Razvoj borbene igre za više igrača primjenom Unity okruženja i Netcode biblioteke

Vila, Leon

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:223607

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-28**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

Leon Vila

RAZVOJ BORBENE IGRE ZA VIŠE IGRAČA PRIMJENOM UNITY OKRUŽENJA I NETCODE BIBLIOTEKE

Diplomski rad

Pula, veljača 2023. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

Leon Vila

RAZVOJ BORBENE IGRE ZA VIŠE IGRAČA PRIMJENOM UNITY OKRUŽENJA I NETCODE BIBLIOTEKE

Diplomski rad

JMBAG: 0316001376
Studijski smjer: Informatika
Kolegij: Dizajn i programiranje računalnih igara
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, veljača 2023. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Leon Vila, 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 nedozvoljen 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
Leon Vila

U Puli, veljača 2023. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, dolje potpisani Leon Vila, ovime dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom "Razvoj borbene igre za više igrača primjenom Unity okruženja i Netcode biblioteke" koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišta Jurja Dobrile u Puli te kopira 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.

Student
Leon Vila

U Puli, veljača 2023. godine

SAŽETAK

Ovaj diplomski rad ima za cilj objasniti proces kreiranja jednostavne računalne igre korištenjem Unity okruženja i nove Netcode biblioteke za umrežavanje. Pokazat će kako stvoriti novi projekt pomoću Unity editora. Potom će se baviti okruženjem i likovima na sceni. Objasnit će kako funkcioniraju sustavi koji kontroliraju tijek igre. Zatim će pokazati kako se unosi s tipkovnice pretvaraju u pokrete u igri. Demonstrirat će principe kreiranja izbornika i upravljanja zvukom. I na samom kraju, ispitat će bitne funkcionalnosti Netcode biblioteke i primijeniti ih u radu unutar projekta.

This master's thesis aims to explain the process of creating a simple computer game using the Unity environment and the new Netcode library for networking. It will show how to create a new project using the Unity editor. Afterward, it will deal with the environment and characters on the scene. It will explain how the systems that control the flow of a game works. It will then show how keyboard inputs translate into in-game movements. It will demonstrate the principles of menu creation and sound management. And at the very end, it will examine the essential functionalities of the Netcode library and apply them to work within the project.

SADRŽAJ

1 Uvod....1

2 Motivacija....2

2.1 PC igre....2

2.2 Žanr....2

2.3 Srodne igre....3

3 Prvi koraci u izradi projekta....5

3.1 Novi projekt u Unity-u....5

3.2 Upravljanje verzijama....8

4 Izrada okoline....10

4.1 Teren....10

4.2 3D modeli....11

4.3 Likovi....12

4.4 Animacije na likovima....13

4.5 Animator....13

4.6 Primjer ciklusa kontrolera....15

4.7 Slojevi animatora....16

5 Sustav borbe....17

5.1 Borci....17

5.2 Izvedba udaraca....18

5.3 Blok i micanje....19

5.4 Registriranje udaraca....20

6 Kontroler inputa....23

6.1 Igračev kontroler....23

6.2 AI kontroler....24

7 Upravitelj igre....25

7.1 Instanciranje likova....26

7.2 Započinjanje borbe....26

7.3 Sinkronizacija života i stamine....27

7.4 Tranzicije između borbi....28

7.5 Pauziranje borbe 29
7.6 Čišćenje scene 29
8 Izbornik 30
8.1 Sprites 30
8.2 Kanvas 31
8.3 Mijenjanje prozora izbornika 32
8.4 Animacije tranzicija prozora izbornika 34
8.5 Odabir likova 35
8.6 Vodič 36
9 Uvodna priča 37
9.1 Tekst 37
9.2 Pozadina 38
10 Audio 39
10.1 Audio upravitelj 39
10.2 Audio Mixer 40
11 Netcode 42
11.1 NetworkManager 42
11.2 Sinkronizacija objekata 43
11.3 Sinkronizacija animacija 44
11.4 Spajanje i započinjanje borbe 45
11.5 Mrežne varijable 46
11.6 RPCs 47
12 Zaključak 49
REFERENCE 50
POPIS SLIKA 52

1 Uvod

Video igre su po svojoj prilici postale vrlo utjecajan oblik zabave. Naime, ako je vjerovati estimacijama svjetskih medijskih prihoda za 2020 godinu [1], industrija video igara zauzima drugo mjesto - odmah iza filmske industrije koja je na prvom. Nadalje, Amazonova live streaming platforma Twitch koja je fokusirana na streamanje video igara kontinuirano registrira 1.2 bilijuna posjeta mjesečno [2] što je čini jednom od najposjećenijih web stranica na svijetu. Koliko su igre raširene u mlađim generacijama govori i činjenica da postoji "Gaming" sekcija na mrežnim divovima kao što su Facebook i YouTube.

Iz priloženog vidimo da postoji interes i market za izradu igara, no izrada igara nije jednostavna stvar. Zbog količine i spektra poslova, na igri obično radi tim ljudi - studio. Neke od ključnih potrebnih uloga za uspješnu realizaciju igre su dizajner, programer, audio dizajner, animator. Svaku od njih možemo rastaviti na mnogo užih specijaliziranijih uloga. Na primjer, programeri mogu biti podijeljeni na mrežne, gameplay, AI programere, arhitekte sustava, game engine inženjere i slično. Kako bi se višestruko smanjila količina programerskog posla, mnogi se odlučuju na korištenje već gotovih programskih okvira koji se nazivaju game engine, od kojih su trenutno na tržištu najpopularniji Unreal Engine i Unity.

Eksternalizacija posla (tzv. outsourcing) digitalne prirode je u današnje internetsko doba lakša nego ikad. Krajem prvog desetljeća 2000-te godine počeli su nicati mnogi web distribucijski servisi koji nude kreatorima prodaju svojih autorskih tvorevina poput 2D i 3D modela, UI elemenata, audio datoteka, animacija, efekata, pa čak i programskih rješenja. Takav lagan pristup potrebnim assetima omogućilo je pojavu neovisnih višenamjenskih game developera. Projekti nastali takvim kupljenim assetima su pretežito generične prirode, ograničeni dostupnim assetima, loše optimizirani. Ali su svakako zabavno i korisno iskustvo za učenje i dublje shvaćanje interakcije između računala i multimedijalnih sadržaja. Iz istog razloga u ovom radu će biti napravljena igra koristeći Unity engine i dostupne assete. Svrha rada nije pokazati kako napraviti igru po perfektnim industrijskim standardima - pošto je za takvu stvar potrebno dugogodišnje iskustvo. No, njime će se prikazati mogući proces izrade igre, dajući pritom neka implementacijska rješenja koja će možda pobuditi maštu čitateljima, ili čak izravno pomoći u shvaćanju nekih problematika s kojim se možemo susresti prilikom izrade.

2 Motivacija

2.1 PC igre

Popularnost igara na osobnim računalima počela je rasti nakon kraha video game konzola 1983. godine i valom povoljnih IBM PC klonova od strane raznih američkih i azijskih kompanija, kao što je primjerice računalo Tandy 1000. Pojavom jeftinijih i pristupačnijih računala povećao se i broj korisnika istih, te su do kraja 1987 osobna računala postala najvažnije tržište za izradu računalnih igara. U 1990-ima, računala su ponovo izgubila veliki dio tržišta pojavom konzola pete generacije kao što su Sega Saturn, Nintendo 64 i PlayStation. Ponovno oživljavanje igara na osobnim računalima dogodilo se 2000-ih nastankom velikim distribucijskih online servisa kao što su Steam i GOG.com [3].

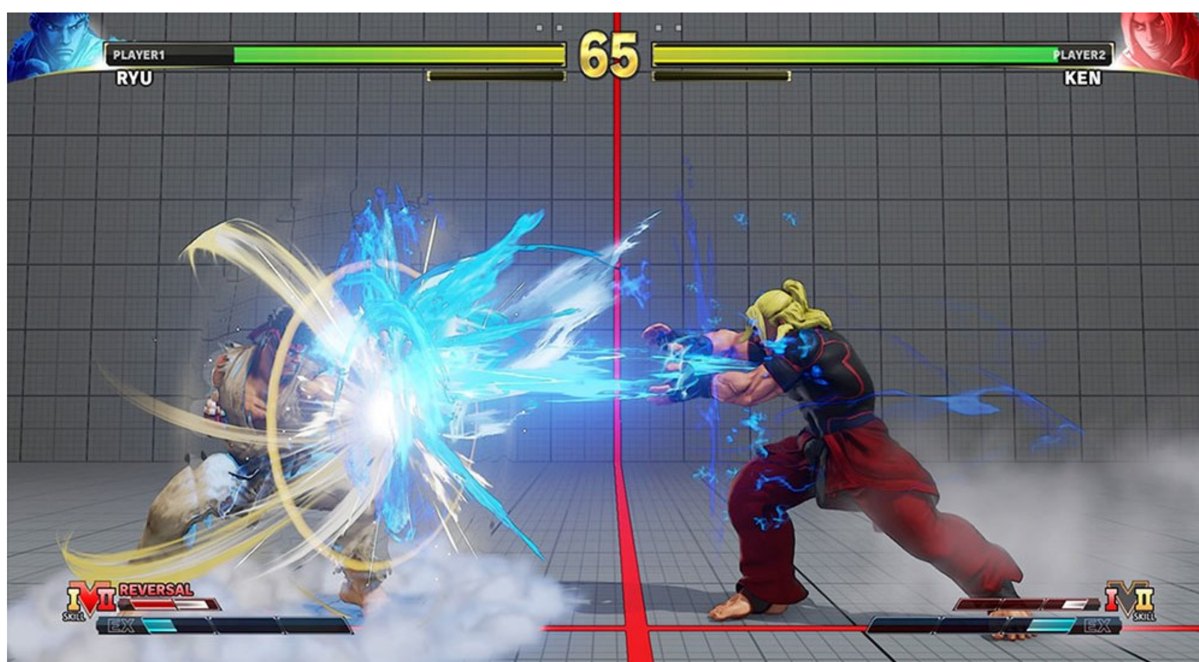
2.2 Žanr

Igre su često kategorizirane prema žanru. Jedan od najstarijih žanrova su borbene igre (eng. fighting games). Ovakve igre karakteriziraju dva borca od kojih je borac na lijevoj strani kontroliran od strane igrača, a desni od strane računala (osim kada se radi o borbi s više igrača u kojem slučaju desnog borca kontrolira drugi igrač). Na vrhu ekrana u svakom kutu nalaze se trake koje predstavljaju živote boraca. Cilj igre je svesti život neprijateljskog borca na nulu. To se postiže uspješnom izvedbom udaraca. Udarci se izvode kombinacijama pritisaka kontrola na kontroleru ili tastaturi. Prema njihovom tipu, udarci se uvijek mogu kategorizirati u više skupina. Primjer kategorizacije tipova udaraca su visoki, srednji, niski, te udarci rukom i udarci nogom. Ovisno o igri, udarci mogu biti drugačije kategorizirani. Strana kojoj je upućen udarac može isti i obraniti. Redovito u ovakvim igrama između traka za život nalazimo vremenski brojač. Vremenski brojač služi kao motivator za akciju. Naime, na početku borbe brojač je namješten na neko razumno vrijeme, primjerice 60 sekundi. Kada brojač dođe na nulu, pobjednikom je proglašen borac s većim brojem životnih bodova. Na taj način, borac s manjim brojem životnih bodova prisiljen je aktivno inicijalizirati konflikt.

Zbog dugovječnosti žanra, te jasnih i uočljivih karakteristika koje su poznate širem rasponu generacija, u ovom radu bit će napravljena igra upravo borbenog žanra.

2.3 Srodne igre

Jedni od najpopularnijih seriala borbenih igara su zasigurno Street Fighter, Mortal Kombat, i Tekken. Slika 1 prikazuje izvedbu udarca u igri Street Fighter V. Na slici su vidljivi svi karakteristični elementi žanra koji su već objašnjeni, ali i nekolicina drugih. Primjerice, na dnu ekrana nalaze se trake koje predstavljaju mjerače za specijalne aktivnosti koje igrač može okinuti tijekom borbe. Prva izdanja Street Fightera nisu sadržavala ukazane mjerače, već su oni uvedeni kroz njene iteracije.



Slika 1 - Street Fighter V [4]

Male izmjene u mehanikama igre implementirane u njenim novijim izdanjima nisu neobična stvar. Primjerice, u igri Mortal Kombat X postoji brojač koji se sastoji od tri trake i naziva se Super Meter. Kada se popuni prva traka, moguće ju je potrošiti da bi se pojačali igračevi specijalni napadi. Trošenje dviju traka daje igraču sposobnost da prekine neprijateljevu kombinaciju (uzastopnu seriju udaraca). Trošenje triju traka pokreće specijalan napad poznat kao "X-Ray". Kada je izašla nova iteracija igre - Mortal Kombat 11, promijenjen je i Super Meter. Umjesto jednog mjerača s tri trake, napravljena su dva mjerača s dvije trake. Jedan mjerač je ofenzivni i koristi se za pojačavanje udaraca i specijalne napade. Drugi mjerač je defenzivni i koristi se za prekide neprijateljeve kombinacije i izbjegavanje udaraca.

Za razliku od Street Fighter V i Mortal Kombat 11, igra Tekken 7 ne sadrži nikakve dodatne mjerače. Umjesto mjerača koji bi aktivacijom pojačavali udarce ili okinuli izvedbu specijalnih udaraca, u Tekkenu su oni dostupni od samog početka. Međutim, izvedba takvih jačih udaraca zahtjeva duže vrijeme izvođenja dajući time neprijatelju više vremena za reakciju. Mehanika Tekken igara ostala je gotovo nepromijenjena od začetaka pa sve do danas. Ona prioritizira sposobnost i vještinu igrača da memorizira duge kombinacije pritisaka kontrola i točnu vremensku egzekuciju pojedinih udaraca.

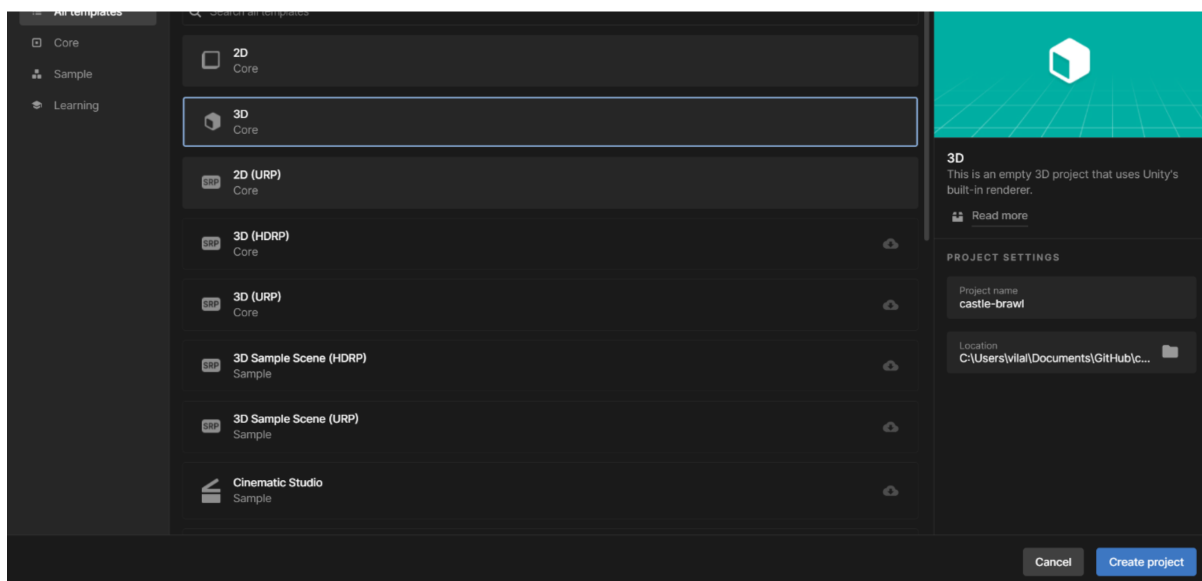
Kroz navedena tri serijala možemo primijetiti male varijante unutar mehanika koje čine svaki od njih posebnima na svoj način. Po tom uzoru igra koja će biti napravljena unutar ovog rada imat će isto malu devijaciju po pitanju mehanike. Naime, postojat će mjerač stamine koji će predstavljati igračevu energiju za izvođenje udaraca. Količina stamine će biti prikazana trakom koja će biti smještena ispod trake za život. Svaki udarac trošit će određenu količinu stamine prilikom izvođenja. Jači udarci zahtijevat će više stamine nego slabiji udarci. Isto tako udarci nogom kojima je dohvat veći trošit će više stamine nego udarci rukom čiji je dohvat manji. U slučaju da borac nema dovoljno stamine za izvođenje udarca, željeni udarac jednostavno neće moći biti izveden i igrač će biti prisiljen čekati da mu se stamina izgenerira.

Osim trake za staminu, igra će biti specifična i po tome što će borci biti uvjetovani s tri različita atributa: snaga (strength), agilnost (agility), izdržljivost (endurance). Snaga će utjecati na količinu štete koju borac može nanijeti udarcima - što je veća snaga bit će veća i šteta. Agilnost će djelovati na brzinu izvođenja udaraca - borac s većom agilnosti brže će izvoditi udarce od borca koji ima manju agilnost. Izdržljivost će pak utjecati na brzinu regeneracije stamine - borac s većom izdržljivosti će u konačnici moći izvesti više udaraca od borca s manjom izdržljivosti.

3 Prvi koraci u izradi projekta

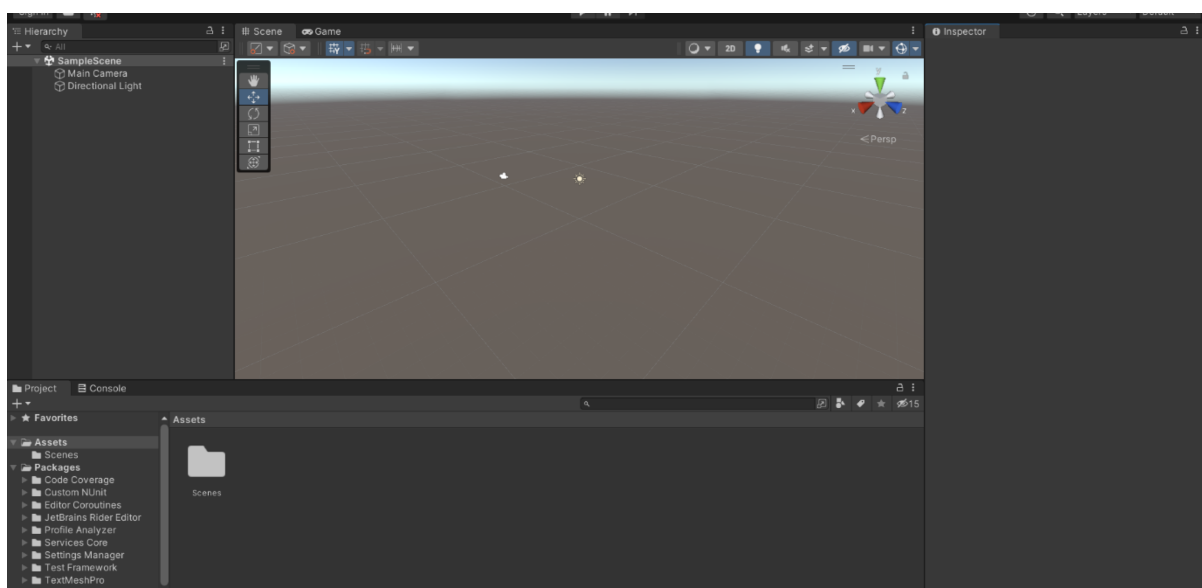
3.1 Novi projekt u Unity-u

Izrada bilo koje Unity igre započinje kreiranjem novog projekta u Unity Hub-u. Za izradu ovog projekta, izabran je 3D predložak. Nakon odabira predloška projekt je nazvan "castle-brawl", što će biti i naziv igre. Potom je odabrana željena lokacija spremanja, te je za kraj kliknuto Create project.



Slika 2 - Kreiranje projekta

Odmah nakon kreiranja novog projekta otvara se Unity editor s praznom scenom.



Slika 3 - Novi projekt

Za bilo kakav rad s Unity editorom potrebno se dobro upoznati s grafičkim sučeljem [5]. Slika 4 prikazuje glavne elemente grafičkog sučelja Unity editora.

Slika 4 – Sučelje [5]

(A) Alatna traka s lijeve strane omogućuje pristup Unity računu i servisima. U sredini se nalaze kontrole za pokretanje simulacije. U desnom kutu tražilica, menu za layere i layout.

(B) Hijerarhijski prozor prikazuje svaki objekt koji se trenutno nalazi na sceni.

(C) Prozor s igrom simulira kako će renderana igra izgledati.

(D) Prozor sa scenom omogućuje da navigiramo kroz scenu i njenim objektima.

(E) Tzv. overlays su alati koji omogućuju brzo i lagano manipuliranje objektima na sceni. Mogu se dodavati i izbacivati po potrebi.

(F) Inspektor prozor daje da vidimo i mijenjamo atribute selektiranog objekta. Budući da različiti objekti imaju različite atribute, ovaj prozor imat će drugačiji raspored svaki put kada je selektiran drugi objekt.

