Izrada videoigre tipa FPS

Pešić, Antonio

Undergraduate thesis / Završni rad

2023

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:963697

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-12-01**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Antonio Pešić

IZRADA VIDEOIGRE TIPA FPS

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Pešić

Matični broj: 35918/07–R

Studij: ***Primjena informacijske tehnologije u poslovanju***

IZRADA VIDEOIGRE TIPA FPS

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Mario Konecki

Varaždin, studeni 2017.

Antonio Pešić

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Korištenjem slobodnih programa poput Unity-a (eng. “Game Engine” ili hrv. “Motor igre”), Blendera (Alat za modeliranje) i Visual studia (eng. Integrated development environment ili hrv. Integrirano razvojno okruženje), omogućeno je svim korisnicima, bilo koje vrste i razine znanja, da u programiranju ili modeliranju naprave svoju videoigricu. U ovome radu nisam koristio klasične izvore informacija i citiranja, nego sam u ovome radu želio pokazati da i osobe koje nemaju pristupu akademskom obrazovanju mogu sami naučiti kako napraviti svoju igricu. Zato će većina izvora biti sa Youtube-a. Početak izgradnje igrice za svaku osobu kreće od instalacije Unity-a i gledanja par Youtube videa ili čitanja priručnika od tima Unity-a kako se uopće koristi program. Napravi se novi projekt i uđe se u glavni prikaz gdje se radi cijela igrica. Bez ikakve pomoći teško se snaći u tako opširnom programu; zato je bitno od nekuda krenuti. Također, dotaknuti ću se pitanja kako kreirati jednostavne modele unutar Blendera. Na prvu ruku on je najpoznatiji program za modeliranje, još je besplatan te svi odmah žele napraviti i svoje 3d modele. Blender je ogroman i kompleksan program koji je vrlo zahtjevan i ogroman za početnike. Najduže sam vremena proveo samo na istraživanju kako napraviti 3d model za moju igricu. Također, potrebno je neko ”Integrirano razvojno okruženje”, u mojem slučaju je to Visual Studio. Programski jezik koji se koristi je C# i preko Youtube videa moguće je sastaviti svoju videoigricu. Preko ovog rada objasnit ću kako se koristi Unity zajedno sa Blenderom i kako pisati skripte koje će Unity prepoznati za logiku u videoigrici.

Ključne riječi: Unity, Blender, Visual studio, Youtube, programski kod, scena, hijerarhija, C#, armatura.

Sadržaj

1. Uvod 1
2. Koncept igre i inspiracija 2
3. Unity 3
   3.1. Ulazak u Unity 3
   3.2. Pregled scene 4
      3.2.1. Scena 5
      3.2.2. Hijerarhija 5
      3.2.3. Projektna mapa 7
      3.2.4. Inspektor 7
      3.2.5. Game 8
      3.2.6. Asset Store i Package Manager 9
   3.3. Početak izrade igrice 11
      3.3.1. Izrada lika u igri 14
      3.3.2. Izrada skripte za pomicanje i gledanje lika u igri 14
      3.3.3. Izgradnja levela 17
      3.3.4. Interaktivnost i animacije 19
      3.3.5. Neprijatelji 23
      3.3.6. NavMesh 24
      3.3.7. Oružje 25
      3.3.8. Život igrača 27
      3.3.9. SpawnPoint i prolaženje igrice 28
      3.3.10. Neprijateljski šef 29
      3.3.11. Audio 29
      3.3.12. 2D scene 30
      3.3.13. Mijenjanje scene 31
4. Blender 31
   4.1. Početak modeliranja 32
      4.1.1. Alatna traka 33
      4.1.2. Hijerarhijski prozor i inspektor 34
         4.1.2.1. Hijerarhijski prozor 34
         4.1.2.2. Inspektor 35
   4.2. Prvi model 35

4.3. Dodavanje kostiju ... 36
4.4. Stavljanje animacije ... 39
4.5. Izvoz lika u Unity ... 40
5. Skripte i kodovi ... 43
5.1. Skripte za igrača ... 43
5.1.1. InputManager ... 43
5.1.2. PlayerMotor ... 44
5.1.3. PlayerHealth ... 46
5.1.4. PlayerLook ... 49
5.2. Skripte za interakciju ... 51
5.2.1. Interactable ... 51
5.2.2. PlayerInteract ... 52
5.2.3. Keypad ... 53
5.3. Neprijatelji ... 54
5.3.1. Enemy ... 54
5.3.2. EnemyAI ... 56
5.3.2.1. StateMachine ... 56
5.3.2.2. SearchState ... 57
5.3.2.3. PatrolState ... 58
5.3.2.4. BaseState ... 59
5.3.2.5. AttackState ... 60
5.3.3. Target ... 61
5.3.4. EnemyForRunning ... 62
5.3.5. Path ... 63
5.4. Oružje ... 65
5.4.1. GunData ... 65
5.4.2. Gun ... 65
5.4.3. PickupWeapon ... 67
5.4.4. WeaponState ... 68
5.4.5. PlayerShoot ... 69
5.5. Spawner ... 70
5.5.1. EnemyForWaves ... 70
5.5.2. WaveSpawner ... 71
5.5.3. EnemySpawnerEasier ... 74
5.6. Neprijateljski šef ... 75
5.6.1. EnemyHeart ... 75
5.6.2. ParentOfTheEnemy ... 76

5.7. 2D skripte 76
5.7.1. StartMenu 76
5.7.2. EndMenu 77
6. Zaključak 78
7. Popis literature 79
8. Popis slika 80
9. Prilozi 82

