se moglo promijeniti dodavanjem mogućnosti penjanja na više razine (stepenice) koje bi igračima dale mogućnost gledanja mape s veće visine te dodavanjem dodatnih objekata koje igrač mora napraviti kao na primjer spasiti neke od preživjelih osoba prije nego što ih zombiji ubiju. Još jedan primjer bio bi promjena sistema za granate. Trenutno se pune nakon određenog vremena, dok bi ih nakon promjene igrač trebao pronaći i pokupiti na nekim dijelovima mape.

Trenutna optimizacija Stellar Decayja grana se na svjetlosnu optimizaciju i GPU skupljanje (GPU batching). Svjetlosna optimizacija koristi svjetlosne teksture koje su pečene i ne trebaju se ponovno kalkulirati. GPU skupljanje je opcija na materijalima u Unityju, gdje se više objekata koji imaju slična svojstva grupiraju te zajedno kalkuliraju.

Stellar Decay bi se dodatno mogao optimizirati promjenom opcija cjevovoda renderiranja. Prvenstveno, implementacijom višenitnih funkcionalnosti kako bi se neki procesi kalkulacija prebacili na druge niti.

Tijekom izrade ovog rada naučene su nove stvari poput rada s osvjetljenjem u Unity okruženju te Networkingom, gdje je Unity Netcode olakšao razumijevanje tih koncepata i olakšao implementaciju igre s više ljudi.

Sažetak

Tema ovog diplomskog rada je izrada pucačine iz ptičje perspektive u Unity okruženju s fokusom na razvoj funkcionalnosti za više igrača. Rad detaljno opisuje korištenje Unity Relay usluge za olakšavanje komunikacije putem interneta, Unity Netcode alate za implementaciju mrežnih funkcija i Unity Lobby koncepta za organizaciju igara za više igrača unutar igre.

Uz to, rad istražuje i opisuje korisničko sučelje (UI) unutar igre, pružajući uvid u dizajn i funkcionalnosti. Skripte i objekti bitne za funkcionalnost igre detaljno su opisani uključujući opise njihovih uloga i zadataka u igri.

Rad dodatno istražuje dinamičke promjene svjetla tijekom igranja igre, pružajući objašnjenje kako i zašto se svjetlosni efekti mijenjaju.

Ključne riječi: Unity, pucačina iz ptičje perspektive, videoigra, Unity Netcode, Unity Relay, Unity Lobby, više igrača, korisničko sučelje

Abstract

The topic of this master's thesis is the development of a top-down shooter in the Unity environment with a focus on multiplayer functionality. The thesis provides a detailed description of the Unity Relay service use to facilitate internet communication, Unity Netcode tools for implementing network features, and the Unity Lobby concept for organizing multiplayer games within the game.

Additionally, the thesis explores and describes the user interface (UI) within the game, offering insights into its design and functionalities. Scripts and objects essential for the game's functionality are thoroughly documented and include descriptions of their roles and tasks in the game.

Furthermore, the thesis investigates dynamic changes in lighting during gameplay, providing an explanation of how and why lighting effects are altered.

Keywords: Unity, top-down Shooter, video game, Unity Netcode, Unity Relay, Unity Lobby, multiplayer, user interface