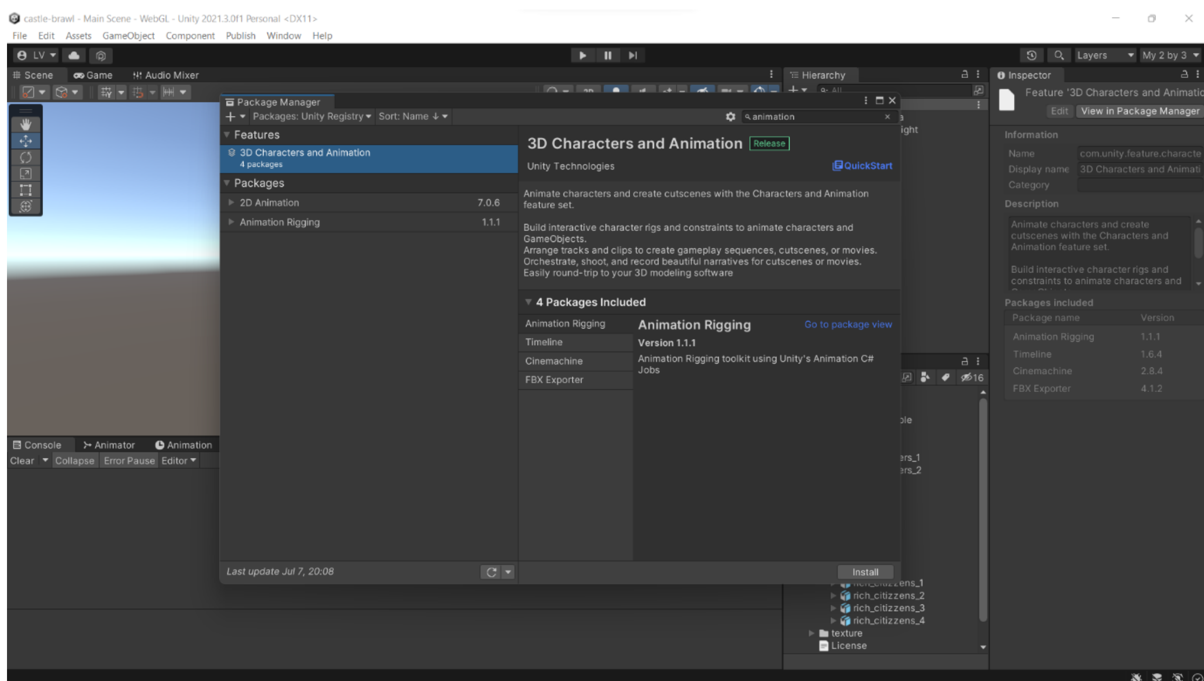
(G) Projektni prozor prikazuje datoteke i foldere u našem projektu.

(H) Statusna traka služi za prikaz notifikacija, te omogućuje brzi pristup alatima kao što je debugging modus.

Nadalje, za izradu igre su često potrebni razni asseti i funkcionalni dodatci. Mnoge ne esencijalne funkcionalnosti nisu sadržane u Unity projektu od samog starta kako bi se uštedjelo na kompleksnosti, te ih se naknadno može dodati putem package Managera. Uz njih, tu možemo naći i brojne druge pakete napravljene od strane vrijednih korisnika koji su svoje dodatke stavili na dostupnost drugima uz pristajuću licencu.

Alternativni način dodavanja asseta ili dijelova koda bio bi kopiranje istih u folder projekta. Dodavanje novih datoteka u folder projekta kroz explorer (izvan editora) nije problem, međutim preporučuje se da micanje i mijenjanje naziva bude učinjeno kroz Unity editor. Razlog tomu je to što Unity za svaku datoteku i folder unutar projekta generira .meta datoteku koja mu služi za referenciranje iste. Ako je datoteci promijenjena putanja izvan editora (a pripadajućoj .meta datoteci nije), onda je objekti koji se njome služe neće moći pronaći [6].

Slika 5 prikazuje preuzimanje grupe paketa za upravljanje likovima i animacijama.

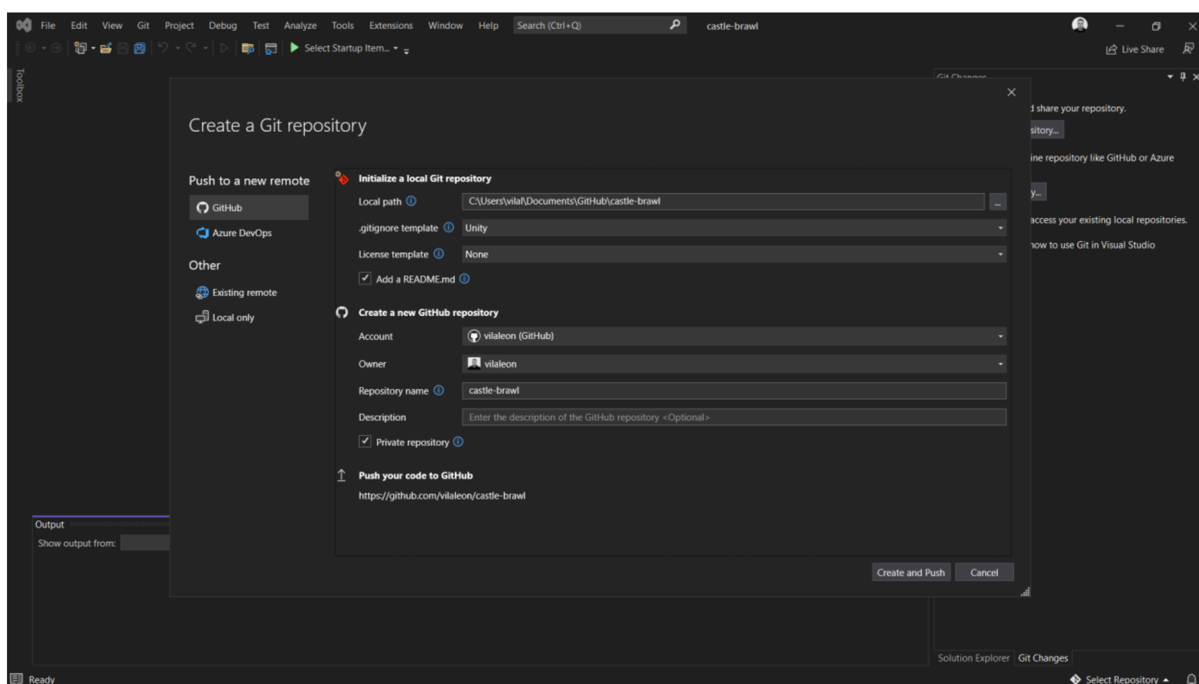


Slika 5 - Package Manager

3.2 Upravljanje verzijama

Sustavi za upravljanje verzijama u suštini nisu potrebni, ali su jako korisni. Oni omogućuju lagano praćenje i revizioniranje promjena u projektu. Ako je učinjena neka pogreška, pomoću takvog sustava možemo brzo i lagano vratiti projekt u prethodno spremljeno stanje bez mnogo razmišljanja i briga da li smo kako treba vratili izmjene na svim dotičnim mjestima.

Za ovaj projekt koristit će se Git kao odabrani sustav za upravljanje verzijama. Nakon instalacije Git programa potrebno je inicijalizirati sustav za praćenje za projektni folder, takav folder se službeno naziva git repozitorij. To se može učiniti na razne načine preko konzole ili nekog sučelja. Slika 6 prikazuje prozor za inicijalizaciju projekta kao git repozitorija koristeći sučelje od Visual Studia.

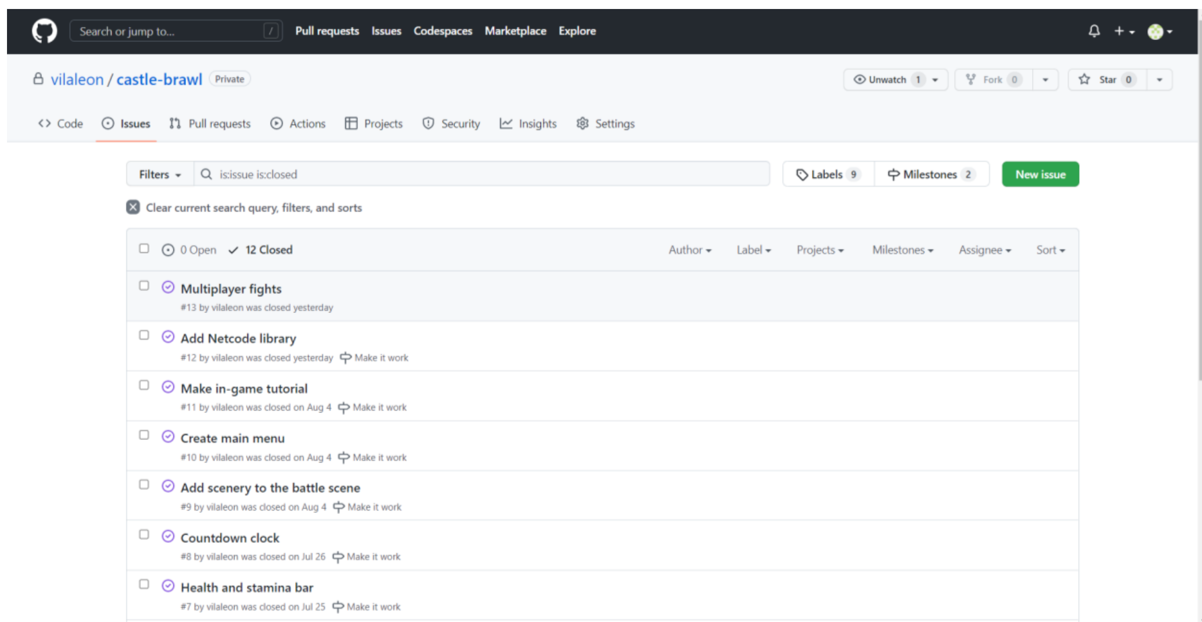


Slika 6 – Repozitorij

Nakon formi za inicijalizacije lokalnog Git repozitorija nalaze se i forme za kreiranje GitHub repozitorija. GitHub je platforma koja daje servis hostanja Git repozitorija putem interneta. To znači da projekt ne samo da će biti spremljen lokalno na osobnom računalu, nego će biti spremljen i na remote serveru. Tako osiguravamo da će verzije projekta biti dostupne čak i ako se nešto desi osobnom računalu. Isto tako, ako netko bude htio kopirati ili sudjelovati na projektu, bit će potrebno samo par klikova da mu se to omogući putem GitHub-a.

Osim što nam pruža remote hosting repozitorija, GitHub isto tako ima vrlo zanimljive alate koji olakšavaju upravljanje projektom kao što je primjerice pregled problema (Issues). Iako ime upućuje na korištenje isključivo za probleme, GitHub dokumentacija predlaže korištenje te funkcionalnosti za praćenje ideja, povratnih informacija, zadataka, ili bugova [7]. U te svrhe se i naširoko svakodnevno koristi.

Slika 7 prikazuje pregled nekih od zadataka koji su napravljeni prilikom izrade igre.



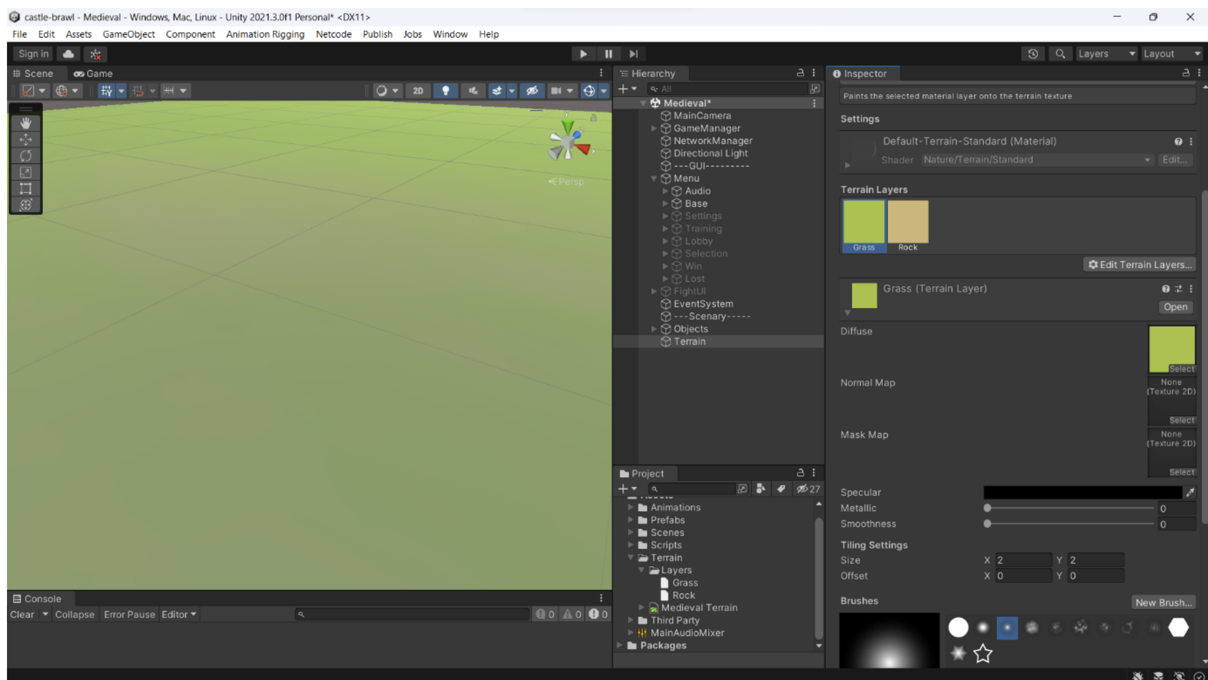
Slika 7 - GitHub Issues

U nastavku rada se više neće spominjati sustav za upravljanje verzijama. On se koristio intenzivno za spremanje napravljenih dijelova, ali nije toliko važan da bi o njemu dalje pisati.

4 Izrada okoline

4.1 Teren

Početak uređivanja okoline unutar igre često započinje s izradom terena. Objekt tipa teren može se dodati klikom na GameObject > 3D Object > Terrain. Klikom na novo nastali teren objekt, otvaraju se u inspektoru svojstva tog objekta gdje se ista mogu i mijenjati. Za ovaj projekt nije bilo potrebe za velikim sređivanjem terena, već je samo trebalo promijeniti njegovu veličinu i ocrtati texture. Za crtanje tekstura terena odabran je Paint terrain izbornik, te potom crtanje tekstura. U Terrain Layers sekciji dodane su dvije teksture koje će biti korištene za podlogu - "grass" i "rock", te su iste potom i nanesene koristeći kist alat. Slika 8 prikazuje odabrane texture i podlogu nakon što su na nju nanijete teksture.

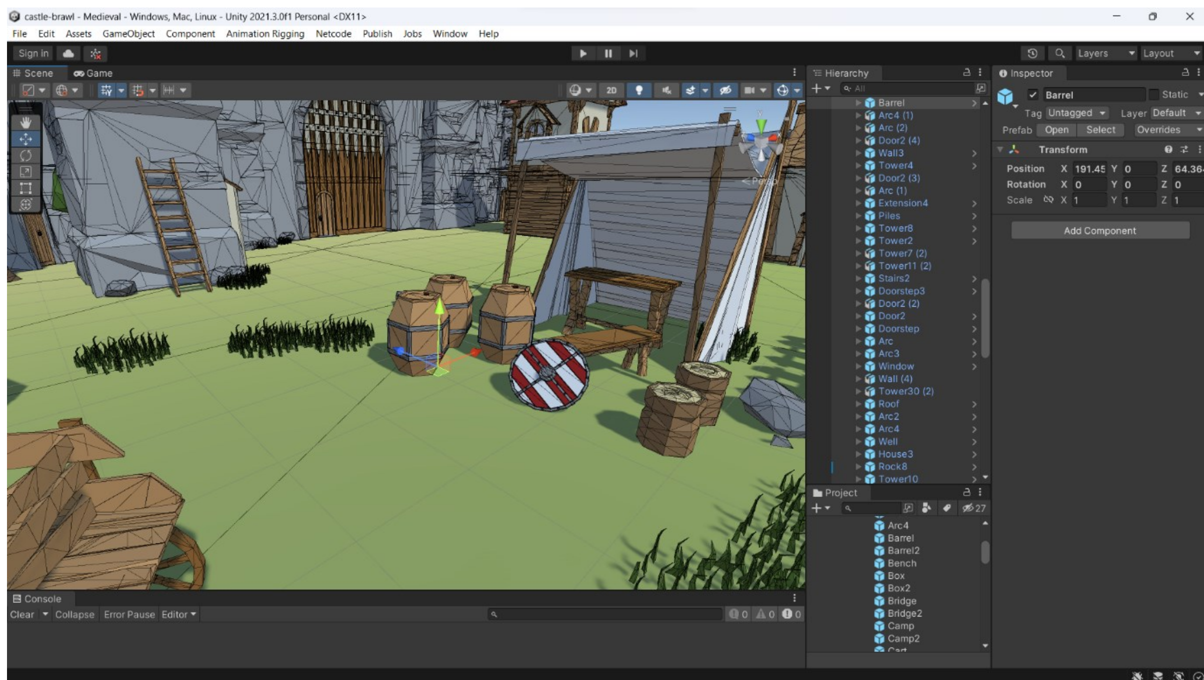


Slika 8 - Teren

4.2 3D modeli

Nakon što je napravljen teren, vrijeme je za ispunjavanje prostora s raznim 3D modelima. Za ovaj projekt preuzeti su asseti s već gotovim 3D objektima srednjovjekovne tematike [8]. Preuzeti asseti sadržavaju modele s manjim brojem poligona (tzv. low-poly). Low-poly modeli se često koriste kako bi se smanjilo opterećenje grafičke kartice. Neki od načina ubacivanja objekata u scenu su povlačenjem s projektnog prozora na hijerarhijski prozor, ili izravnim povlačenjem na scenu. Kada su modeli ubačeni u scenu, može ih se micati uz pomoć alata na overlay alatnoj traci, ili mijenjanjem Transform parametara u inspektoru.

Slika 9 prikazuje modele postavljene na scenu. Desno od scene vidljiv je njihov hijerarhijski prozor i inspektor prozor koji pokazuje na fokusirani element (u ovom slučaju bačvu).



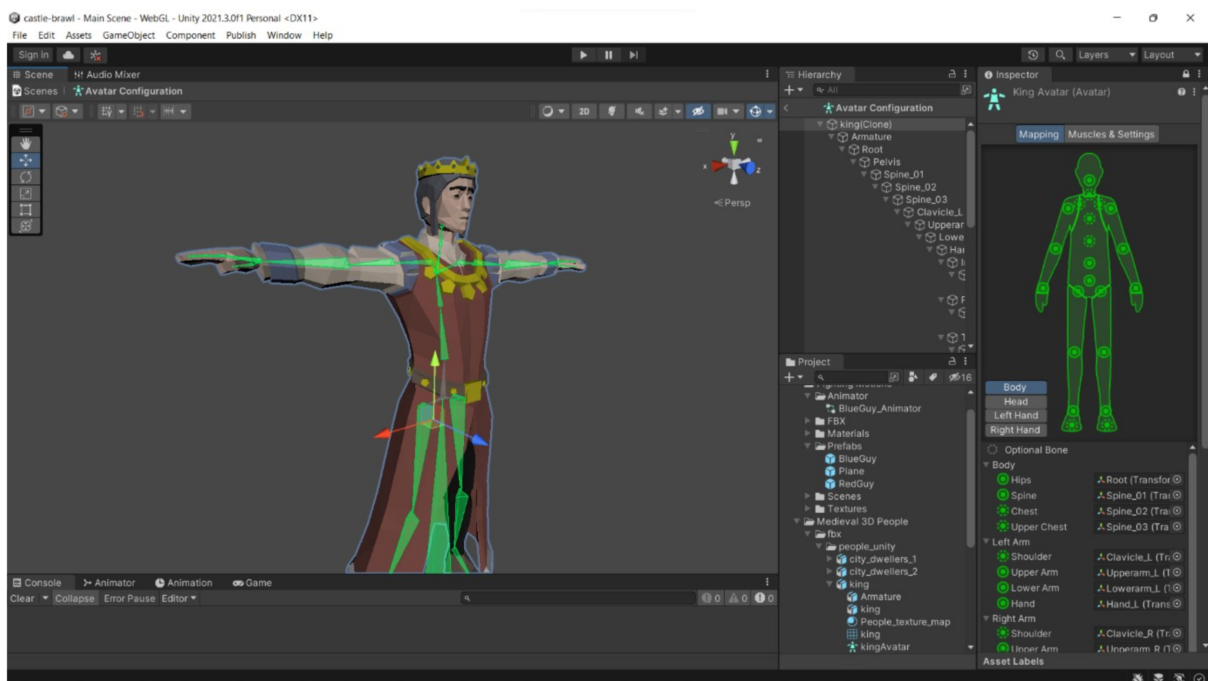
Slika 9 - Modeli u sceni

U hijerarhijskom i projektnom prozoru sa slike isto tako vidimo da su objekti ubačenih modela obojeni plavom bojom. Plava boja označava da su oni prefabricirani (prefabs). To znači da se sve komponente i kompozitni objekti unutar njih gledaju kao cjelina. Preciznije, ubaci li se u scenu više kopija istog prefab objekta, dovoljno je promijeniti izvorni prefab i promjene će se reflektirati na svim kopijama unutar scene [9].

4.3 Likovi

Modeli likova bi trebali odgovarati izgledom i strukturom ostalim modelima u sceni da bi se održao vizualan sklad igre. Konkretno u slučaju ovog projekta, to bi značilo da trebaju biti srednjovjekovne tematike i imati low-poly mesh. Isto tako, kako bi se mogli lakše animirati, njihovi modeli trebaju imati kostur (rig). Preuzeti asseti likova za naš projekt [10] došli su u formatu .fbx. Odmah nakon preuzimanja za svakog lika trebalo je namjestiti da je tip kostura humanoid. To se čini na način da se klikne na njegov asset > Rig > Animation Type > Humanoid.

Nadalje, u inspektoru prozoru za .fbx datoteku, klikom na avatar > Configure Avatar će se otvoriti Avatar Configuration prozor (Slika 10). Ovdje se može provjeriti da li su sve "kosti" mapirane kako treba na odgovarajuće dijelove tijela (zelena silueta na desnoj strani slike). Rig će biti od ključne važnosti za postavljanje animacija.