Literatura

1. Unity Technologies, “Manual: Render pipelines introduction ”, [Online]. Dostupno: https://docs.unity3d.com/Manual/render-pipelines-overview.html
2. Unity Technologies, “Unity Lobby Service”, [Online]. Dostupno: https://docs.unity.com/ugs/en-us/manual/lobby/manual/unity-lobby-service
3. Unity Technologies, “Unity Relay”, [Online]. Dostupno: https://docs.unity.com/ugs/en-us/manual/relay/manual/introduction
4. Unity Technologies, “NetworkManager”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/components/networkmanager/
5. Unity Technologies, “NetworkObject”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/
6. Unity Technologies, “NetworkTransform”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/
7. Unity Technologies, “NetworkVariable”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/index.html
8. Unity Technologies, “About Netcode for GameObjects”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/about/
9. Unity Technologies, “Ticks and update rates”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/learn/ticks-and-update-rates/
10. Alien Swarm Wiki, “Alien Swarm Reactive Drop”, [Online]. Dostupno: https://alienswarm.fandom.com/wiki/Alien_Swarm_Reactive_Drop
11. Wikipedia, “Darkwood”, [Online]. Dostupno: https://en.wikipedia.org/wiki/Darkwood
12. Wikipedia, “Call of Duty: World at War”, [Online]. Dostupno: https://en.wikipedia.org/wiki/Call_of_Duty:_World_at_War#Nazi_Zombies
13. Unity Technologies, “Manual: Line Renderer component”, [Online]. Dostupno: https://docs.unity3d.com/Manual/class-LineRenderer.html
14. Unity Technologies, “NetworkBehavior”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkbehavior/
15. Unity Technologies, “NetworkAnimator”, [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/components/networkanimator

16. SimpleColor Studios, “SS1_Bandit_WeaponsPackage_PBR”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/props/weapons/ss1-bandit-weaponspackage-pbr-216132#content
17. Mohammed Qamar, “Road Flares”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/props/road-flares-209209
18. Leandro Melchior, “Steampunk chest”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/props/steampunk-chest-69723
19. Artalasky, “Modern Zombie Free”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/characters/humanoids/modern-zombie-free-58134
20. AlexMakes3D, “Futuristic soldier - Scifi character”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/characters/humanoids/sci-fi/futuristic-soldier-scifi-character-202085
21. Fun Assets, “Guns Pack: Low Poly Guns Collection”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/props/guns/guns-pack-low-poly-guns-collection-192553
22. karboosx, “Sci-Fi Styled Modular Pack”, [Online]. Dostupno: https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-styled-modular-pack-82913
23. Unity Technologies, “Manual: Light Modes”, [Online]. Dostupno: https://docs.unity3d.com/Manual/LightModes.html
24. Birdmask Studio, “Swapping lightmaps in realtime”, [Online]. Dostupno: https://birdmaskstudio.itch.io/unity3d-swapping-lightmaps-in-realtime
25. Unity Technologies, “Manual: Navigation System in Unity”, [Online]. Dostupno: https://docs.unity3d.com/Manual/nav-NavigationSystem.html
26. Unity Technologies, “Cinemachine Documentation”, [Online]. Dostupno: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/index.html
27. Unity Technologies, “The Cinemachine Brain”, [Online]. Dostupno: https://learn.unity.com/tutorial/the-cinemachine-brain#
28. Unity Technologies, “Setting Virtual Camera properties ”, [Online]. Dostupno: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.9/manual/CinemachineVirtualCamera.html
29. Little Robot Sound Factory, “UI Sfx“, [Online]. Dostupno: https://assetstore.unity.com/packages/audio/sound-fx/ui-sfx-36989

30. Hildur Guðnadóttir, “The Door“, [Online]. Dostupno: https://www.youtube.com/watch?v=QC1aBVDsoWw
31. Nox_Sound, “Footsteps - Essentials“, [Online]. Dostupno: https://assetstore.unity.com/packages/audio/sound-fx/foley/footsteps-essentials-189879
32. Mikael Venninen, “Sci-fi Guns SFX Pack“, [Online]. Dostupno: https://assetstore.unity.com/packages/audio/sound-fx/sci-fi-guns-sfx-pack-181144#description
33. Placidplace, “heartbeat“, [Online]. Dostupno: https://pixabay.com/sound-effects/search/heartbeat/
34. gamesounds, “typewriter-return-1.wav“, [Online]. Dostupno: https://gamesounds.xyz/?dir=Sound%20Effects/Tools
35. gamesounds, “Generator Hum.wav“, [Online]. Dostupno: https://gamesounds.xyz/?dir=BBC%20Sound%20Effects%20Library/BBC%2019%20-%20Electronically%20Generated%20Sounds