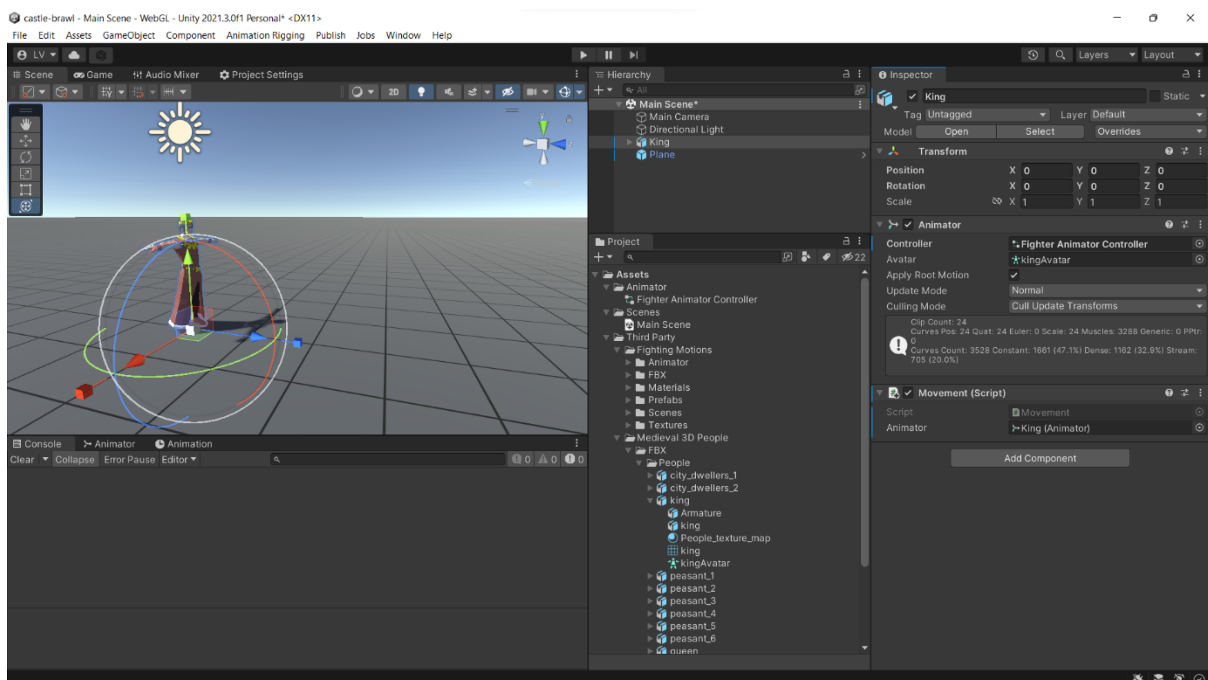


Slika 10 - Avatar konfiguracija

4.4 Animacije na likovima

4.5 Animator

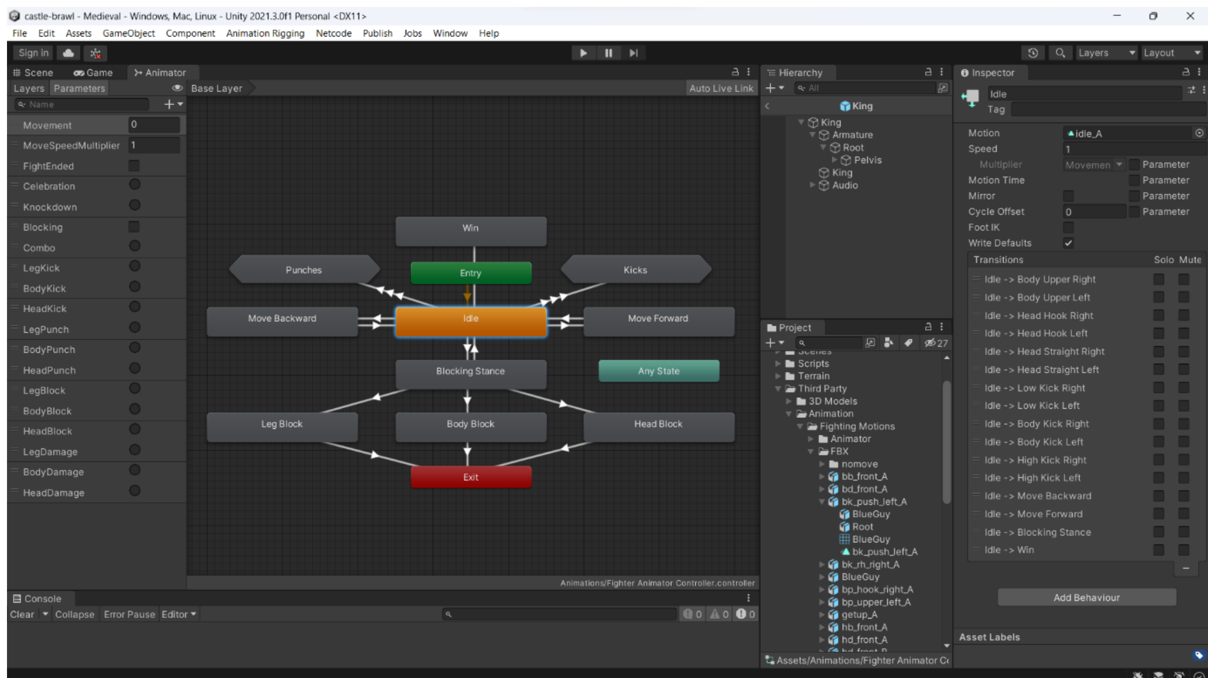
Pošto se radi o borbenoj igri, preuzet je asset s borbenim animacijama koje su namještene za humanoidni kostur [11]. Potom je kreiran novi kontroler animacija klikom na Assets > Create > Animation Controller, s nazivom "Fighting Animation Controller". Svim likovima dodana je nova komponenta imenom Animator (Slika 11). U Animator polju Controller je zatim za svakog lika označena referenca na novonapravljen Fighting Animation Controller. Slično je napravljeno i za polje Avatar, gdje je za svakog lika označen dotični avatar iz .fbx datoteke - to je kostur koje će animacije koristiti. Za kraj, dobiveni objekti likova s namještenim animatorom spremljeni su kao prefab.



Slika 11 – Animator

Nakon gornjih koraka likovi imaju sve komponente koje trebaju imati, međutim još uvijek nisu animirani. Sljedeći korak je otvaranje Fighting Animation Controllera. Pri otvaranju otvorit će se novi prozor s nazivom "Animator". U Animator prozoru vidljiva su sva stanja u kojem animacije mogu biti. Po zadanim postavkama novi kontroler uvijek ima tri elementa: Entry (ulaz), Exit (izlaz), Any State (bilo koje stanje). Desnim klikom na površinu otvora se kontekstualni menu s opcijom dodavanja novih praznih stanja. Desnim klikom na stanje otvara se menu s opcijom dodavanja tranzicije prema drugom stanju. Stanja i tranzicije čine osnovnu cjelinu Animator sustava [12].

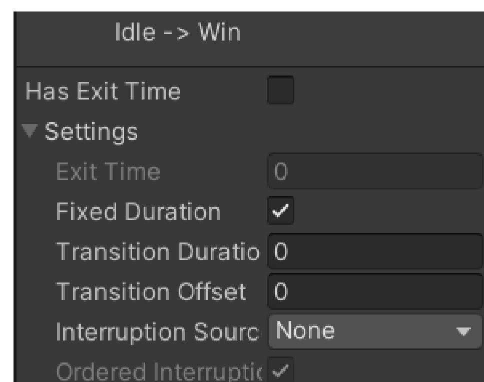
Slika 12 prikazuje krajnji prikaz stanja i tranzicija kontrolera postavljenog na likovima. Na njoj je za primjer fokus stavljen na Idle stanje. Postavke stanja vidljive su u inspektor prozoru s desne strane. Polje Motion služi za referenciranje animacija za stanje. U primjeru sa slike stavljena je referenca na animaciju naziva "idle_A", te će ista biti aktivna kada je aktivno i Idle stanje. Ostala stanja na isti način referenciraju pojedine animacije.



Slika 12 - Fighting Animation Controller

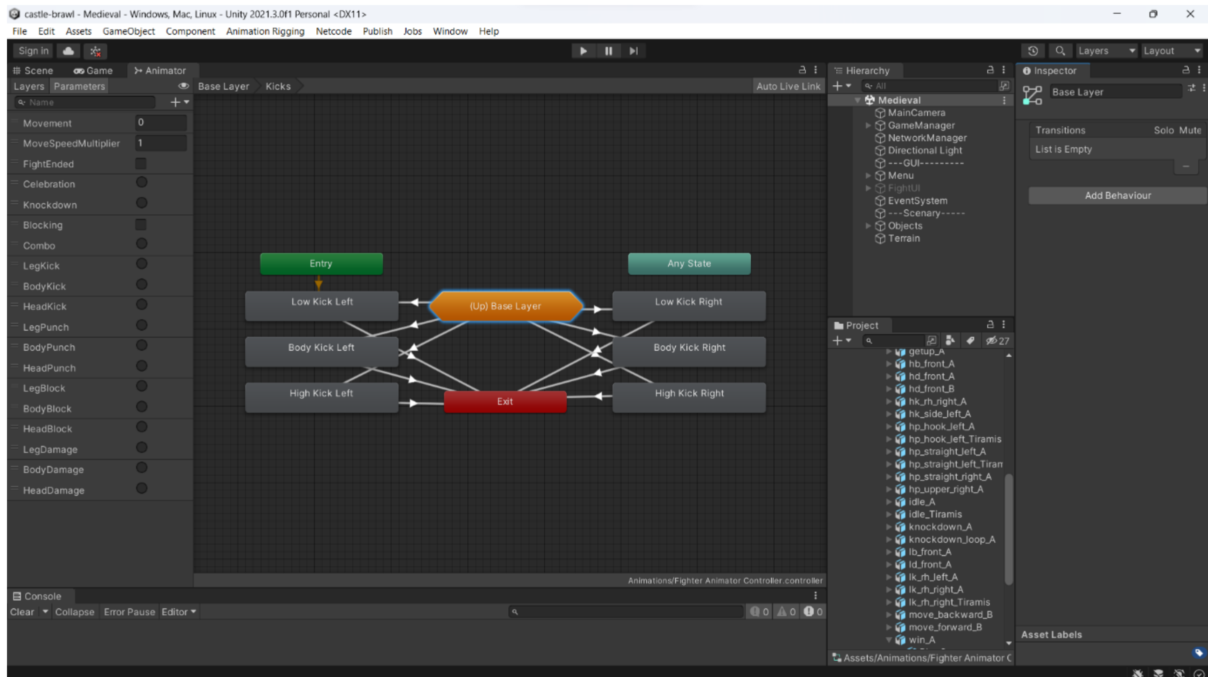
Na dnu inspektor prozora u tom primjeru nalazi se lista tranzicija koje vode iz Idle stanja u druga stanja. Svaka tranzicija ima uvjet koji treba biti zadovoljen kako bi ona započela. Na lijevoj strani primjera vidljivi su svi parametri koji su definirani kao uvjeti pojedinih tranzicija.

Slika 13 prikazuje jednu napravljenu tranziciju. Kako bi animacije izgledale reaktivnije, polje s nazivom Transition Duration smanjeno je na 0. Nadalje, isključeno je Has Exit Time. Kada bi ono bilo uključeno, to bi značilo da tranzicija može započeti jedino kada animacija dosegne određenu točku u prethodnoj tranziciji. Na isti način postavljene su i ostale tranzicije za ovaj projekt.



Slika 13 - Tranzicija

Animator kontroler s prethodne stranice (Slika 12) ima sva stanja prikazana pravokutnim oblikom osim "Punches" i "Kicks". To je zato što to u stvari nisu stanja, već poddijagrami stanja. Slika 14 prikazuje prikaz stanja otvaranjem poddijagrama "Kicks".



Slika 14 - Sub-State Machine

Poddijagrami su korisni kada treba razbiti kompleksnost većeg dijagrama na manje preglednije cjeline (tzv. top-down dizajn).

4.6 Primjer ciklusa kontrolera

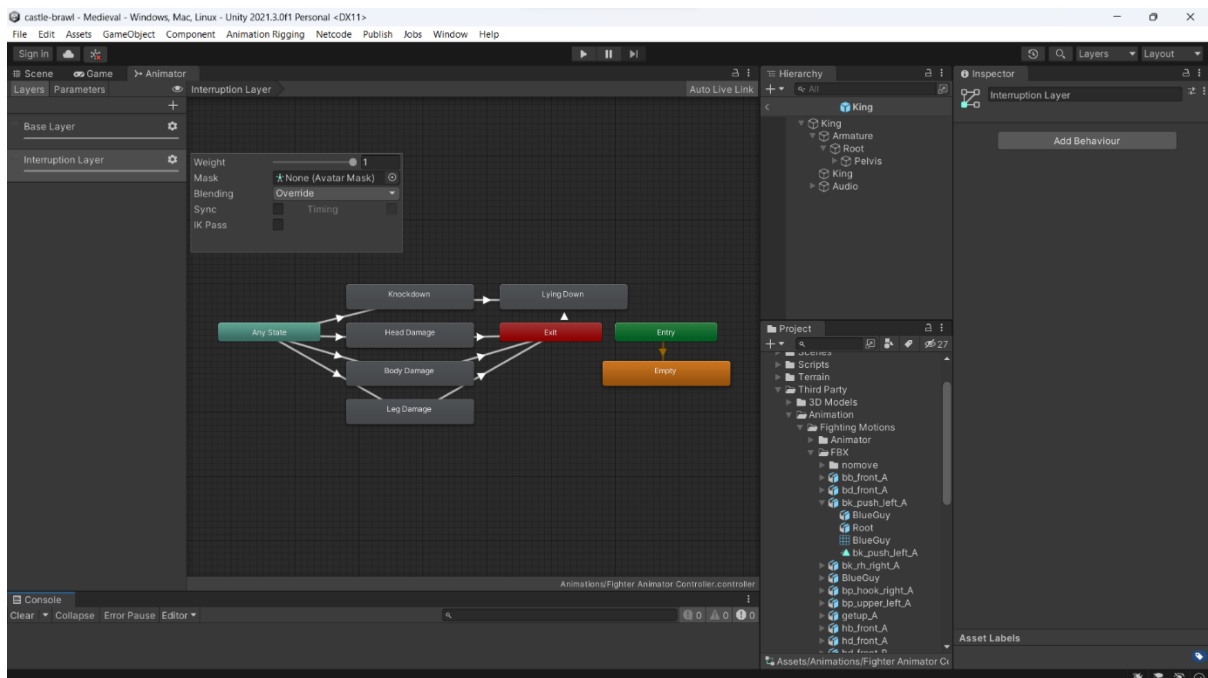
Kako bismo dobili bolju predodžbu kako kontroler radi, razmotrimo primjer prijelaza stanja kontrolera na liku:

- Lik i komponente su učitane u scenu. Kontroler iz Entry stanja tranzicijom prelazi u Idle stanje i na liku se izvršava animacija za stajanje na mjestu.
- Tijekom igre parametar "Head Kick" se uključi. Tranzicija koja vodi do stanja High Kick Left unutar poddijagrama "Kicks" ima kao uvjet taj parametar te se uključuje. High Kick Left stanje je sada uključeno i odvija se animacija za visoki udarac nogom.
- Nakon što je animacija za udarac nogom završila, tranzicija koja vodi prema Exit nema uvjet te se automatski uključuje. Izlazi se is poddijagrama i stanje se vraća na Entry kao u prvoj točci.

4.7 Slojevi animatora

Unutar animator prozora na vrhu popisa parametra nalazi se i opcija za uređivanje slojeva (Layers). Slojevi služe za izradu kompleksnih animacija [13]. Primjer kompleksne animacije bi bilo imati jedan sloj za gornji dio tijela, a jedan za donji. Na taj način imali bismo dva različita toka stanja na svakom sloju omogućujući da mijenjamo stanja animacija na gornjem i donjem dijelu tijela u isto vrijeme.

Druga korisna primjena slojeva je ta što s jednim slojem možemo prijeći preko drugog sloja iste težine. To svojstvo je i iskorišteno za izradu ovog projekta. Slika 15 prikazuje otvoren prozor stanja za sloj koji je naknadno dodan pod nazivom "Interruption Layer". Ključno je da je Blending na sloju postavljen na Override, što je isto vidljivo na slici. Ovaj sloj služi za animacije trzaja prilikom primitka udarca. Primijetimo isto da tranzicije ne započinju iz Entry, nego iz Any State. Any State je najlakše zamisliti kao stanje koje je spojeno na svako drugo stanje u animatoru. Na taj način se tranzicija može započeti iz bilo kojeg stanja uz uvjet da je zadovoljen uvjetni parametar za njeno započinjanje. Na ovaj način dobiven je efekt da kada je okinut parametar za primanje štete, na Interruption Layeru će započeti tranzicija u odgovarajuće stanje s prikladnom animacijom. Ta animacija će napraviti override nad bilo kojom animacijom koja se odvijala na osnovnom sloju.



Slika 15 - Slojevi animatora

5 Sustav borbe

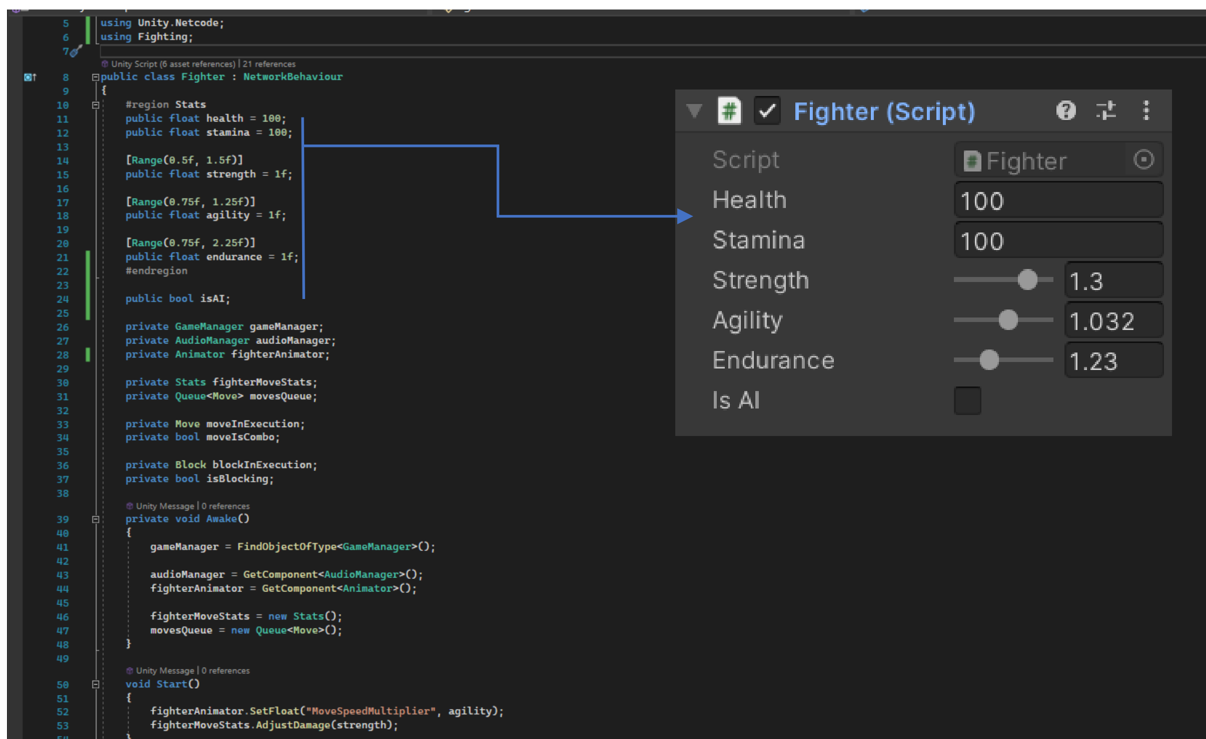
5.1 Borci

Sustav borbe započinje deklariranjem imenskog prostora koji je nazvan "Fighting". U njemu su enumerirane vrste mogućih udaraca i blokova. U njemu je isto tako implementirana klasa "Stats" u kojoj će se nalaziti statovi za udarce (Slika 16).

Zatim je napravljena skripta "Fighter" (Slika 17) koja je dodana kao komponenta svim likovima-borcima. Ona će služiti za upravljanje logikom boraca. Korisno je primijetiti da su public atributi unutar skripte dostupni u inspektor prozoru unutar Unity editora (plava strelica).

```
namespace Fighting
{
    public enum Move
    {
        None,
        LegKick,
        BodyKick,
        HeadKick,
        LegPunch,
        BodyPunch,
        HeadPunch
    }

    public enum Block
    {
        None,
        LegBlock,
        BodyBlock,
        HeadBlock
    }

    public class Stats
    {
        public class Move
        {
            public Fighting.Move name;
            public Block block;
            public string trigger;
            public float damage;
            public float stamina;
            public float height;
        }

        public Move[] moves =...;

        public Move Get(Fighting.Move move)...

        public void AdjustDamage(float strength)
        {
            foreach (var move in moves)
            {
                move.damage *= strength;
            }
        }
    }
}
```

Slika 16 - Namespace za pokrete



Slika 17 - Klasa Fighter

Slika 17 s prethodne strane isto prikazuje među ostalom i funkcije Awake i Start. Awake se poziva kada je skriptni objekt inicijaliziran, a ako je riječ o novoj sceni, onda kada su svi aktivni objekti inicijalizirani. Start se poziva nešto kasnije - prije prve Update funkcije. Funkcija Update (koja nije vidljiva na toj slici) poziva se svaki frame. Unity dokumentacija predlaže korištenje Awake za inicijalizaciju varijabli [14]. U implementaciji Awake funkcije iz Fighter klase vidljiva je inicijalizacija i instanciranje klase "Stats" koja je definirana u namespaceu "Fighting". Osim toga vidljivo je i dohvaćanje komponente Animator o kojoj je bilo riječ u prošlom poglavlju. Tu se još nalaze inicijalizacije par drugih objekata o kojima će biti više riječi u nastavku rada.

5.2 Izvedba udaraca

Izvedba udaraca sastoji se od 2 metode: AddMoveToQueue i ExecuteNextMoveInQueue. Slika 18 prikazuje implementaciju navedenih. AddMoveToQueue dodaje udarac u red čekanja sve dok red ne sadrži maksimalan broj rezerviranih udaraca.

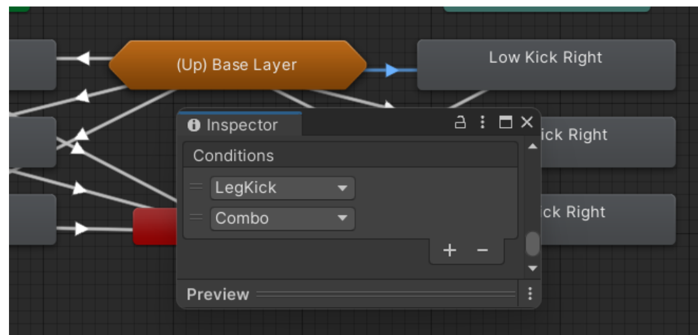
```
public void AddMoveToQueue(Move move)
{
    if (movesQueue.Count < MAX_MOVES)
    {
        fighterAnimator.SetBool("Blocking", false);
        movesQueue.Enqueue(move);
    }
}
void Update()
{
    if (fighterAnimator.GetCurrentAnimatorStateInfo(0).IsName("Idle")) ExecuteNextMoveInQueue();
}
private void ExecuteNextMoveInQueue()
{
    Move move;
    if (movesQueue.TryDequeue(out move) && stamina >= fighterMoveStats.Get(move).stamina)
    {
        if (move == moveInExecution && !moveIsCombo)
        {
            moveIsCombo = true;
            fighterAnimator.SetTrigger("Combo");

            if (move.ToString().Contains("Kick"))
            {
                audioManager.Play("Kick Grunt");
            }
            else
            {
                audioManager.Play("Punch Grunt");
            }
        }
        else
        {
            moveIsCombo = false;
            moveInExecution = move;
        }

        fighterAnimator.SetTrigger(move.ToString());
        if (NetworkManager.IsListening)...
        else
        {
            stamina -= fighterMoveStats.Get(move).stamina;
        }
    }
    else
    {
        moveIsCombo = false;
        moveInExecution = Move.None;
    }
```

Slika 18 - Provedba udarca

Svaki frame poziva se funkcija Update koja, ako je lik u Idle animaciji (tj. ne radi ništa), poziva metodu ExecuteNextMoveInQueue (egzekuciju sljedećeg udarca u redu).

ExecuteNextMoveInQueue uzima sljedeći udarac u redu čekanja i ukoliko borac ima dovoljno stamine provodi egzekuciju na način da postavi odgovarajući triger parametar na animatoru lika. Taj triger parametar će pokrenuti odgovarajuću tranziciju na animatoru i samim time stanje s animacijom traženog udarca. No prije postavljanja tog glavnog parametra za animaciju udarca postoji još jedna mala selekcija koja prati da li je isti udarac zaredan (combo).

Combo je slučaj kada je udarac jednak prethodnom udarcu. Kada se to dogodi, audio manager na borcu će izvršiti poseban zvuk (o audio kontroli će biti govora kasnije), te će se postaviti još jedan dodatni triger parametar na animatoru koji će rezultirati paljenju alternativne tranzicije s malo drugačijom animacijom udarca (Slika 19).



Slika 19 - Alternativni udarac

Na samom kraju stamina lika će biti umanjena za vrijednost stamine tog udarca.

5.3 Blok i micanje

Prije nego budu objašnjeni načini registriranja udarca i štete, treba analizirati još dvije ključne metode - Movement i Block. Movement (micanje) uzima kao argument decimalnu vrijednosti i namješta odgovarajući parametar na animatoru koji će rezultirati animacijom hoda za naprijed ili nazad. Da bi lik zadržao konačnu poziciju nakon izvedbe animacije, potrebno je označiti Apply Root Motion na Animatoru.

Block (blokiranje) uzima kao argument enumeriranu block vrijednost. Ukoliko ne postoji udarac u provedbi znači da je moguće izvesti blok, i stoga se namješta varijabla za stanje bloka.

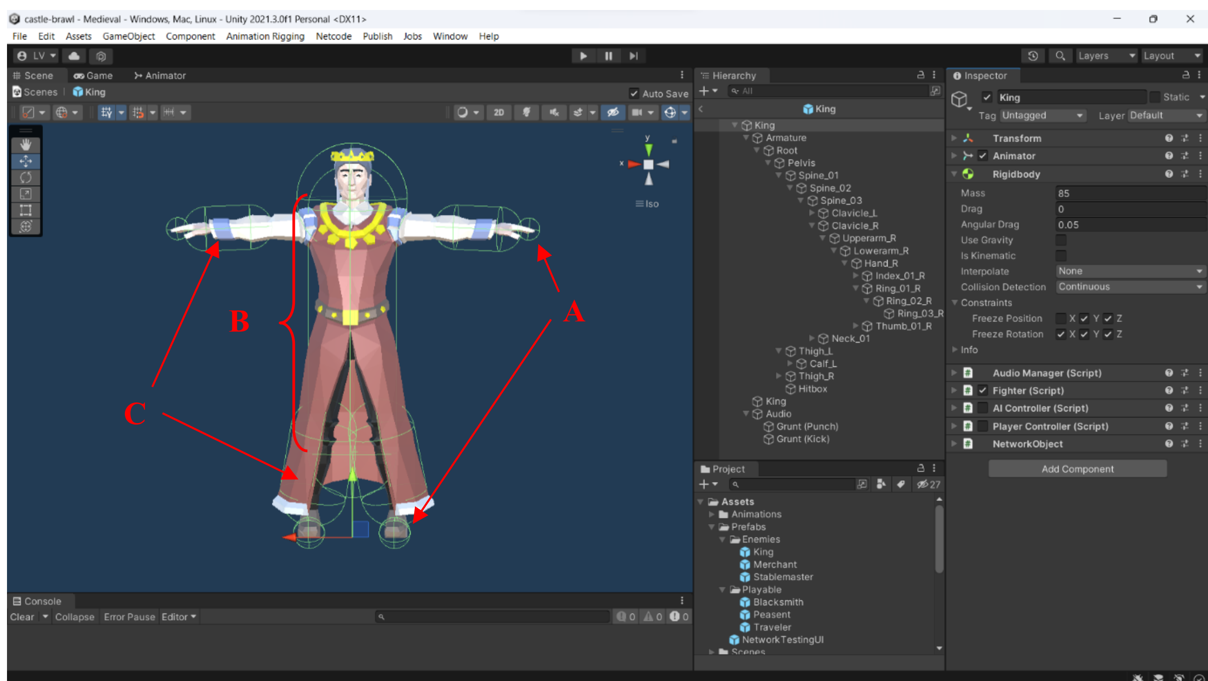
```
140     6 references
        public void Movement(float horizontalInput)
141     {
142         fighterAnimator.SetFloat("Movement", horizontalInput);
143     }
144
        3 references
145     public void TryToBlock(Block block)
146     {
147         if (moveInExecution == Move.None)
148         {
149             if (fighterAnimator.GetCurrentAnimatorStateInfo(0).IsName("Idle"))
150             {
151                 fighterAnimator.SetBool("Blocking", true);
152             }
153
154             isBlocking = true;
155             blockInExecution = block;
156         }
157     }
158
```

Slika 20 - Blok i micanje

5.4 Registriranje udaraca

Objašnjeno je kako "Fighter" klasa provodi izvedbu udarca: dodavanjem željenog udarca u red čekanja i egzekucijom udaraca iz reda kada je lik u Idle stanju. Rezultat tih radnji bit će animacije tj. pomicanje lika unutar prostora. Kako bi to pomicanje moglo interaktirati s okolinom potrebno je dodati kolajdere (colliders) i okidače (triggers) [15].

Kolajderi služe da bi mogli detektirati njihove kolizije u prostoru. No, da bi se ta detekcija desila, barem jedan od objekata u koliziji mora imati komponentu Rigidbody. Slika 21 prikazuje stavljanje kolajdera jednom od likova tokom izrade projekta. Na desnoj strani slike, u inspektoru, vidljiva je Rigidbody komponenta. Pod Constrains su zamrznute sve osi i rotacije osim X osi, to je iz razloga što će se likovi micati samo po osi X. Promijenjena je i Collision Detection sa zadanog Discrete na Continuous jer je potrebna preciznija detekcija. Ugašen je Use Gravity jer nisu potrebne dodatne fizikalne kalkulacije pošto se pomicanje likova u ovom projektu radi putem animacija, a ne fizike. Ostalo je pušteno na zadanim postavkama.



Slika 21 - Colliders

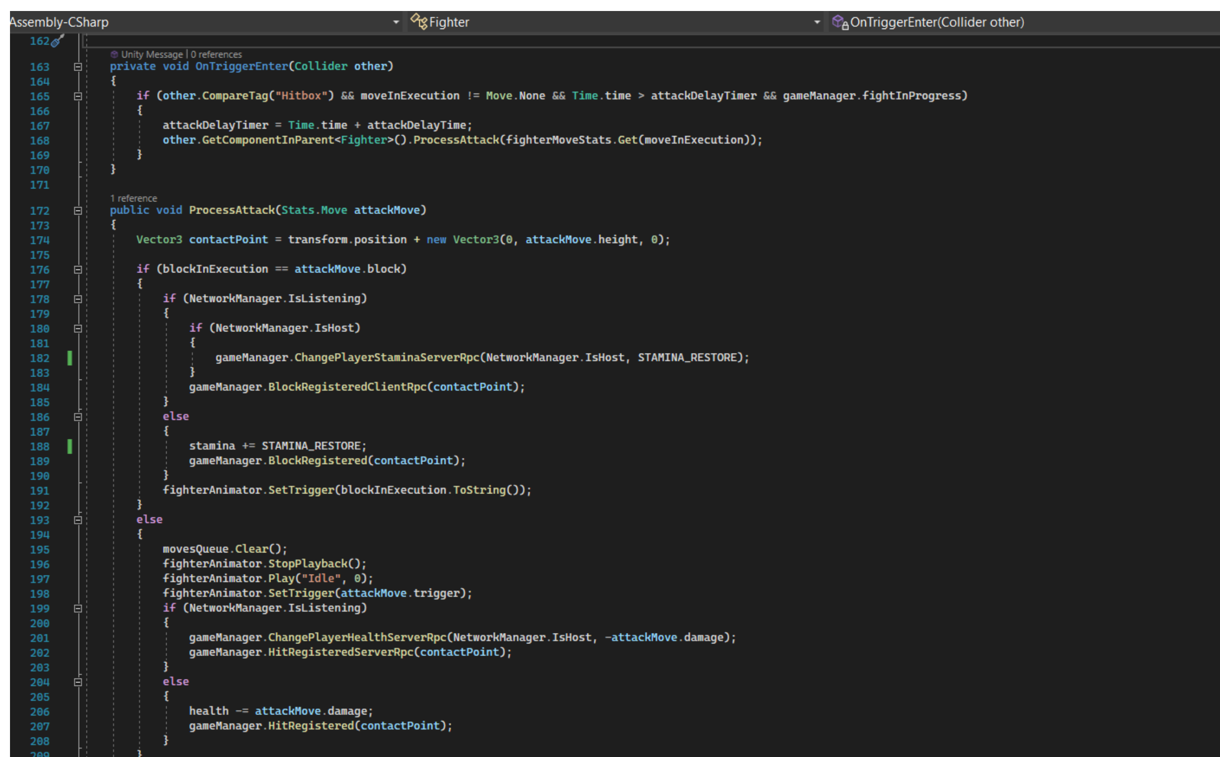
Nakon dodavanja Rigidbody komponente, dodani su i kolajderi na pojedine dijelove tijela tako da u konačnici imamo mrežu kolajdera. Prema zadaćama možemo ih razvrstati u tri grupe.

(A) Sferni kolajderi na vrhovima šaka i nožnih prstiju imaju označenu kvačicu Is Trigger. Kada ovi kolajderi dođu u kontakt s drugim kolajderima znači da je lik udario u nešto.

(B) Sferni kolajder za glavu i cilindrični za tijelo nalaze se u objektu tagiranim s tagom "Hitbox". Tag je dodan tako da je kliknuto Tag > Add Tag > New Tag. Kada sferni kolajderi udare u kolajder objekta tagiranim sa "Hitbox" znači da je udarac bio uspješan.

(C) Cilindrični kolajderi na rukama i nogama služe da bi davali privid volumena likova. Oni će biti ti koji prilikom udarca odguruju lika u kontaktu.

Slika 22 prikazuje dio koda za registraciju udaraca. OnTriggerEnter funkcija se poziva svaki put kada trigger kolajder uđe unutar drugog kolajdera. Selekcijom se potvrđuje da kolajder u pitanju ima "Hitbox" tag, da se kolizija desila prilikom izvođenja udarca, da je prošlo dovoljno vremena između zadnjeg udarca (eliminacija slučajnih višestrukih poziva), te da je borba aktivna. Ukoliko su ti uvjeti zadovoljeni poziva se metoda ProcessAttack na pogođenom liku i kao argument se predaje udarac u izvođenju.



```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Hitbox") && moveInExecution != Move.None && Time.time > attackDelayTimer && gameManager.fightInProgress)
    {
        attackDelayTimer = Time.time + attackDelayTime;
        other.GetComponentInParent<Fighter>().ProcessAttack(fighterMoveStats.Get(moveInExecution));
    }
}

public void ProcessAttack(Stats.Move attackMove)
{
    Vector3 contactPoint = transform.position + new Vector3(0, attackMove.height, 0);

    if (blockInExecution == attackMove.block)
    {
        if (NetworkManager.IsListening)
        {
            if (NetworkManager.IsHost)
            {
                gameManager.ChangePlayerStaminaServerRpc(NetworkManager.IsHost, STAMINA_RESTORE);
            }
            gameManager.BlockRegisteredClientRpc(contactPoint);
        }
        else
        {
            stamina += STAMINA_RESTORE;
            gameManager.BlockRegistered(contactPoint);
        }
        fighterAnimator.SetTrigger(blockInExecution.ToString());
    }
    else
    {
        movesQueue.Clear();
        fighterAnimator.StopPlayback();
        fighterAnimator.Play("Idle", 0);
        fighterAnimator.SetTrigger(attackMove.trigger);
        if (NetworkManager.IsListening)
        {
            gameManager.ChangePlayerHealthServerRpc(NetworkManager.IsHost, -attackMove.damage);
            gameManager.HitRegisteredServerRpc(contactPoint);
        }
        else
        {
            health -= attackMove.damage;
            gameManager.HitRegistered(contactPoint);
        }
    }
```

Slika 22 - Registriranje udarca

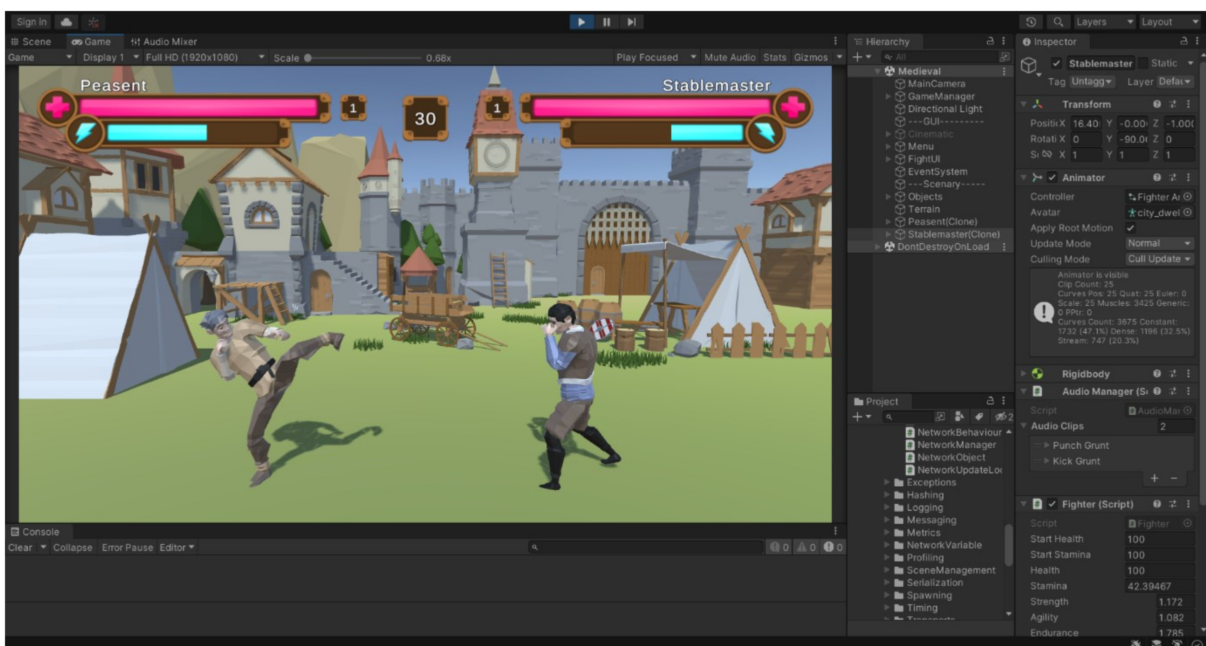
Metoda ProcessAttack za argument uzima primljeni udarac. Prvo određuje približno mjesto kontakta tako da uzme poziciju lika i dodamo visinu udarca - alternativa bi bila korištenje

Collision.GetContact metode u OnTriggerEnter funkciji. Nakon toga uspoređuje da li blok u izvođenju odgovara adekvatnom bloku primljenog udarca. Ukoliko odgovara, znači da je blok uspješan i povećava staminu, javlja Game Manageru da se desio kontakt na upravo tom mjestu, te trigera adekvatnu animaciju za blok. Da blok nije bio uspješan, očistio bi se red čekanja na udarce, namjestio Animator tako da trigera animaciju za primanje udarca, oduzeo život za vrijednost štete udarca, te za kraj javio Game Manageru da se desio udarac na upravo tom mjestu. U oba gornja slučaja zadaća Game Managera bila bi da pusti odgovarajući zvuk. Za udarac bi još bila napravljena nova instanca particle efekta na mjestu kontakta (Slika 23).

```
    1 reference
121 public void HitRegistered(Vector3 contactPoint)
122 {
123     Instantiate(hitParticle, contactPoint, hitParticle.transform.rotation);
124     audioManager.PlayRandomWithTag("hit");
125 }
126
    1 reference
127 public void BlockRegistered(Vector3 contactPoint)
128 {
129     audioManager.PlayRandomWithTag("block");
130 }
```

Slika 23 - Pozvane metode na GM

Slika 24 prikazuje primjer izvedbe udarca nogom za vrijeme simulacije igre.

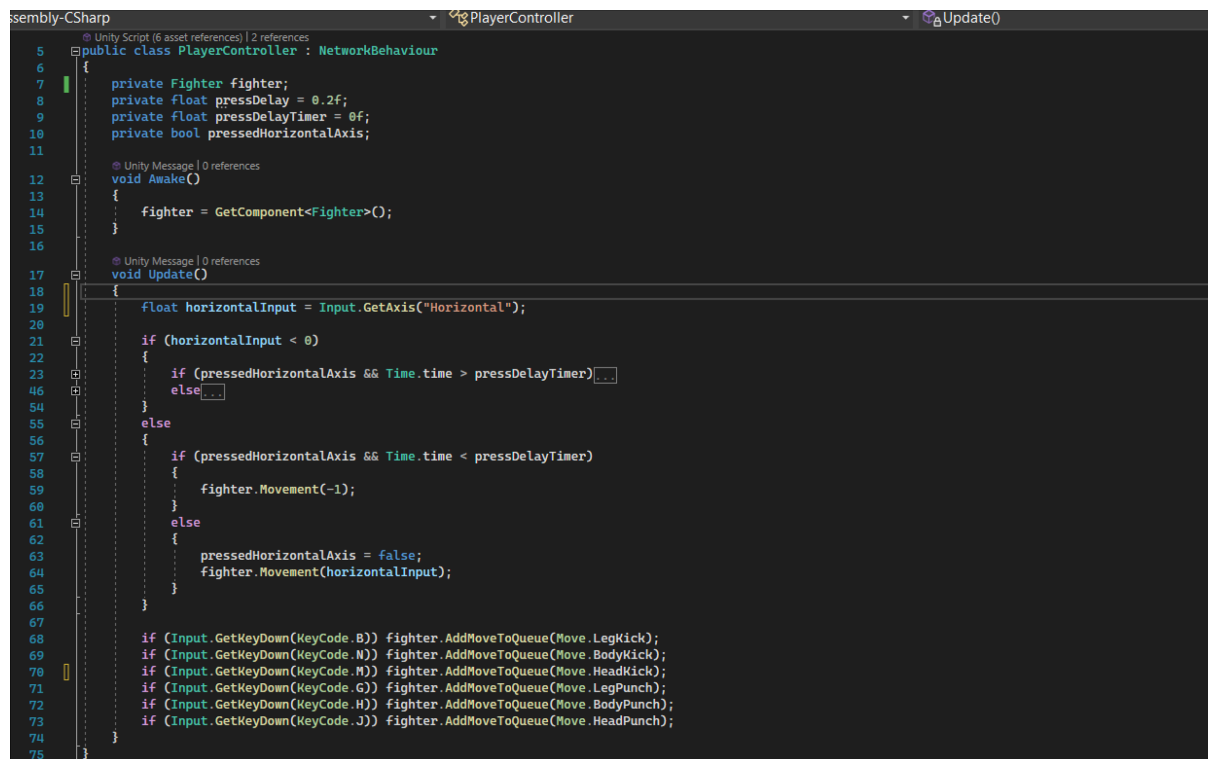


Slika 24 - Izvedba udarca u simulacij

6 Kontroler inputa

6.1 Igračev kontroler

U prošlom poglavlju govorilo se na koji način klasa "Fighter" izvodi i registrira udarce. No primanje inputa od igrača i prosljeđivanje toj klasi omogućeno je izradom PlayerController klase (Slika 25) i pomoću Unity Input Managera [16].



Slika 25 - Player Controller

Kao što se vidi, unutar funkcije Update provjerava se da li UnityEngine.Input.GetAxis registriran pritisak na horizontalnoj osi. Unity će ovisno o intenzitetu i vremenu pritiska dodijeliti toj osi broj u rasponu [-1,1]. Za promjenu osjetljivosti i tipki koje se računaju za pojedinu os može se ići na Project Settings > Input Manager > Axes. Ukoliko je vrijednost osi manja od nula postoje dvije mogućnosti:

- Igrač želi blokirati udarac - tada će držati tipku dovoljno dugo da vrijeme prijeđe timer
- Igrač želi ići prema nazad - samo će stisnuti tipku tako da vrijeme neće prijeći timer

Implementacija za gornje nije toliko bitna tako da je taj dio uvučen i ne vidi se na slici.