Stellar Decay GitHub: https://github.com/CroBanana/Stellar-Decay.git

Popis slika

Slika 1. Snimka ekrana iz igrice Darkwood 2
Slika 2. Snimka ekrana iz zombi moda Call of Duty: World at War 3
Slika 3. Snimka ekrana iz Alien Swarm: Reactive Dropa 4
Slika 4. Prikaz Render Pipeline Converter prozora u Unityju 6
Slika 5. Prikaz problema nepostojeće teksture 7
Slika 6. CreateRelay funkcija iz TestRelay skripte 8
Slika 7. JoinRelay funkcija iz TestRelay skripte 8
Slika 8. Start funkcija iz LobbySystem skripte 10
Slika 9. CreateLobby funkcija iz LobbySystem skripte 11
Slika 10. ListLobbies funkcija iz LobbySystem skripte 12
Slika 11. JoinLobbyByCode funkcija iz LobbySystem skripte 12
Slika 12. KickPlayer funkcija iz LobbySystem skripte 13
Slika 13. Prikaz vremena ažuriranja u ovisnosti stope ažuriranja [9] 14
Slika 14. Prikaz NetworkManager komponente 16
Slika 15 Prikaz svih Network objekata 16
Slika 16. Prikaz NetworkObject komponente na objektu 17
Slika 17. TakeDmgServerRpc funkcija iz Stats skripte 18

Slika 18. Prikaz NetworkVariable hp iz Stats skripte 19
Slika 19.NetworkTransform komponenta na objektu 19
Slika 20. Prikaz izbornika za upis imena 21
Slika 21. Prikaz glavnog izbornika 21
Slika 22. Prikaz kreiranja nove sobe 22
Slika 23. Prikaz izgleda sobe 22
Slika 24. Prikaz aktivnih soba 23
Slika 25. Prikaz upisa koda za pridruživanje privatnoj sobi 23
Slika 26. Prikaz izbornika za opcije 24
Slika 27. Prikaz kontrola igrača tijekom igre 24
Slika 28. Prikaz informacija o igri 25
Slika 29. Prikaz korisničkog sučelja u igri 25
Slika 30. Prikaz životnih bodova igrača 26
Slika 31. Sučelje nakon smrti igrača 26
Slika 32. Sučelje nakon smrti svih igrača 27
Slika 33. Prikaz sučelja za opcije tijekom igre 27
Slika 34. Prikaz svih komponenata na igraču 28
Slika 35. Prikaz Transform komponente na igraču 28
Slika 36. Prikaz Animator komponente na igraču 28
Slika 37. Prikaz Base Layer sloja 29
Slika 38. Prikaz Top Body Layer sloja 30
Slika 39. Prikaz maske koja se nalazi na Top Body i Hands Shoot Rifle slojevima 30
Slika 40. Prikaz Hands Shoot Rifle sloja 31
Slika 41. Prikaz Character Controller komponente s Player objekta 31
Slika 42. Prikaz Rig Builder i Bone Renderer komponenata s Player objekta 32
Slika 43. Prikaz Add Player komponente s Player objekta 32
Slika 44. Update funkcija iz Player skripte 33
Slika 46. Movement funkcija iz Player skripte 33
Slika 47. Move funkcija iz Player skripte 34
Slika 48. PlayerInteraction komponenta na Player objektu 34
Slika 49. Update funkcija iz PlayerInteract skripte 35
Slika 50. Interact funkcija iz PlayerInteract skripte 35
Slika 51. Prikaz funkcija za kreiranje zvuka iz AudioPlayer skripte 35
Slika 52. PlayerZombieNoise funkcija iz AudioPlayer skripte 36
Slika 53. Mrežne varijable hp i playerName iz Stats skripte 36
Slika 54. OnNetworkSpawn funkcija iz Stats skripte 37
Slika 55. UpdateHpImage funkcija iz Stats skripte 38