Nakon osi, gleda se pritisak pojedinih ključnih tipki na tastaturi. U slučaju da je registriran pritisak, poziva se AddMoveToQueue metoda i predaje joj se udarac koji odgovara toj tipki.

6.2 AI kontroler

Kada je potrebno da likom upravlja računalo umjesto igrača, jednostavno će se zamijeniti PlayerController klasa sa AIController klasom. AIController pozivat će iste metode kao što bi to radio igrač pritiskom na tipke.

Za naš projekt napravili smo jednostavnog reaktivnog agenta. Logika reaktivnog agenta je ništa drugo nego jedan složeni diagram toka. Agent će u ovoj igri raspolagati s tri težine - easy, medium, hard. Ovisno o težini, on će imati pristup kvalitetnijim setom odluka.

```
        Unity Message | 0 references
26      private void Update()
27      {
28          if (blockInExecution != Block.None) fighterController.TryToBlock(blockInExecution);
29      }
30
        Unity Message | 0 references
31      void FixedUpdate()
32      {
33          cycleCooldown = ++cycleCooldown % 10;
34          if (cycleCooldown != 0) return;
35          playerDistance = gameObject.transform.position.x - gameManager.player.transform.position.x;
36          inHandReach = playerDistance <= HAND_REACH;
37          inLegReach = playerDistance <= LEG_REACH;
38          randomNumber = Random.Range(1, 100);
39
40          if (difficulty == level.hard)
41          {
42              if (Time.time < blockCooldown) return;
43              if (inLegReach)
44              {
45                  Block block = gameManager.player.GetBlock();
46                  if (randomNumber <= 50 && block != Block.None)
47                  {
48                      blockInExecution = block;
49                      blockCooldown = Time.time + 1f;
50                  }
51                  else
52                  {
53                      blockInExecution = Block.None;
54                      if (inHandReach)
55                      {
56                          if (randomNumber <= 30) fighterController.AddMoveToQueue(Move.LegPunch);
57                          else if (randomNumber <= 50) fighterController.AddMoveToQueue(Move.HeadPunch);
58                          else if (randomNumber <= 65) fighterController.AddMoveToQueue(Move.BodyPunch);
59                      }
60                      else
61                      {
62
63                          if (fighterController.stamina > 35)
64                          {
65                              if (randomNumber <= 30) fighterController.AddMoveToQueue(Move.BodyKick);
66                              else if (randomNumber <= 50) fighterController.AddMoveToQueue(Move.LegKick);
67                              else if (randomNumber <= 65) fighterController.AddMoveToQueue(Move.HeadKick);
68                          }
```

Slika 26 - AI Controller

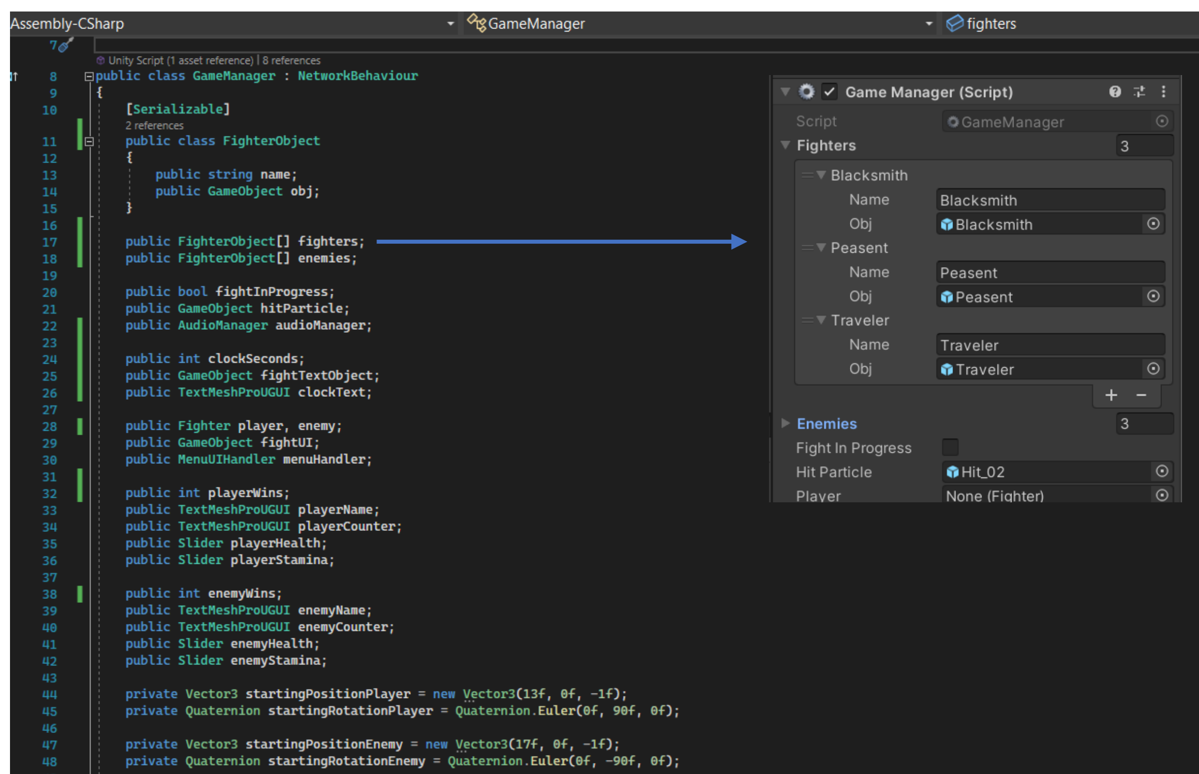
Slika 26 prikazuje dio implementacije agenta. Pošto će se u ovoj igri FixedUpdate ponavljati 50 puta u sekundi. Stavljen je na sam početak "cycleCooldown" koji bi trebao propustiti svako deseto izvođenje. Time agent ima brzinu reakcije 200 ms. Zatim se uzima pozicija igrača kroz Game Manager i računa da li je igrač unutar dosega za udarce rukom i nogom (senzorni dio agenta). Nakon toga slijedi uzimanje nasumične vrijednosti koja će se koristiti kako bi se postignuta doza nepredvidljivosti udaraca. Na kraju je implementirana logika nalik dijagramu toka kojom agent izvodi akcije.

7 Upravitelj igre

Game Manager je srce igre. Zadaće Game Managera u ovom projektu su:

- Instancirati likove-borce na polje
- Započeti borbu
- Sinkronizirati život i staminu likova-boraca s UI elementima
- Upravljati tranzicijama između borbi
- Očistiti scenu nakon borbe

Da bi se implementirali gornji zahtjevi, njegova klasa započinje s mnogobrojnim deklaracijama (Slika 27).



Slika 27 - GM deklaracije

Definirana je nova klasa FighterObject da bi se na lagan način spojili nazivi likova s njihovim objektima. Stavljen joj je atribut "[Serializable]" kako bi se mogli prikazati polja u Unity editoru (vidi plavu strelicu). Polje "fighters" sadržavat će likove koje igrač može odabrati, dok će polje "enemies" sadržavati neprijateljske likove.

7.1 Instanciranje likova

Prva stavka Game Managera je instancirati likove-borce na polje. Slika 28 prikazuje metodu koja se poziva prilikom odabira lika. Prvo se prebacuje pozadinska glazba na borbenu glazbu. Nakon toga se instancira lik s indexom predanim kao argumentom i dodjeli mu se tag "Player". Zatim se namjesti ime lika na UI text polju. Isto tako broj pobjeda se namjesti na 0 u varijabli i na UI polju. Potom se slično ponovi za neprijateljskog lika uz iznimku da se dodatno namjesti "isAI" varijabla na istinito.

```
205    public void FighterSelected(int index)
206    {
207        audioManager.Stop("Ambient");
208        audioManager.Play("Combat");
209
210        player = Instantiate(fighters[index].obj, startingPositionPlayer, startingRotationPlayer).GetComponent<Fighter>();
211        playerName.text = fighters[index].name;
212        player.tag = "Player";
213        playerCounter.text = (playerWins = 0).ToString();
214
215        enemy = Instantiate(enemies[0].obj, startingPositionEnemy, startingRotationEnemy).GetComponent<Fighter>();
216        enemy.isAI = true;
217        enemyName.text = enemies[0].name;
218        enemyCounter.text = (enemyWins = 0).ToString();
219    }
```

Slika 28 - Instantacija likova

7.2 Započinjanje borbe

Metoda StartFight aktivira UI za borbu, te poziva korutinu CountdownCoroutine. Korutina se izvodi konkurentno s ostatkom programa. U korutinama s "yield return" odgađamo izvođenje koda na željeno vrijeme. To je iskorišteno i za odbrojavanje, pošto ne postoji dostupna animacija koja bi to odbrojavanje animirala.

```
136    public void StartFight()
137    {
138        fightUI.SetActive(true);
139        fightInProgress = true;
140        StartCoroutine(CountdowCoroutine());
141    }
       2 references
142    IEnumerator CountdowCoroutine()
143    {
144        clockSeconds = 30;
145        clockText.text = clockSeconds.ToString();
146        audioManager.Play("Fight");
147        fightTextObject.GetComponent<TextMeshProUGUI>().text = "3"; yield return new WaitForSeconds(1);
148        fightTextObject.GetComponent<TextMeshProUGUI>().text = "2"; yield return new WaitForSeconds(1);
149        fightTextObject.GetComponent<TextMeshProUGUI>().text = "1"; yield return new WaitForSeconds(1);
150        fightTextObject.GetComponent<TextMeshProUGUI>().text = "Fight!";
151        fightTextObject.GetComponent<Animation>().Play("FadeOut");
152        player.Unfreeze();
153        enemy.Unfreeze();
154
155        bool endedByTime = false;
156        while (fightInProgress)
157        {
158            clockSeconds = clockSeconds - 1;
159            clockText.text = clockSeconds.ToString();
160            if (clockSeconds == 3)...
166            if (clockSeconds == 2)...
170            if (clockSeconds == 1)...
174            if (clockSeconds == 0)
175            {
176                endedByTime = true;
177                break;
178            }
179            yield return new WaitForSeconds(1);
180        }
181        if (endedByTime)
182        {
183            if (player.health > enemy.health)
184            {
185                EndFight("Enemy");
186            }
187            else
188            {
189                EndFight("Player");
190            }
```

Slika 29 - Početak borbe

Nakon odbrojavanja poziva se Unfreeze metoda u "Fighter" klasi za igrača i neprijatelja. Freeze i Unfreeze metode onemogućuju i omogućuju komponentu s klasom PlayerController i AIController nad pojedinim likom. Na taj način može se onemogućiti tj. omogućiti kretnje likova kada je to potrebno. Kada je borba započela smanjuje se vrijeme na brojaču svaku sekundu. Kada i ako brojač dođe na 0, borba je završila i u sljedećoj selekciji biti će pozvana metoda EndFight sa tagom gubitnika (igrač pobjeđuje ukoliko ima više života od neprijatelja).

7.3 Sinkronizacija života i stamine

Treća stavka koju Game Manager treba imati je sinkronizacija života i stamine. Za tu svrhu napravljene su metode HealtUpdate i StaminaUpdate (Slika 30). Njih se poziva unutar funkcije FixedUpdate. Za razliku od Update koja se poziva svaki frame, FixedUpdate se poziva fiksan broj puta. Ako bi iz nekih razloga trebalo promijeniti periodu ponavljanja, to bi se moglo učiniti u Edit > Settings > Time > Fixed Timestep. Za ovu igru nije potrebno ažurirati život i staminu svaki frame, tako da se korištenjem FixedUpdate smanjuje broj nepotrebnih komputacija.

```
// Unity Message | 0 references
void FixedUpdate()
{
    if (fightInProgress)
    {
        HealthUpdate();
        StaminaUpdate();
    }
}

// 3 references
public void HealthUpdate()
{
    playerHealth.value = player.health;
    enemyHealth.value = enemy.health;

    if (player.health <= 0)
    {
        player.Knockdown();
        EndFight("Player");
    }
    else if (enemy.health <= 0)
    {
        enemy.Knockdown();
        EndFight("Enemy");
    }
}

// 1 reference
public void StaminaUpdate()
{
    playerStamina.value = player.stamina;
    enemyStamina.value = enemy.stamina;
}
```

Slika 30 - Sinkronizacija života i stamine

StaminaUpdate metoda ažurira elemente UI koji prikazuju stamine likova. HealtUpdate čini to isto za život, i uz to još provjerava da li je jedan od likova izgubio sav život i time poražen. Ukoliko je slučaj da je lik poražen, na poraženom liku će se pozvati metoda Knockdown koja će samo javiti animatoru da započne animaciju za poraz, te će se nakon toga pozvati EndFight sa tagom gubitnika.

7.4 Tranzicije između borbi

Četvrta stavka Game Managera je upravljanje tranzicijama između borbi. Prethodno je rečeno da se u slučaja poraza poziva EndFight metoda (Slika 31). Prilikom poziva ove metode postavlja se varijabla "fightInProgress" na false. Potom se pušta audio za pobjedu ako je igrač pobijedio, odnosno za gubitak ukoliko je igrač izgubio. Ažuriraju se elementi UI-a i puštaju animacije za slavlje na pripadajućem liku. Zamrznu se kontroleri na likovima i provjerava se da li je i jedan od boraca skupio dovoljno pobjeda za kraj. Ukoliko je kraj, poziva se odgovarajuća metoda za tranziciju na izborniku (o izborniku će se govoriti u sljedećem poglavlju). Ukoliko nije kraj, poziva se korutina za novu rundu.

```
196    public void EndFight(string tag)
197    {
198        fightInProgress = false;
199
200        if (tag == "Player")
201        {
202            audioManager.Play("GameOver");
203            fightTextObject.GetComponent<TextMeshProUGUI>().text = "Game Over";
204            fightTextObject.GetComponent<CanvasGroup>().alpha = 1;
205            enemyCounter.text = (++enemyWins).ToString();
206            enemy.Celebration();
207        }
208        else
209        {
210            audioManager.Play("Victory");
211            fightTextObject.GetComponent<TextMeshProUGUI>().text = "Victory";
212            fightTextObject.GetComponent<CanvasGroup>().alpha = 1;
213            playerCounter.text = (++playerWins).ToString();
214            player.Celebration();
215        }
216
217        enemy.Freeze();
218        player.Freeze();
219
220        if (enemyWins == WIN_CAP)
221        {
222            menuHandler.FightLost();
223        }
224        else if (playerWins == WIN_CAP)
225        {
226            menuHandler.FightWin();
227        }
228        else
229        {
230            StartCoroutine(NewRoundCoroutine());
231        }
232    }
```

Slika 31 - Završetak borbe

Ukoliko je igrač pobijedio dovoljni broj rundi za kraj, na izborniku za kraj biti će gumb koji nudi igraču opciju da nastavi borbu sa sljedećim neprijateljem. Klikom na taj gum pozvati će se metodu ContinueToNextFight (Slika 32). Ona će pozvati metodu za odgovarajuću tranziciju kamere za sljedeću borbu ovisno o pobijeđenom neprijatelju.

```
316    public void ContinueToNextFight()
317    {
318        if (gameManager.totalWins == MINI_BOSS_TH) FightToMiniBoss();
319        else if (gameManager.totalWins == BOSS_TH) MiniBossToBoss();
320        else BossToBase();
321    }
```

Slika 32 - Nastavak borbe

7.5 Pauziranje borbe

Unity fizika izvodi se na temelju vremenske varijable unutar klase Time. Promjenom Time.timeScale na 0 efektivno se pauzira igra. Frekvencija izvođenja FixedUpdate funkcija isto tako ovisi o timeScale varijabli pa su time i one pauzirane. Za prestanak pauze potrebno je vratiti timeScale nazad na vrijednost 1. Slika 33 prikazuje implementaciju pauze.

```
401    public void PauseGame()
402    {
403        if (Time.timeScale == 0)
404        {
405            Time.timeScale = 1;
406            pauseMenu.SetActive(false);
407        }
408        else
409        {
410            Time.timeScale = 0;
411            pauseMenu.SetActive(true);
412        }
413    }
```

Slika 33 - Pauza

7.6 Čišćenje scene

Nakon završetka borbe, kada igrač klikne na UI izborniku gumb za izlaz u glavni izbornik, pozvati će se metoda FightEndedCleanup (Slika 34) čija je zadaća maknuti UI za borbu, promijeniti glazbu, te uništiti instance likova.

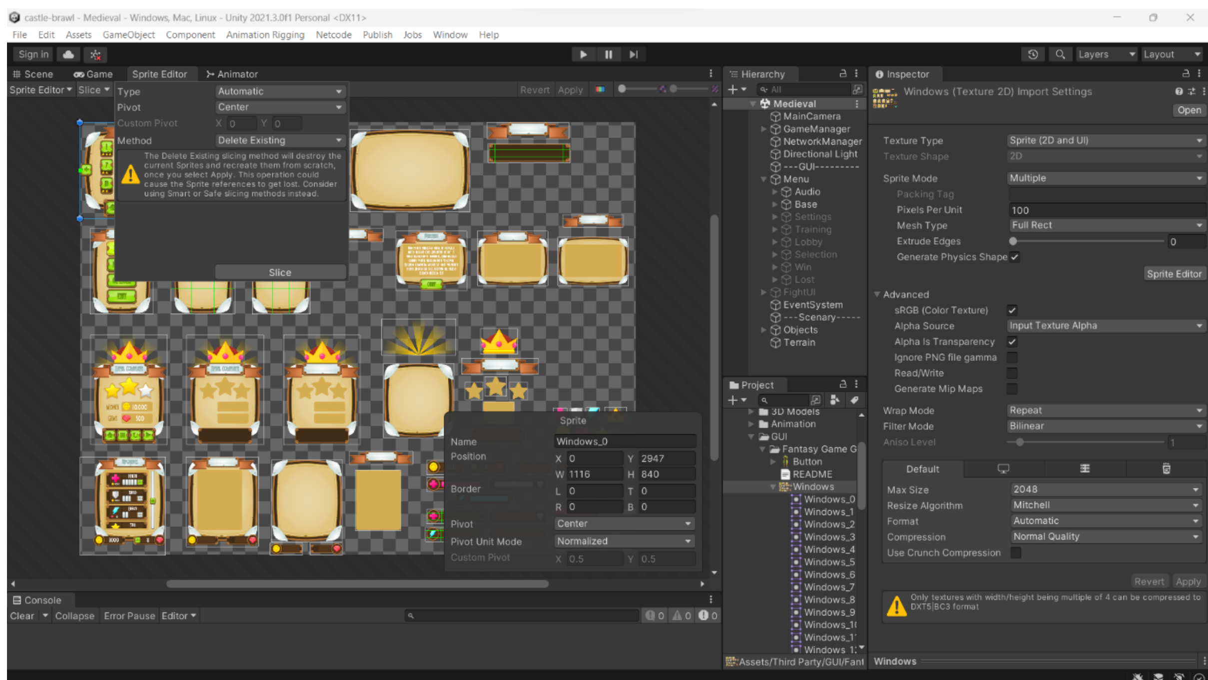
```
277    public void FightEndedCleanup()
278    {
279        fightUI.SetActive(false);
280        audioManager.Play("Ambient");
281        audioManager.Stop("Combat");
282        Destroy(player.gameObject);
283        Destroy(enemy.gameObject);
284    }
```

Slika 34 - Cleanup

8 Izbornik

8.1 Sprites

Za elemente izbornika skinut je prikladan asset u obliku .png formata [17]. Slika 35 prikazuje preuzeti asset otvoren u Unity editoru. Rasterski format slike koji se može rastaviti na manje dijelove naziva se sprite. Vektorski formati ne mogu biti sprite-ovi jer zbog načina na koji su formatirani onemogućuju rezanje na manje dijelove (nisu bitmapa).



Slika 35 - Sprites

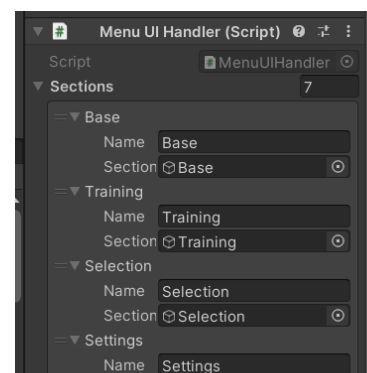
Prva stvar koju treba napraviti sa sprite assetom je mapirani pojedine manje dijelove unutar slike. To je moguće napraviti klikom na Sprite Editor > Slice > [odabiranje postavki] > Slice > Apply. Za ovaj projekt je u postavkama stavljeno da tip rezanja bude automatski. Nakon automatskog rezanja par rubova nije bilo dobro prepoznato, te ih je bilo potrebno manualno namjestiti.

8.2 Kanvas

Prvo treba dodati kanvas u scenu klikom na GameObject > UI > Canvas. Za ovaj projekt kanvas objekt je nakon dodavanja preimenovan u "Menu". Da bi dimenzije elemenata uvijek ostale iste naspram ekrana, u komponenti Canvas Scaler treba namjestiti opciju Scale With Screen Size. Nakon toga, na kanvas objekt dodane su još dvije nove komponente Canvas Group i Animation. U Animation komponenti dodane su dvije nove animacije "FadeIn" i "FadeOut" - naime jedna namješta Alpha parametar iz Canvas Group sa 0 na 1 linearno kroz jednu sekundu, a druga radi inverziju prve. Slika 36 prikazuje Menu kanvas nakon namještanja komponenti. U inspektor prozoru vidljivi su parametri triju navedenih komponenti: Canvas Scaler, Canvas Group, i Animation.

Slika 36 - GUI

Ispod komponente Animation vidljiva je komponenta Menu UI Handler. Gledajući na nju u inspektoru dobivamo parametre kao na slici desno (Slika 37). Na slici možemo vidjeti sekcije gdje su spojeni nazivi sekcija s objektima unutar hijerarhije objekata iz Menu kanvasa. Vidljivo je 7 sekcija koje će biti sadržane u igri: baza, vodič, opcije, selekcija, lobi, te sekcija za pobjedu i sekcija za izgubljenu borbu.



Slika 37 - Sekcije u inspektoru

Unutar klase Menu UI Handler, sekcije su implementirane kao ugniježđena klasa.

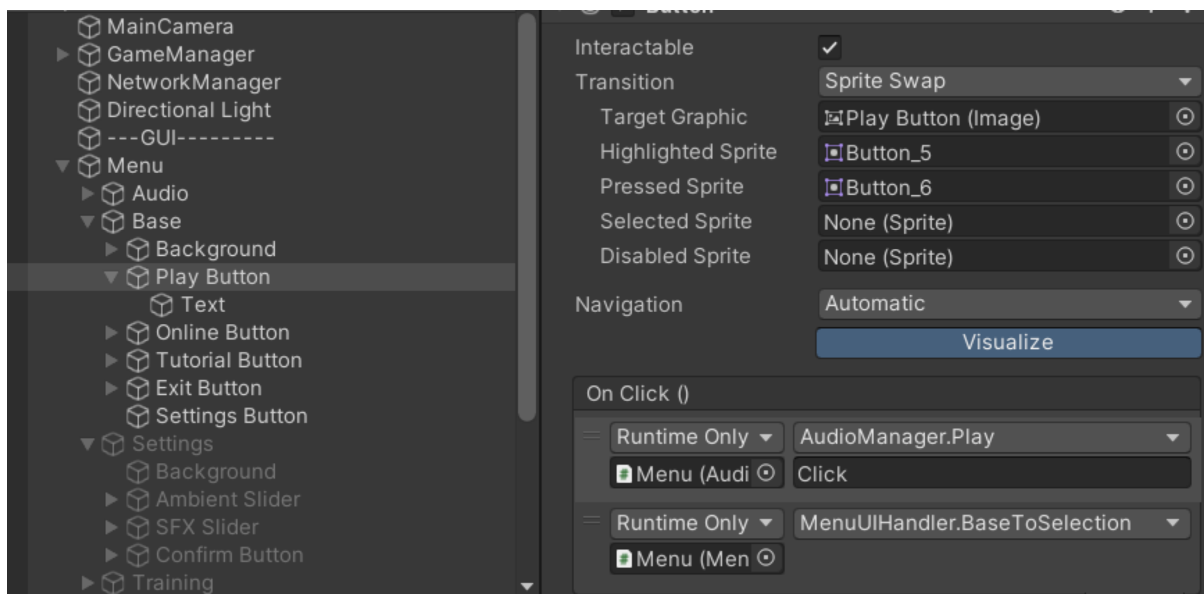
```
[Serializable]
1 reference
public class Section
{
    public string name;
    public GameObject sectionObject;
}
```

Slika 38 - Section

Razlog zašto su referencirani objekti iz scene kao sekcije je taj što su u tim objektima kreirani UI elementi za pojedine prozore izbornika. Kako korisnik bude klikao po izborniku, trenutna sekcija biti će isključena iz scene, a nova će se uključiti.

8.3 Mijenjanje prozora izbornika

Mijenjanje prozora tj. sekcije započinje kada korisnik klikne gumb. Slika 39 prikazuje dio prozora inspektora za gumb "Play Button". Kada je dugme pritisnuto biti će emitiran OnClick event i time pozvane sve metode koje slušaju na taj event. U primjeru je vidljivo da su za slušače na "Play" dugme stavljene metode AudioManager.Play i BaseToSelection.



Slika 39 - Gumb

Slika 40 prikazuje implementaciju metode "BaseToSelection". Pri pozivu metode postavlja se trigger parametar za animaciju kamere, poziva se "FadeOut" animacija koja je prethodno stavljena na kanvas, te se onemogućuje interakcija s kanvasom kako korisnik ne bi slučajno stisnuo neko nevidljivo dugme za vrijeme tranzicije. Isto tako poziva se korutina koja će nakon dvije sekunde promijeniti prozor izbornika (sekciju). Potom će se pozvati "FadeIn" animacija, te ponovo omogućiti interakciju s kanvasom.

```
147    public void BaseToSelection()
148    {
149        cameraAnimator.SetTrigger("BaseToSelection");
150        menuAnimation.Play("FadeOut");
151        GetComponent<CanvasGroup>().interactable = false;
152        StartCoroutine(BaseToSelectionCoroutine());
153
154        IEnumerator BaseToSelectionCoroutine()
155        {
156            yield return new WaitForSeconds(2f);
157            ChangeSection("Selection");
158            menuAnimation.Play("FadeIn");
159            GetComponent<CanvasGroup>().interactable = true;
160        }
161    }
```

Slika 40 - Mijenjanje prozora

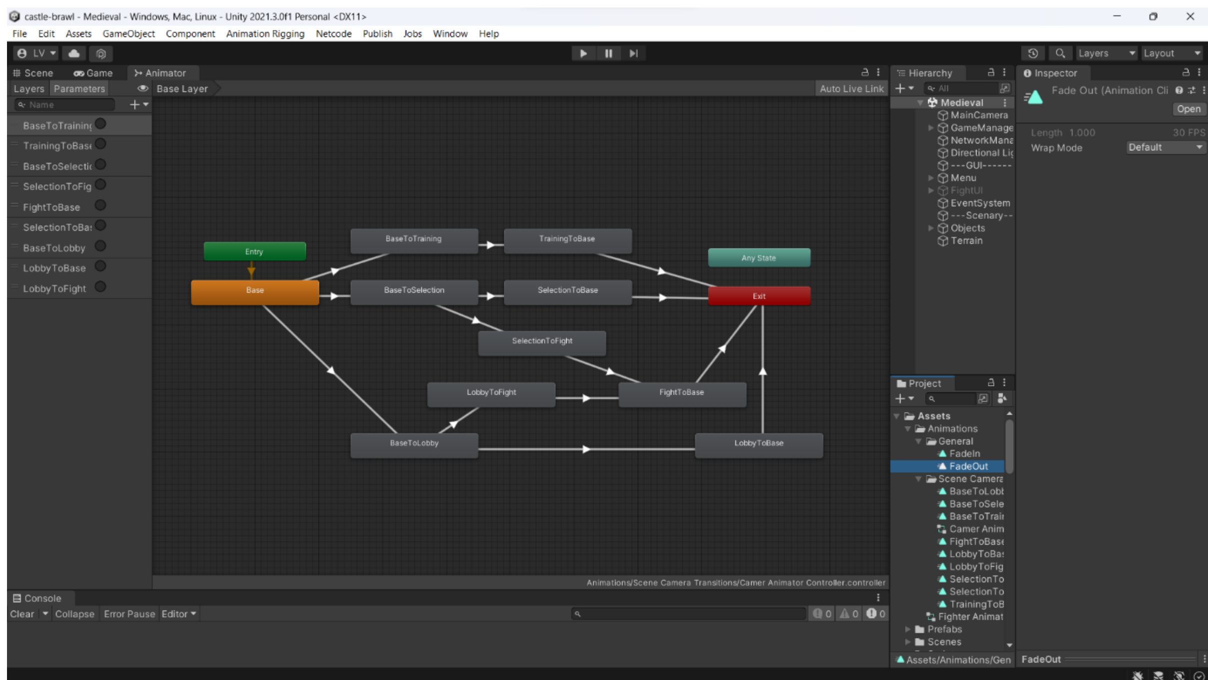
Na identičan način rade i ostale metode za mijenjanje sekcija. Slika 41 prikazuje sve metode za promjene sekcija. Na samom kraju slike vidljiva je implementacija ExitGame metode za izlazak iz programa. Dok je igra pokrenuta u editoru, treba se koristiti ExitPlaymode metodu iz UnityEditor klase. Za provjeru da li je u editoru koristi se pretprocesorska direktiva IF.

```
       Unity Message | 0 references
31     private void Awake()[...]
       0 references
37     public void BaseToSettings()[...]
       0 references
51     public void SettingsToBase()[...]
       0 references
65     public void BaseToTraining()[...]
       2 references
80     public void TrainingToBase()[...]
       0 references
95     public void BaseToLobby()[...]
       0 references
110    public void LobbyToBase()[...]
       1 reference
125    public void LobbyToFight()[...]
       0 references
139    public void BaseToSelection()[...]
       1 reference
154    public void SelectionToBase()[...]
       1 reference
169    public void SelectionToFight()[...]
       0 references
183    public void FightToBase()[...]
       2 references
199    public void FightWin()[...]
       2 references
205    public void FightLost()[...]
       0 references
211    public void ExitGame()
212    {
213        GetComponent<CanvasGroup>().interactable = false;
214        StartCoroutine(ExitGameCoroutine());
215
216        IEnumerator ExitGameCoroutine()
217        {
218            yield return new WaitForSeconds(0.5f);
219        #if UNITY_EDITOR
220            UnityEditor.EditorApplication.ExitPlaymode();
221        #else
222            Application.Quit();
223        #endif
224        }
```

Slika 41 - Menu UI Handler metoda

8.4 Animacije tranzicija prozora izbornika

U prošlom potpoglavlju je bila prikazana metoda za promjenu menu sekcije. Unutar nje je u jednom momentu namješten trigger parametar na animatoru od kamere za tranziciju sekcije. To je zato što smo je napravljen efekt da kad korisnik klikne određeni gumb za novi prozor, kamera promijeni položaj na predodređeno mjesto unutar scene.

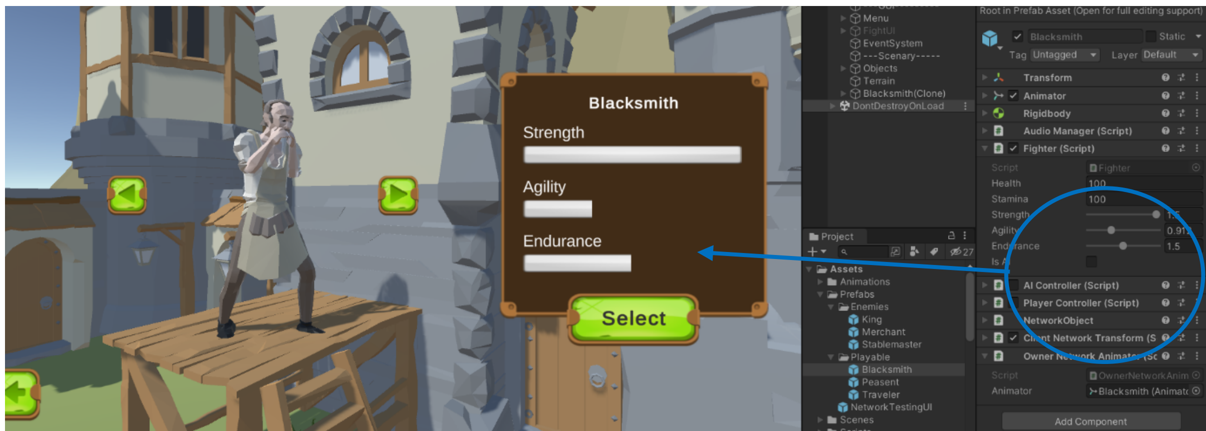


Slika 42 - Animator kamere

Slika 42 prikazuje kontroler animacija za kameru. Stanje "Base" nema nikakvu pridruženu animaciju jer će kamera već biti na početnoj poziciji kad se scena učita. Nakon što se uključi trigger određene tranzicije, stanje će se promijeniti i započeti će animacija pridružena tom stanju. Za razliku od animacija koje su bile stavljene na likovima, na animacijama kamere isključeno je ponavljanje (eng. loop). Ukoliko to ne bi bio slučaj, kamera bi u nedogled ponavljala tranziciju od početnog položaja do krajnjeg položaj.

8.5 Odabir likova

Prilikom izbora likova otvoren je prozor u kojem su prikazani statovi trenutno odabranog lika (Slika 43). Ti statovi su u stvari statovi koji su implementirani unutar klase Fighter (vidi plavu strelicu) za svakog pojedinog lika.



Slika 43 - Izbor likova

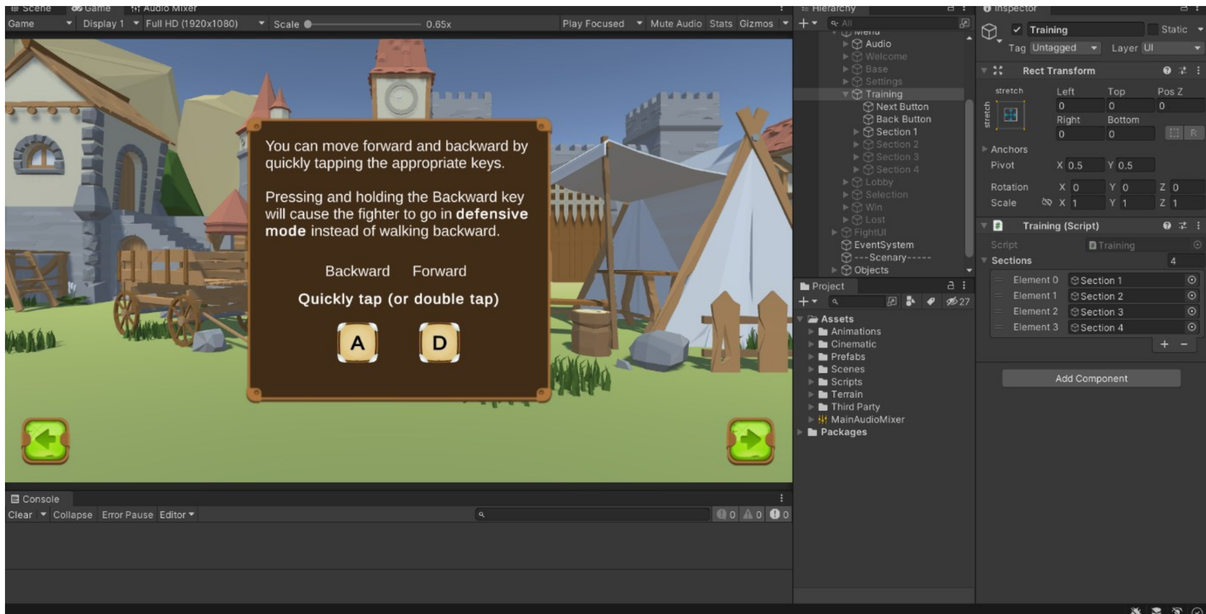
Slika 44 prikazuje metodu SetFighter koja stvara instancu trenutnog lika na sceni i postavlja UI slidere prema vrijednostima statova. Ispod nje nalaze se druge metode koje se nalaze u prozoru za izbor likova. FighterSelected metoda poziva metodu za tranziciju kamere na borbu, te istoimenu metodu upravitelja predajući mu indeks odabranog lika.

```
19  private void SetFighter()
20  {
21      Destroy(currentFighter);
22      currentFighter = Instantiate(gameManager.fighters[index].obj, position, rotation);
23      currentFighter.transform.SetParent(gameObject.transform);
24      nameText.text = gameManager.fighters[index].name;
25      strengthSlider.value = currentFighter.GetComponent<Fighter>().strength;
26      agilitySlider.value = currentFighter.GetComponent<Fighter>().agility;
27      enduranceSlider.value = currentFighter.GetComponent<Fighter>().endurance;
28  }
    Unity Message | 0 references
29  void OnEnable()
30  {
31      index = 0;
32      SetFighter();
33  }
    0 references
34  public void Previous()
35  {
36      if (--index == -1) index = gameManager.fighters.Length - 1;
37      SetFighter();
38  }
    0 references
39  public void Next()
40  {
41      if (++index == gameManager.fighters.Length) index = 0;
42      SetFighter();
43  }
    0 references
44  public void Back()
45  {
46      Destroy(currentFighter);
47      menuHandler.SelectionToBase();
48  }
    0 references
49  public void FighterSelected()
50  {
51      Destroy(currentFighter);
52      gameManager.FighterSelected(index);
53      menuHandler.SelectionToFight();
```

Slika 44 - Implementacija izbora likova

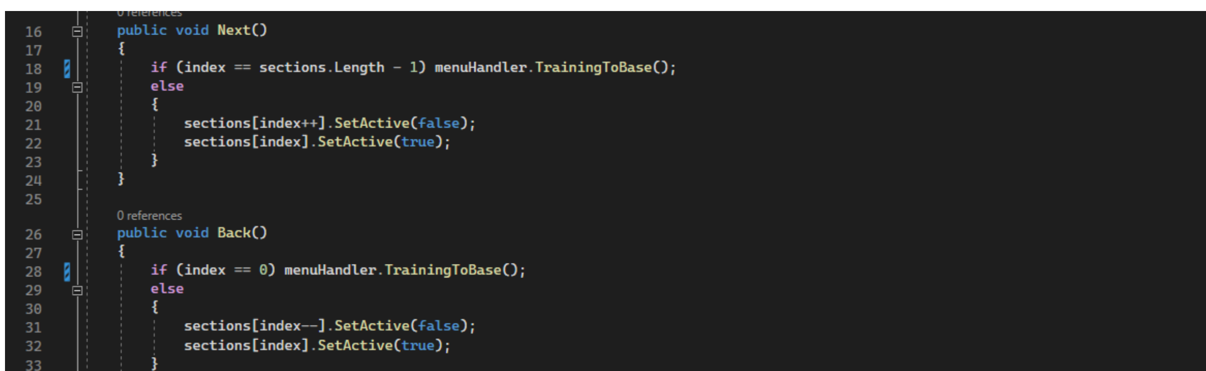
8.6 Vodič

Kroz izbornik moguće je otvoriti sekciju vodiča. Vodič je sastavljen od više panela koji su u kodu nazvane sekcijama kao i kod izbornika. Svaki panel se sastoji od pozadinske slike, texta i manjih slika elemenata. Slika 45 prikazuje izgled početnog panela.



Slika 45 - Vodič sekcija

U donjem lijevom i desnom kutu postavljeni su gumbovi za kretanje po panelima. Klikom na pojedini gumb se prvo isključuje panel na trenutnom indexu, potom mijenja index, i za kraj uključuje novi panel na novom indexu (Slika 46).



Slika 46 - Gumbovi za naprijed - nazad

9 Uvodna priča

Zanimljiva priča motivira igrača da sazna kraj. Na taj način je moguće držati igrača zainteresiranim za igru dugo nakon što mehanika postane monotona.

9.1 Tekst

Za ovu igru zamišljeno je da se prilikom pokretanja igre igraču predoči uvodna priča koja će ga motivirati da dospije do finalnog neprijatelja. U tu svrhu napisano je tri paragrafa texta koji su spremljeni u tri zasebne tekstualne datoteke. Te datoteke biti će pročitane od strane programa za vrijeme izvođenja. Na taj način ako bude trebalo mijenjati neki dio priče, neće trebati mijenjati kod, već će biti dovoljno mijenjati tekst unutar tih datoteka. Time se isto postiže raspodjela rada jer programeri neće morati biti uključeni u kreativni proces pisanja teksta.

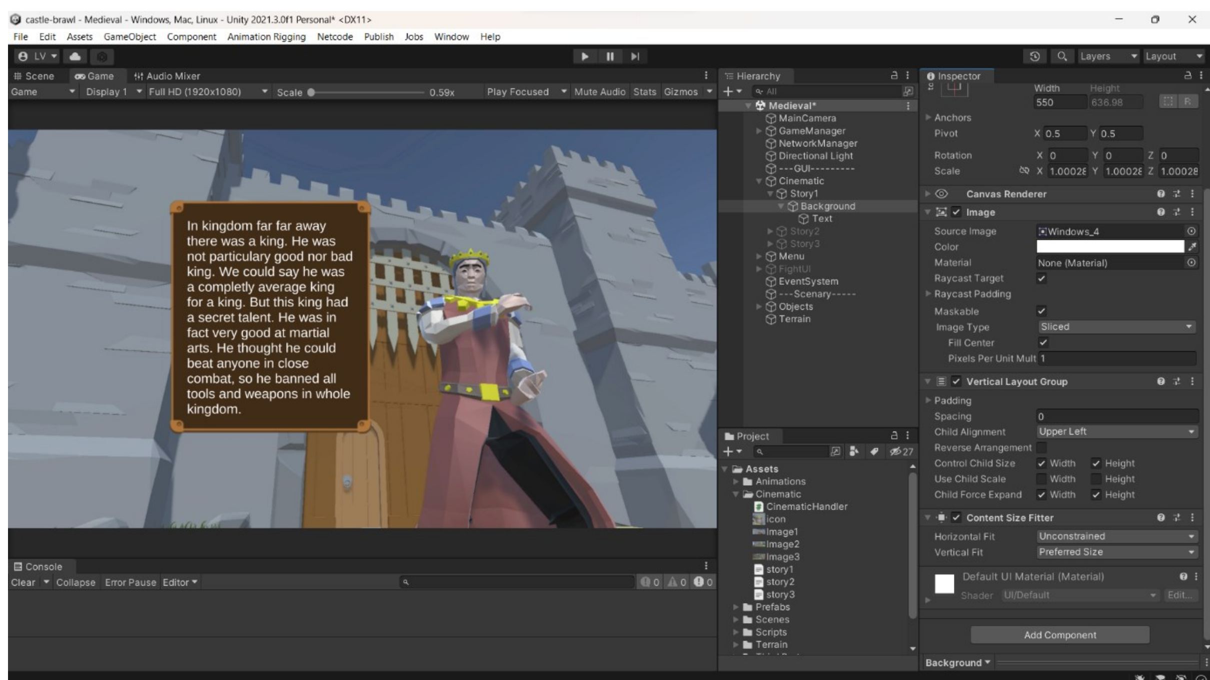
Slika 47 prikazuje definiciju nove klase CinematicHandler koja će se baviti tranzicijama UI-a u skladu s tom uvodnom pričom. Na početku klase deklarirane su potrebne varijable. U Start funkciji inicijalizirane su varijable na početna stanja, te pozvana korutina za prikaz prvog paragrafa. Korutina će u brzim vremenskim razmacima čitati znak po znak iz tekstualne datoteke i dodavati isti znak u odgovarajuće tekstualno polje na UI-u, zatim će promijeniti objekt (za sljedeći dio priče) i pozvati korutinu za sljedeći paragraf.

```
5   public class CinematicHandler : MonoBehaviour
6   {
7       public GameObject board1, board2, board3, menu;
8       public TextAsset story1, story2, story3;
9       public TextMeshProUGUI story1text, story2text, story3text;
10      readonly float TEXT_SPEED = 0.05f;
11      readonly float TRANSITION_TIME = 2f;
12
        Unity Message | 0 references
13      void Start()
14      {
15          story1text.text = "";
16          story2text.text = "";
17          story3text.text = "";
18          menu.SetActive(false);
19          StartCoroutine(Board1Coroutine());
20      }
21
        Unity Message | 0 references
22      void Update()...
53
        1 reference
54      IEnumerator Board1Coroutine()
55      {
56          board1.SetActive(true);
57          foreach (char t in story1.text)
58          {
59              story1text.text += t;
60              yield return new WaitForSeconds(TEXT_SPEED);
61          }
62          yield return new WaitForSeconds(TRANSITION_TIME);
63
64          board1.SetActive(false);
65          StartCoroutine(Board2Coroutine());
66      }
        2 references
67      IEnumerator Board2Coroutine()
68      {
69          board2.SetActive(true);
70
71          foreach (char t in story2.text)
```

Slika 47 - Tranzicije u uvodu

9.2 Pozadina

Dobiven je efekt pisanja teksta priče u tekstualno polje UI elementa. Za ovu igru zamišljeno je da se pozadina tekstualnog polja širi prilikom ispisivanja teksta priče. Jedan od načina da se napravi efekt proširivanja pozadine ovisno o veličini teksta je da se objekt s tekstom postavi kao dijete od objekta s pozadinom. Zatim se objektu s pozadinom doda komponenta Vertical Layout Group i postavi Control Child Size polje na istinito za visinu i za širinu. Zatim se još treba dodati komponenta Content Size Fitter i postaviti polje Vertical Fit na Preffered Size (Slika 48). Alternativni pristup bio bi napraviti animaciju koja će proširivati visinu pozadine u nekom vremenskom intervalu.