Slika 56. Funkcije za slanje štete iz Stats skripte 38
Slika 57. Prikaz PlayerDead komponente na Player objektu 39
Slika 58. Start, ActivateUI i DisableRagdoll funkcije iz PlayerDead skripte 39
Slika 59. DisableComponents, Dead i ActivateRagdoll funkcija iz PlayerDead skripte 40
Slika 60. RespawnAfter korutina iz PlayerDead skripte 40
Slika 61. Start funkcija iz PlayerGuns skripte 41
Slika 62. Prvi dio Update funkcije iz PlayerGuns skripte 42
Slika 63. Drugi dio Update funkcije iz PlayerGuns skripte 42
Slika 64. Treći dio Update funkcije iz PlayerGuns skripte 43
Slika 65. Četvrti dio Update funkcije iz PlayerGuns skripte 44
Slika 66. NewWeapon funkcija iz PlayerGuns skripte 45
Slika 67. Prikaz ClientNetworkTransform komponente na oružju 46
Slika 68. Prikaz Weapon komponente na oružju 47
Slika 69. Update funkcija iz Weapon skripte 48
Slika 70. Shoot funkcija iz Weapon skripte 49
Slika 71. ShootRaycast funkcija i dvije funkcije vezane za sinkronizaciju pucanja 50
Slika 72. SpawnTrailAndDMG funkcija iz Weapon skripte 51
Slika 73. Prikaz svih komponenata na Zombi objektu 52
Slika 74. Prikaz Animator komponente na Zombi objektu 52
Slika 75. Prikaz Baze Layer sloja Zombie kontrolera 52
Slika 76. Prikaz Hands Attack sloja Zombie kontrolera 53
Slika 77. Prikaz AudioPlayer komponente sa Zombi objekta 53
Slika 78. Prikaz NavMeshAgent komponente sa Zombi objekta 54
Slika 79. FollowPlayer funkcija iz Zombie skripte 55
Slika 80. Update funkcija iz Zombie skripte 55
Slika 81. Prikaz svih komponenti na škrinji za puške 56
Slika 82. Trigger funkcije iz SpawnGun skripte 56
Slika 83. Interact i OpenCrateServerRpc funkcije iz SpawnGun skripte 57
Slika 84. Rotate i NewGunSpawned funkcije iz SpawnGun skripte 57
Slika 85. Prikaz InteractableText komponente na škrinji 58
Slika 86. Prikaz teksta za interakciju i ikone za interakciju tijekom igre 58
Slika 87. Prikaz svih komponenata na generatoru 58
Slika 88. Prikaz PowerGeneratorV2 komponente na generatoru 59
Slika 89. Trigger funkcije iz PowerGeneratorV2 skripte 60
Slika 90. Interact i StartGeneratorServerRpc funkcije iz PowerGeneratorV2 skripte 60
Slika 91. GeneratorStarting korutina iz PowerGeneratorV2 skripte 61
Slika 92. Explode funkcija iz Grenade skripte 61

Slika 93. Prikaz objekata koji sadrže sve skupine u jednom objektu 62
Slika 94. Snimka ekrana, prikaz igrive mape 62
Slika 95. Prikaz svjetlosnih proba u sceni 64
Slika 96. Prikaz Alternative Light Manager komponente 65
Slika 97. Označena trenutno korištena svjetlosna tekstura nekog objekta 65
Slika 98. Prikaz jednog elementa iz liste svjetlosnih proba iz Alternative Light Manager komponente 66
Slika 99. Orginalna funkcija za promjenu svjetlosnih proba i svjetlosnih tekstura 67
Slika 100. Nova funkcija za promjenu svjetlosnih proba i svjetlosnih tekstura 67
Slika 101. Update funkcija iz PlayerFollow skripte 68
Slika 102. Prikaz komponenti na glavnoj kameri 68
Slika 103. Prikaz CinemachineBrain komponente na kameri 69
Slika 104. Prikaz CinemachineVirtualCamera komponente na kameri 69
Slika 105. Start i CheckPlayerHP funkcije iz PlayerCameraFollow skripte 70
Slika 106. FollowPlayer funkcija iz PlayerCameraFollow skripte 70
Slika 107. Shake i ShakeOver funkcije iz CameraShake skripte 71