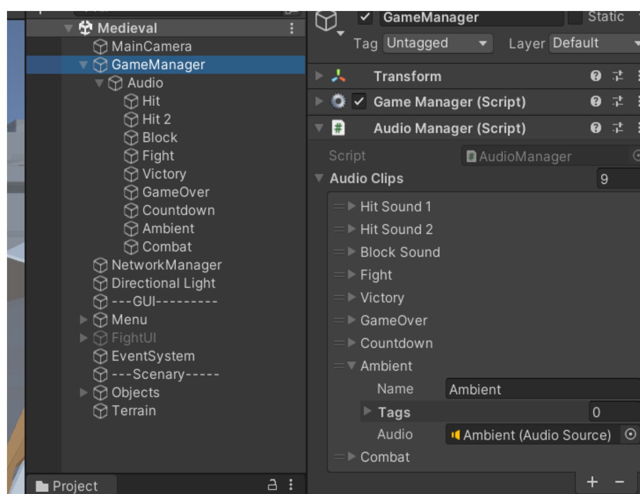


Slika 48 - Veličina konteksta

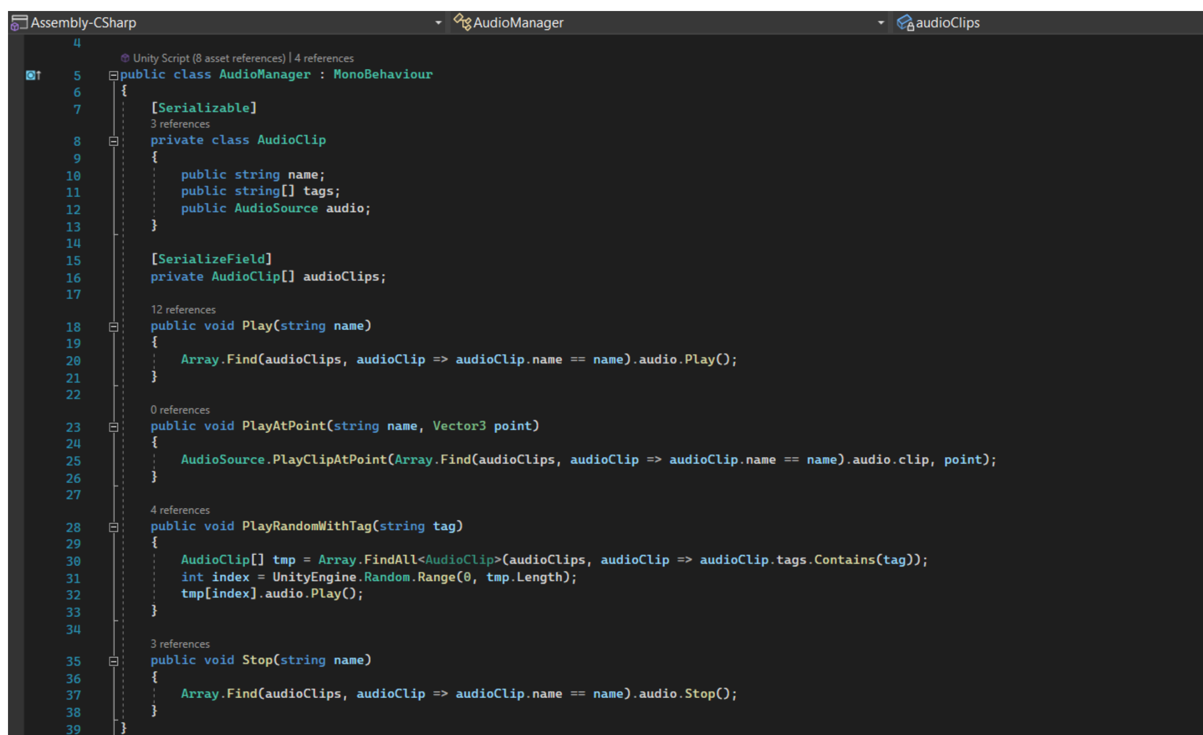
10 Audio

10.1 Audio upravitelj

Svakom objektu na koji se želi dodati audio dodana je komponenta Audio Manager. Zatim mu je dodan novi prazni objekt kao dijete. Radi lakše navigacije objekt je nazvan "Audio". U taj Audio objekt su potom za svaki zvuk ili melodiju koja je bila potrebna dodavani novi objekti s komponentom Audio Source (Slika 49). Za kraj su u inspektor prozoru Audio Managera dodane reference na sve te Audio Source objekte napravljene u prethodnom koraku. Svakoj referenci je isto tako upisan željeni naziv i željeni tag. Slika 49 isto tako prikazuje primjer otvorenog inspektora od komponente Audio Manager. Objekti s Audio Source komponentom vidljivi su u hijerarhijskom prozoru s lijeve strane slike kao djeca Audio objekta.



Slika 49 - Audio objekti



```
public class AudioManager : MonoBehaviour
{
    [Serializable]
    private class AudioClip
    {
        public string name;
        public string[] tags;
        public AudioSource audio;
    }

    [SerializeField]
    private AudioClip[] audioClips;

    public void Play(string name)
    {
        Array.Find(audioClips, audioClip => audioClip.name == name).audio.Play();
    }

    public void PlayAtPoint(string name, Vector3 point)
    {
        AudioSource.PlayClipAtPoint(Array.Find(audioClips, audioClip => audioClip.name == name).audio.clip, point);
    }

    public void PlayRandomWithTag(string tag)
    {
        AudioClip[] tmp = Array.FindAll<AudioClip>(audioClips, audioClip => audioClip.tags.Contains(tag));
        int index = UnityEngine.Random.Range(0, tmp.Length);
        tmp[index].audio.Play();
    }

    public void Stop(string name)
    {
        Array.Find(audioClips, audioClip => audioClip.name == name).audio.Stop();
    }
}
```

Slika 50 - Klasa AudioManager

Slika 50 s prethodne strane prikazuje implementaciju Audio Manager komponente. U njoj je prvo definirana ugniježđena klasa AudioClip koja će sadržavati Audio Source, te naziv i tagove za taj audio klip. Zatim je deklarirano polje tih AudioClip objekata (to je upravo polje u koje su preko inspektora referencirani svi audio izvore). Nakon toga napravljene su sljedeće metode:

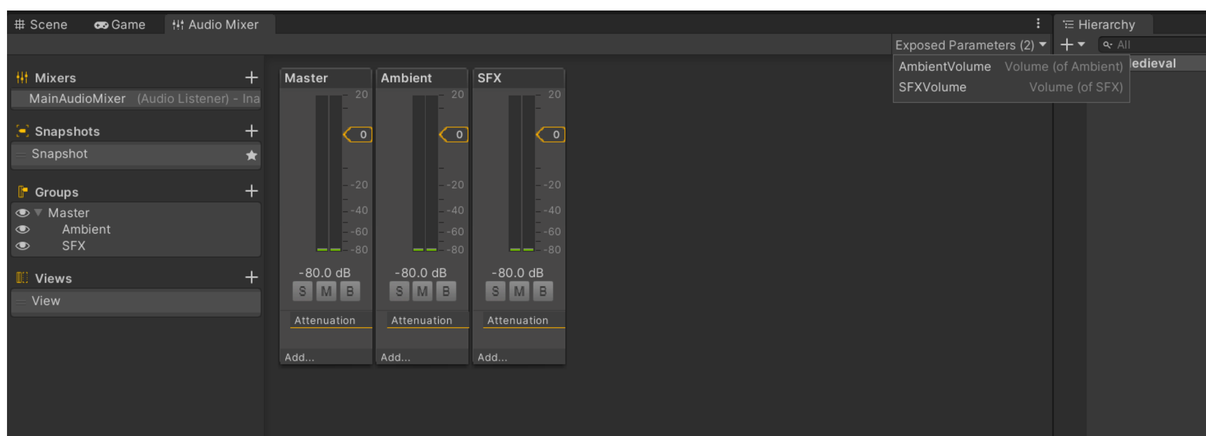
- Play - pokreni audio izvor s traženim nazivom
- PlayAtPoint - pokreni audio izvor s traženim nazivom na traženoj točci
- PlayRandomWithTag - pokreni random audio izvor s traženim tagom
- Stop - stopiraj izvođenje audio izvora

Implementacije gornjih metoda vidljive su na istoj slici, te ih zbog trivijalnosti nije potrebno posebice objašnjavati.

10.2 Audio Mixer

Da bi se kasnije moglo imati veću kontrolu nad audio izvorima potrebno je napraviti Audio Mixer [18]. Audio Mixer dodaje se klikom na Assets > Create > Audio Mixer. Nakon dodavanja Audio Mixera, za ovaj projekt su u njemu napravljene dvije nove podgrupe s nazivima "Ambient" i "SFX". Potom je za svaku izložen parametar Volumen da bi se on mogao mijenjati putem koda i samim time glasnoća grupa. Slika 51 prikazuje mixer s grupama.

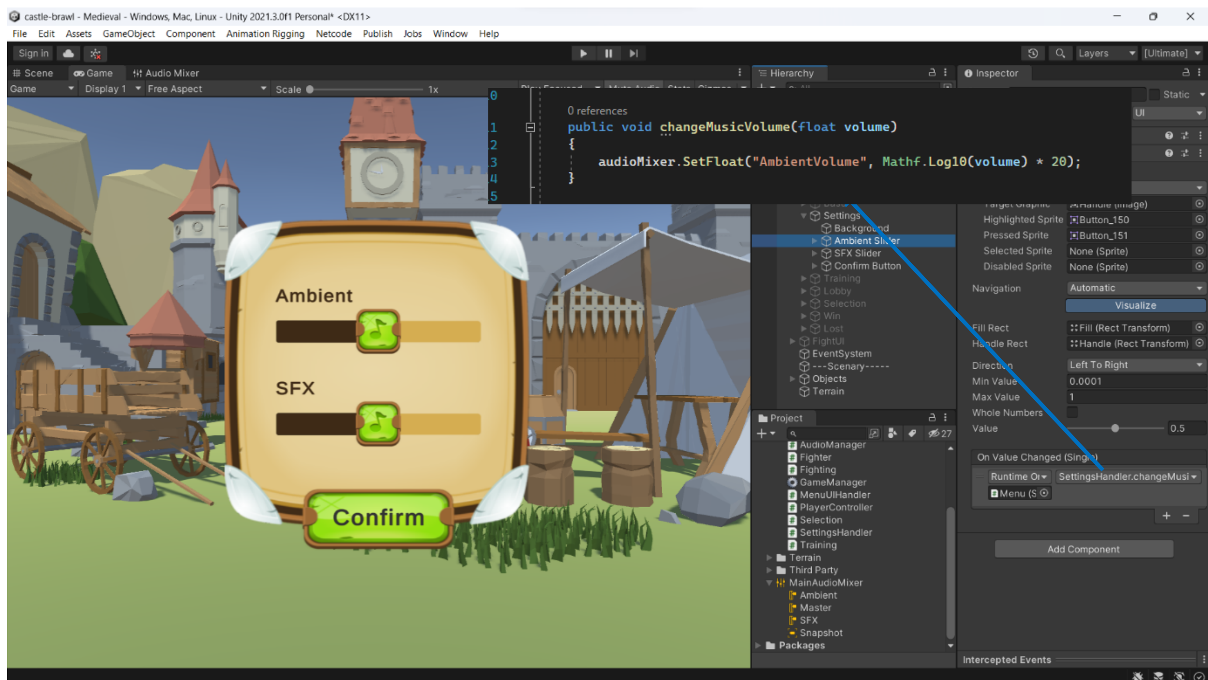
Kako bi mijenjanje volumena grupa utjecalo na volumen pojedinih audio izvora unutar igrice, na Output polju za svaku Audio Source komponentu mora se pridružiti grupa kojoj pripada. To je i učinjeno u ovom projektu, pozadinska glazba smještena je unutar grupe "Ambient", zvučni efekti kao što su udarci i slično smješteni su u grupu "SFX".



Slika 51 - Audio Mixer

Nakon što je napravljen mixer i svaki audio izvor stavljen u odgovarajuće skupine, može se koristiti volume parametar za izradu UI komponente koja će dozvoljavati korisniku mijenjanje glasnoće zvukova.

Slika 52 prikazuje UI prozor za mijenjanje glasnoće. Svaki slider pri promjeni emitira event OnValueChanged s trenutnom vrijednosti slidera kao argumentom. Kao slušači tih evenata pridružene su metode changeMusicVolume za SFX slider, te changeAmbientVolume za Ambient slider. Time je uspješno i brzo implementirana kontrola zvuka.



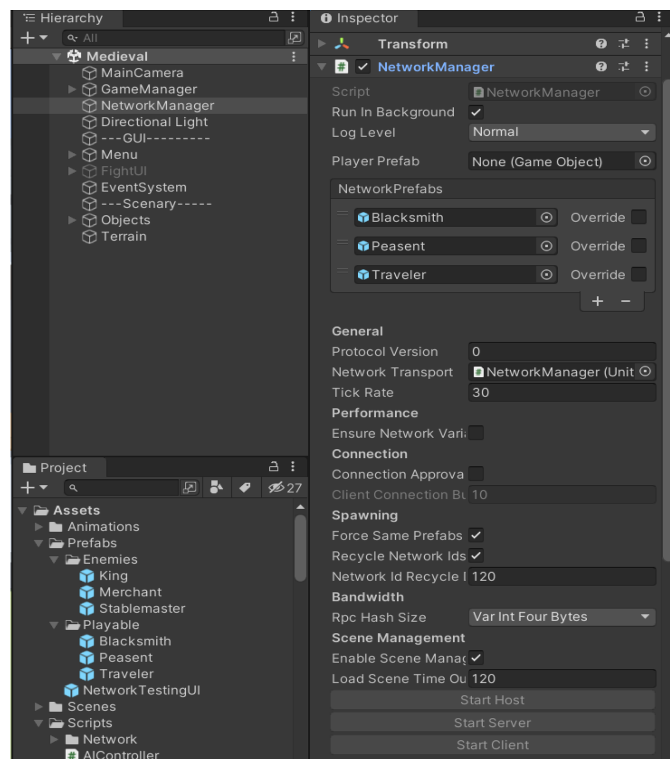
Slika 52 - Audio slideri

11 Netcode

Netcode je novija biblioteka napravljena za Unity s misijom da se apstraktira logika umrežavanja. Za izradu projekta korištena je verzija 1.0.0. Da bi se započeo rad s ovom bibliotekom, dovoljno ju dodati u projekt putem package managerea [19].

11.1 NetworkManager

Za početak, prvo je potrebno napraviti novi prazan objekt, koji se obično naziva "NetworkManager". Njemu se pridružuje istoimena komponenta Network Manager, te komponenta Unity Transport. Za ovaj projekt zadane postavke nisu mijenjane osim što su unutar Network Manager komponente, pod Network Prefabs poljem, dodani objekti likova koji će biti dostupni igračima na mreži. Slika 53 prikazuje otvoren inspektor prozor s otvorenim Network Managerom za ovaj projekt.



Slika 53 - NetworkManager

Network Manager komponenta služi kao centrala za sve osnovne Netcode mogućnosti [20]. U ovoj klasi se isto tako mogu naći reference na druge Netcode sub-sisteme. Kako bi se omogućila bilo koja mrežna akcija kao što je npr. slanje poruka, potrebno je imati server i barem jednog klijenta (ili host koji je server i klijent u isto vrijeme). Za započinjanje servera, hosta, ili klijenta, koriste se sljedeće metode:

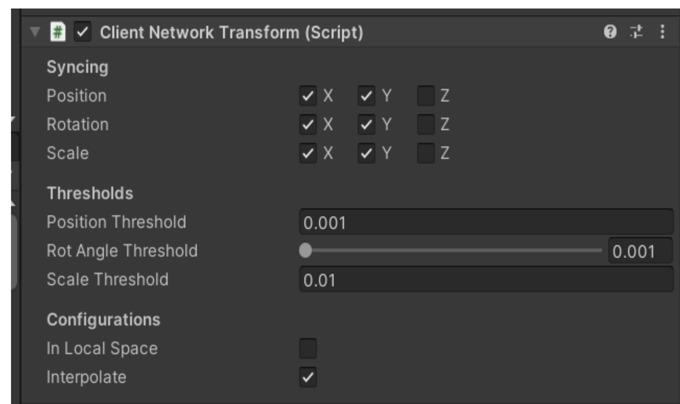
```
NetworkManager.Singleton.StartServer();
NetworkManager.Singleton.StartHost();
NetworkManager.Singleton.StartClient();
```

Prilikom startanja jedne od gornjih metoda Network Managera, za spajanje će se koristiti adresa i port broj iz klase UnityTransport.ConnectionData.

11.2 Sinkronizacija objekata

Pozicija, rotacija i veličina objekta mrežnog objekta je sinkronizirana jednom kada je objekt stvoren (spawned). Da bi se dalje nastavilo sinkronizirati taj mrežni objekt na svim klijentima, potrebno mu je dodijeliti komponentu NetworkTransform [21].

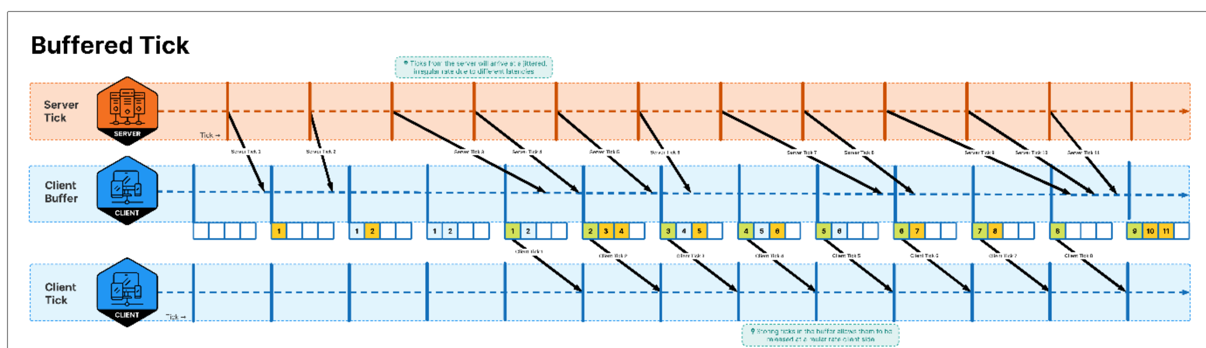
Slika 54 prikazuje dodanu Network Transform komponentu na borcima. Potrebno je bilo maknuti kvačice sa Syncing za Z os pošto se likovi-borci nikad neće micati po njoj. Da ta kvačica nije maknuta, informacije za tu os bi se nepotrebno slale preko mreže.



Slika 54 - NetworkTransport

Pod Thresholds je moguće namjestiti najmanju potrebnu promjenu vrijednosti kako bi se ona sinkronizirala. Drugim riječima, povećavajući threshold smanjuje se promet na mreži ali se isto tako smanjuje preciznost sinkronizacije.

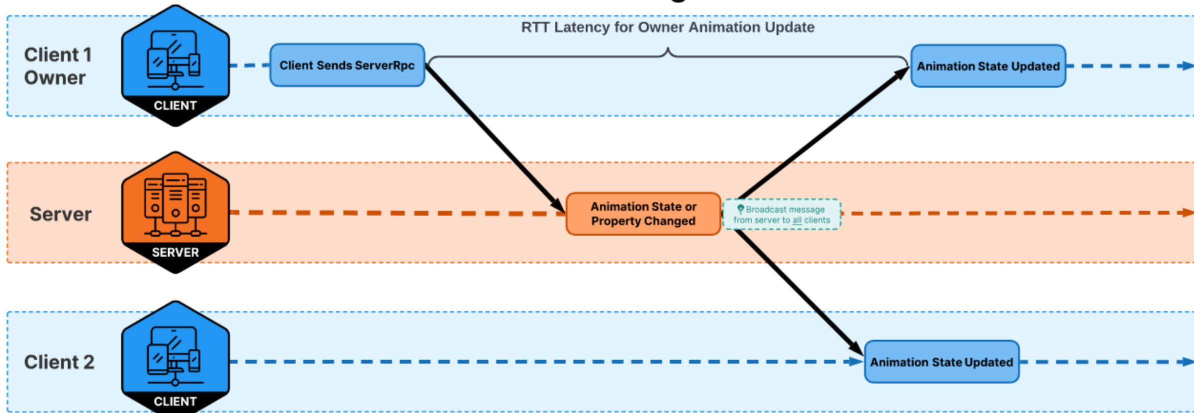
Configuration polje daje dva dodatna izbora. In local space označava da objekt sinkronizira lokalnu umjesto globalne tranzicije. Interpolation dodaje primljene promjene u buffer, te ih prije puštanja izglađuje da izgledaju prirodno bez naglih zastajkivanja (Slika 55).



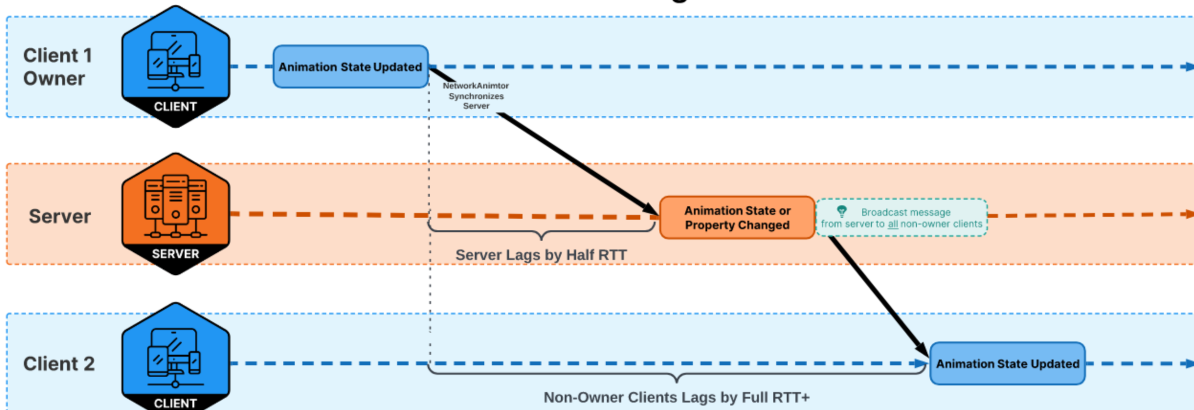
Slika 55 - Buffered tick [21]

11.3 Sinkronizacija animacija

Kada se igrači pridružuju online sesiji, objekti će biti sinkronizirani sa Animator-ovim svojstvima i stanjima, te tranzicijama u progresu. Međutim, ovisno o autoritativnom modu, ograničeno je tko može inicijalizirati promjenu stanja Animatora [22]. Postoje dva slučaja: Server Authoritative (pred-zadan) i Client Authoritative. U Server Authoritative modu, bilo koja promjena stanja animacije će biti započeta od strane servera i potom sinkronizirana sa klijentima. Pošto je server taj koji okida animaciju, klijenti koji je inicijalizirao animaciju može lagirati za otprilike jedan RTT (round trip time). Kod Client Authoritative moda klijent započinje animaciju i javlja serveru koji potom sinkronizira animacije na drugim klijentima. U ovom primjeru ostali klijenti lagiraju za jedan RTT, dok je izvorni klijent nema laga. Slijedne dijagrame za oba moda vidimo na slikama ispod.



Slika 56 - Server Authoritative [22]



Slika 57 - Client Authoritative [22]

11.4 Spajanje i započinjanje borbe

Kako bi se omogućilo korisniku da unese željenu adresu i port, potrebno je napraviti odgovarajuće UI elemente. Slika 58 prikazuje finalnu inačicu UI elemenata za spajanje u ovom projektu.

Slika 58 - Spajanje

Kada korisnik završi upisivanje u bilo koje polje za upis adrese ili porta pozvati će se metoda UpdateConnectionData koja će promijeniti adresu i port unutar komponente UnityTransport (Slika 59). Klikom na dugme Host i Client će se pozvati metode StartHost i StartClient. Klikom na dugme za nazad pozvat će se metoda Disconnect.

```
27      public void UpdateConnectionData()
28      {
29          NetworkManager.Singleton.GetComponent<UnityTransport>().SetConnectionData(
30              address.text,
31              UInt16.Parse(port.text)
32          );
33      }
        0 references
34      public void StartHost()
35      {
36          NetworkManager.Singleton.StartHost();
37      }
        0 references
38      public void StartClient()
39      {
40          NetworkManager.Singleton.StartClient();
41      }
        0 references
42      public void Disconnect()
43      {
44          NetworkManager.Singleton.Shutdown();
45      }
        0 references
46      public void MultiplayerBattleStart()
47      {
48          gameManager.FighterSelectedMultiplayer();
49          gameManager.GetComponent<NetworkGameHandler>().ToBattleClientRpc(new ClientRpcParams {
50              Send = new ClientRpcSendParams {
51                  TargetClientIds = new ulong[] { 0, 1 }
52              }
53          });
```

Slika 59 - Metode za spajanje

Klikom na dugme Start poziva se MultiplayerBattleStart. Metoda FighterSelectedMultiplayer za hosta i clienta instancira nasumičnog lika i potom zove metodu unutar komponente NetworkObjext da uspostavi spawn sa vlasništvom na mreži. Host uvijek ima id 0, a pošto je ovo igra za dva igrača klijent će imati 1. Na kraju će se pozvati FighterSelectedClientRpc metoda na svim klijentima koja će promijeniti audio unutar igre, te stavke na UI-u prije početka borbe.

```
1 reference
290 public void FighterSelectedMultiplayer()
291 {
292     int indexP1 = UnityEngine.Random.Range(0, 2);
293     GameObject player1 = Instantiate(fighters[indexP1].obj, startingPositionPlayer, startingRotationPlayer);
294     player1.GetComponent<NetworkObject>().SpawnWithOwnership(0);
295     player = player1.GetComponent<Fighter>();
296     player.tag = "Player";
297
298     int indexP2 = UnityEngine.Random.Range(0, 2);
299     GameObject player2 = Instantiate(fighters[indexP2].obj, startingPositionEnemy, startingRotationEnemy);
300     player2.GetComponent<NetworkObject>().SpawnWithOwnership(1);
301     enemy = player2.GetComponent<Fighter>();
302
303     hostObjectId.Value = player1.GetComponent<Unity.Netcode.NetworkObject>().NetworkObjectId;
304     playerObjectId.Value = player2.GetComponent<Unity.Netcode.NetworkObject>().NetworkObjectId;
305     FighterSelectedClientRpc(indexP1, indexP2);
306 }
307
308 [ClientRpc]
    1 reference
309 public void FighterSelectedClientRpc(int indexP1 = 0, int indexP2 = 0)
310 {
311     audioManager.Stop("Ambient");
312     audioManager.Play("Combat");
313
314     playerName.text = fighters[indexP1].name;
315     enemyName.text = fighters[indexP2].name;
316
317     playerCounter.text = (playerWins = 0).ToString();
318     enemyCounter.text = (enemyWins = 0).ToString();
319 }
320
```

Slika 60 - Instanciranje likova

11.5 Mrežne varijable

Jedan od načina komunikacije između servera i klijenata je uporabom mrežne varijable. Mrežnu varijablu mogu čitati svi po zadanim postavkama, ali moguće je specificirati da je čitljiva samo vlasniku objekta. Pisati u mrežnu varijablu po zadanim postavkama može samo server, no može se specificirati da to može i vlasnik objekta koji nije nužno server [23].

Ispod vidimo primjer korištenja mrežne varijable u ovom projektu kako bi se moglo pratili ID za host i client igrača.

```
78    public NetworkVariable<ulong> hostObjectId = new NetworkVariable<ulong>(0);
79    public NetworkVariable<ulong> playerObjectId = new NetworkVariable<ulong>(0);
```

Slika 61 - NetworkVariable

11.6 RPCs

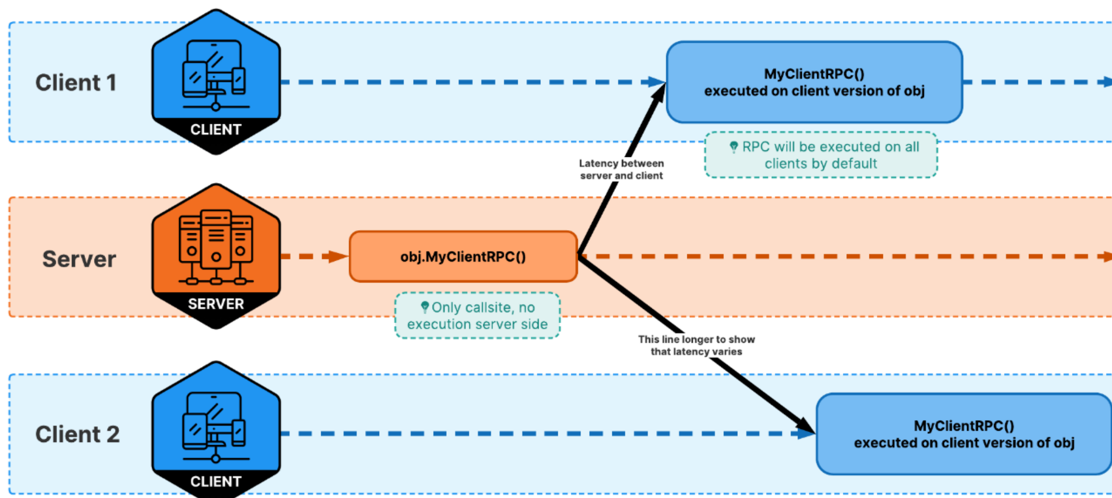
RPC (Remote Procedure Call) je protokol kojim se poziva procedura u programu koji se nalazi na drugom računalu na mreži [24].

Prema dokumentaciji [25], odabir između korištenja RPC i mrežne varijable se svodi na sljedeće:

- RPC se koriste za evente u tranziciji, informacije korisne samo u momentu primanja.
- Mrežne varijable se koriste za uporna stanja, informacije koje će trajati duže od momenta.

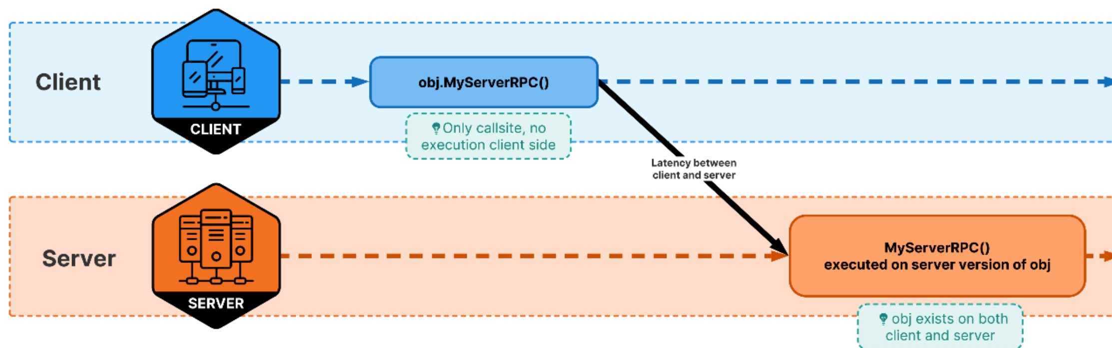
Netcode sadrži dvije varijante RPC poziva: server i client.

Kad je u pitanju server, on može pozvati client RPC na mrežnom objektu. Taj poziv će biti stavljen u lokalni red i potom poslan svim client mašinama, na kojima će se potom i izvršiti. Ukoliko se naredba ne želi poslati svim spojenim mašinama, u RPC pozivu je potrebno specificirati primatelje.



Slika 62 - Client RPCs [24]

Client može pozvati server RPC metodu na mrežnom objektu. Taj poziv će biti stavljen u lokalni red čekanja i potom poslan na server, gdje će biti izvršen na serverskoj verziji tog mrežnog objekta.



Slika 63 - Server RPCs [24]

Da bi se označile RPC procedure moraju se koristiti atributi ClientRpc i ServerRpc. Slika 64 prikazuje RPC procedure koje su napravljene za potrebe naše igre. Ove procedure su gotovo identične istoimenim metodama prije viđenim u radu, s time da su izmijenjeni dijelovi koji rade za RPC pozive. RPC metode uz to što moraju imati potreban atribut, moraju imati naziv koji sadržava sufiks RPC atributa.

```
399
400     [ClientRpc]
        1 reference
401     public void FighterSelectedClientRpc(int indexP1 = 0, int indexP2 = 0)...
412
413     [ClientRpc]
        1 reference
414     public void HitRegisteredClientRpc(Vector3 contactPoint)...
419
420     [ServerRpc(RequireOwnership = false)]
421
        1 reference
422     public void HitRegisteredServerRpc(Vector3 contactPoint)...
426
427     [ClientRpc]
        4 references
428     public void EndFightClientRpc(string tag, ClientRpcParams clientRpcParams = default)...
469
470     [ClientRpc]
        1 reference
471     public void StartFightClientRpc(ClientRpcParams clientRpcParams = default)...
482
483     [ServerRpc(RequireOwnership = false)]
        1 reference
484     public void ChangePlayerHealthServerRpc(bool isHost, float value)...
522
523     [ServerRpc(RequireOwnership = false)]
        3 references
524     public void ChangePlayerStaminaServerRpc(bool isHost, float value)...
537
```

Slika 64 - RPCs

12 Zaključak

Prateći ovaj rad pokazan je opseg posla koji je ključan za izradu računalnih igara. Izrada je započela uređivanjem scene, potom je slijedio razvoj sustava borbe koji čini mehaniku igre. Nakon izrade mehanike napravljen je kontroler za inpute igrača, te upravitelj igre koji brine se o svim aspektima borbe. Potom je napravljen izbornik i upravitelj zvukom. Igra je završena izradom funkcionalnosti za borbu protiv drugih igrača putem mreže. U svakom od tih koraka ostalo je mjesta za unapređenje. Međutim, vrijeme i dostupni resursi uvelike ograničavaju dubinu i kvalitetu krajnjeg produkta. Primjerice, kako sad stoji za mrežnu igru, potrebno je omogućiti u Firewall postavkama prosljeđivanje paketa za dogovoreni port, što možda neće biti trivijalan zadatak za svakog korisnika, a nije ni najsigurnije rješenje. Po tom pitanju potrebno je napraviti relay proxy server koji će služiti kao sigurni posrednik između klijenata (ili koristiti već gotova rješenja kao Unity Relay servis).

Nadalje, da nije korišten game engine izrada igre bila bi nezamislivo teži proces. Na primjer, kada je trebalo dodati collider, dodana je odgovarajuća komponenta i time pošteđen posao osmišljavanja i implementacije iste. Isto tako, treba primijetiti da su skripte napravljene u ovom projektu dodavane objektima kao i same komponente. To je zato što one u stvari i jesu komponente. Naime, prilikom definiranja klase u skripti uvijek je nasljeđivana jedna od dvije klase: MonoBehaviour ili NetworkBehaviour (koja nasljeđuje MonoBehaviour). Klasa MonoBehaviour nasljeđuje klasu Behavior koja nasljeđuje klasu Component. To je primjer multi-level nasljeđivanja. Svaka naslijeđena klasa proširuje prijašnju sa novim svojstvima. Na taj način je iz prve ruke vidljiva efikasnost objektno-orijentiranom programiranja. Štoviše, sama arhitektura Unity sustava je vrlo lijep primjer arhitekture bazirane na komponentama (eng. component-based architecture).

Mogućnosti Unity editora idu dalje od same izrade igara. Umjesto scene koja je napravljena za ovu igru, mogla su se importati puno realističniji modeli dijelova kuće, te složiti scenu po zadanim arhitektskim nacrtima i koristiti Unity za izradu rendera za taj projekt. Na sličan način bi se Unity mogao koristiti za izradu animiranih filmova. U tom slučaju modele i animacije bi se mogle uvesti u scenu, a Unity bi se koristio za svijetlo i kameru. U svakom slučaju pravilnije je gledati Unity kao jedan ekosustav koji omogućuje kreatorima spajanje multimedijalnog sadržaja u 2D i 3D prostoru, nego samo kao alat za izradu video igara.

REFERENCE

[1] Deloitte Digital, "Global media revenue 2020, by category," December 2019. [Online]. Dostupno: https://www.statista.com/statistics/1132706/media-revenue-worldwide.

[2] SimilarWeb, "Total global visitor traffic to Twitch.tv 2021-2022," June 2022. [Online]. Dostupno: https://www.statista.com/statistics/507849/twitch-visits-worldwide.

[3] Wikipedia contributors, "PC game," [Online]. Dostupno: https://en.wikipedia.org/wiki/PC_game.

[4] Capcom Co., "Street Fighter V Official Online Manual," capcom.com, [Online]. Dostupno: http://game.capcom.com/manual/sfv/en-uk/.

[5] Unity Technologies, "Unity’s interface," [Online]. Dostupno: https://docs.unity3d.com/2021.3/Documentation/Manual/UsingTheEditor.html.

[6] Unity Technologies, "Asset Metadata," [Online]. Dostupno: https://docs.unity3d.com/Manual/AssetMetadata.html.

[7] GitHub, Inc., "About issues," [Online]. Dostupno: https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.

[8] M. Vikainis, "3D Cartoon Village," 2018. [Online]. Dostupno: https://assetstore.unity.com/packages/3d/environments/fantasy/3d-cartoon-village-126869.

[9] Unity Technologies, "Prefabs," [Online]. Dostupno: https://docs.unity3d.com/Manual/Prefabs.html.

[10] CraftPix, "Free Medieval 3D People Low Poly Models," craftpix.net, 2021. [Online]. Dostupno: https://craftpix.net/freebies/free-medieval-3d-people-low-poly-models.

[11] Magicpot Inc., "Fighting Motions Vol.1," Magicpot Inc., 2016. [Online]. Dostupno: https://assetstore.unity.com/packages/3d/animations/fighting-motions-vol-1-76699.

[12] Unity Technologies, "Animator Controller," [Online]. Dostupno: https://docs.unity3d.com/2021.3/Documentation/Manual/class-AnimatorController.html.

[13] Unity Technologies, "Animation Layers," [Online]. Dostupno: https://docs.unity3d.com/2021.3/Documentation/Manual/AnimationLayers.html.

[14] Unity Technologies, "MonoBehaviour," [Online]. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html.

[15] Unity Technologies, "Introduction to collision," [Online]. Dostupno: https://docs.unity3d.com/Manual/CollidersOverview.html.

[16] Unity Technologies, "Input Manager," [Online]. Dostupno: https://docs.unity3d.com/Manual/class-InputManager.html.

[17] Z. Alfitra, "Free Fantasy Game GUI," [Online]. Dostupno: https://www.gameart2d.com/free-fantasy-game-gui.html.

[18] Unity Technologies, "Audio Mixer," [Online]. Dostupno: https://docs.unity3d.com/Manual/AudioMixer.html.

[19] Unity Technologies, "Installation," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/installation/install.

[20] Unity Technologies, "NetworkManager," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/components/networkmanager.

[21] Unity Technologies, "NetworkTransform," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/index.html.

[22] Unity Technologies, "NetworkAnimator," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/1.0.0/components/networkanimator/index.html.

[23] Unity Technologies, "NetworkVariable," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/index.html.

[24] Unity Technologies, "Sending Events with RPCs," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/messaging-system/index.html.

[25] Unity Technologies, "RPC vs NetworkVariable," [Online]. Dostupno: https://docs-multiplayer.unity3d.com/netcode/current/learn/rpcvnetvar/index.html.

POPIS SLIKA

Slika 1 - Street Fighter V [4] ... 3
Slika 2 - Kreiranje projekta ... 5
Slika 3 - Novi projekt ... 5
Slika 4 – Sučelje [5] ... 6
Slika 5 - Package Manager ... 7
Slika 6 – Repozitorij ... 8
Slika 7 - GitHub Issues ... 9
Slika 8 - Teren ... 10
Slika 9 - Modeli u sceni ... 11
Slika 10 - Avatar konfiguracija ... 12
Slika 11 – Animator ... 13
Slika 12 - Fighting Animation Controller ... 14
Slika 14 - Tranzicija ... 14
Slika 14 - Sub-State Machine ... 15
Slika 15 - Slojevi animatora ... 16
Slika 17 - Namespace za pokrete ... 17
Slika 17 - Klasa Fighter ... 17
Slika 18 - Provedba udarca ... 18
Slika 20 - Alternativni udarac ... 19
Slika 20 - Blok i micanje ... 19
Slika 21 - Colliders ... 20
Slika 22 - Registriranje udarca ... 21
Slika 23 - Pozvane metode na GM ... 22
Slika 24 - Izvedba udarca u simulacij ... 22
Slika 25 - Player Controller ... 23
Slika 26 - AI Controller ... 24
Slika 27 - GM deklaracije ... 25
Slika 28 - Instantacija likova ... 26
Slika 29 - Početak borbe ... 26
Slika 30 - Sinkronizacija života i stamine ... 27
Slika 31 - Završetak borbe ... 28
Slika 32 - Nastavak borbe ... 28

Slika 33 - Pauza 29
Slika 34 - Cleanup 29
Slika 35 - Sprites 30
Slika 36 - GUI 31
Slika 38 - Sekcije u inspektoru 31
Slika 38 - Section 32
Slika 39 - Gumb 32
Slika 40 - Mijenjanje prozora 33
Slika 41 - Menu UI Handler metoda 33
Slika 42 - Animator kamere 34
Slika 43 - Izbor likova 35
Slika 44 - Implementacija izbora likova 35
Slika 45 - Vodič sekcija 36
Slika 46 - Gumbovi za naprijed - nazad 36
Slika 47 - Tranzicije u uvodu 37
Slika 48 - Veličina konteksta 38
Slika 50 - Audio objekti 39
Slika 50 - Klasa AudioManager 39
Slika 51 - Audio Mixer 40
Slika 52 - Audio slideri 41
Slika 54 - NetworkManager 42
Slika 55 - NetworkTransport 43
Slika 55 - Buffered tick [21] 43
Slika 56 - Server Authoritative [22] 44
Slika 57 - Client Authoritative [22] 44
Slika 58 - Spajanje 45
Slika 59 - Metode za spajanje 45
Slika 60 - Instanciranje likova 46
Slika 61 - NetworkVariable 46
Slika 62 - Client RPCs [24] 47
Slika 63 - Server RPCs [24] 48
Slika 64 - RPCs 48

PRILOZI ZA OBRANU RADA

Build za Windows, Intel 64-bit

https://www.dropbox.com/s/1pahbi9g8nqtpu8/CastleBrawl_v1.0.2.zip

Repozitorij za pregled koda

https://github.com/vilaleon/castle-brawl