1. Uvod

Tema je izabrana zbog ljubavi prema FPS igrama. U ovome radu naučit će se kako se radi sa “Unity Game Engine” i “Blender” programima. Unity game engine je na svakoj preporuci kada se napiše na internetu: “How to learn to make a 3D game”. Zbog tog sam izabrao Unity, i zbog tog što sam ranije već radio neke sitne projekte unutar tog programa. Iako sam upoznat već sa programiranjem, prebacivanje na drugi jezik s kojima nisam toliko upoznat će biti veliki izazov. Prebacivanje iz Front End Developmenta u izradnju igrice me zanima kako ću uspjet napraviti ovaj rad sa već nekim predznanjem u programiranju. Cilj ovog rada je napraviti početničku igricu koja ima početak i kraj. Također je cilj ovog rada da se nauči neka nova osoba kako se izrađuje igrica kroz sve korake. Na kraju kada se završi rad, dat ću nekome da ga pročita i kaže svoje mišljenje, da li je razumio poantu i da li bi on uz ovaj rad mogao početi raditi na svojoj igrici.

Ovaj rad će objasniti najbitnije dijelove Unity-a za početnike, kako napraviti početnu scenu, kako napraviti prvi projekt i većinu opcija koje su ponuđene. Što su objekti, skripte, što su karakteristike objekta, sve će se objasniti unutar ovog rada.

Također će se objasniti Blender aplikacija. Ne toliko detaljno jer je program jako širok i prepun je opcija koje nisu napravljene samo za razvoj igrica. Pogledao sam neke video prije same izgradnje videoigre da dobijem osjećaj da sada radim na igrici a ne na web stranici.

2. Koncept igre i inspiracija

Odabrao sam old-school ili 'Boomer shooter' tip FPS igara zbog moje strasti prema tom žanru. Još od svojih mlađih, dana pa sve do danas, slijedim industriju i pratim najnovije izlaze u žanru 'boomer shooter'. 'Boomer shooter', prema izmišljenoj definiciji, predstavlja svaki FPS koji nasljeđuje stil prvih FPS igara poput 'Doom', 'Wolfenstein 3D' ili 'Quake'. Karakteristike ovih igara u njihovo vrijeme bile su 3D grafika, brza kretanja igrača i projektilno oružje koje se koristi za nanošenje štete neprijateljima/igračima. Tijekom godina, kako se razvijao softverski i hardverski svijet, razvijale su se i igre ovog tipa.

Primjerice, igre kao 'Serious Sam', 'Gore', 'Cube' i mnoge druge, ostale su vjerne konceptu hodanja kroz linearni svijet i borbe u arenama, nakon čega slijedi ulazak u novu arenu. Posljednjih je nekoliko godina indie gaming scena procvjetala, a velik broj 'Boomer shooter' igara pojavio se na sceni. Igre poput 'Hyperviolent', 'Selaco', 'Prodeus' i mnoge druge igre slijede isti stil. One zadržavaju karakteristike starijih igara, poput linearnog dizajna i borbi u arenama, ali su obogaćene suvremenim značajkama. Animacije, kvaliteta grafike i mogućnosti su danas puno veće nego ikad prije, što omogućava bilo kome da razvije svoj vlastiti projekt.

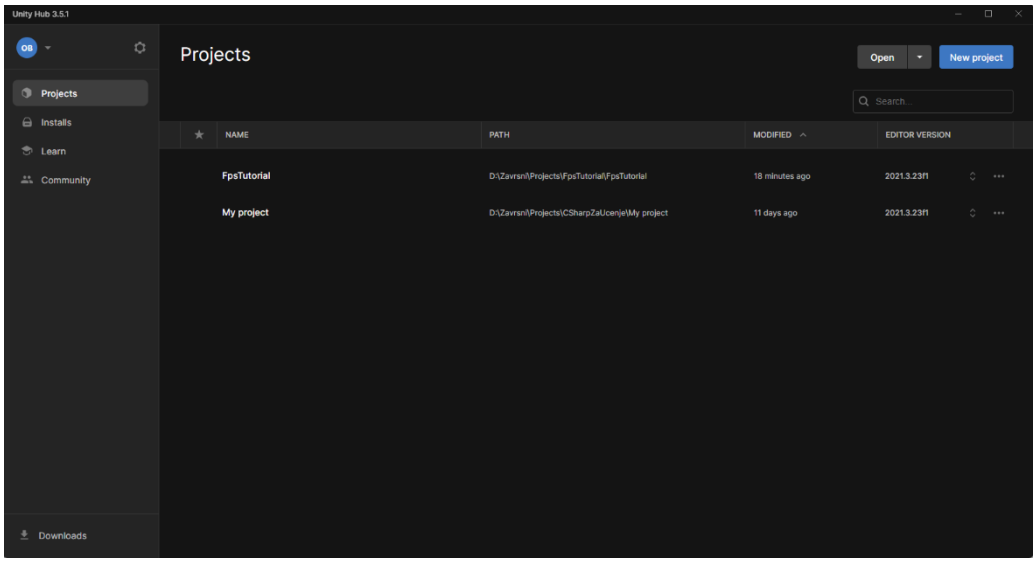
U ovom radu ćemo demonstrirati kako bilo koja osoba, bez obzira na prethodno iskustvo u programiranju ili izradi videoigara, može stvoriti jednostavan old-school FPS naslov koristeći moderne alate koji nisu bili dostupni u vreme kada su takve igre prvobitno nastajale. Igrica se sastoji od tri levela koji započinju u istoj sobi gdje se može odabrati jedno od oružja ponuđenih, a cilj je preživjeti “Arenu” u kojoj se generiraju neprijatelji u valovima. Igrica se sastoji od tri levela, dok se u zadnjem levelu nalazi neprijateljski šef (eng. “Boss enemy”). Kako je napisano prije, inspiracija je došla od ranijih igara, ali također je ubačeno još par ideja koje su preuzete od nekih novih igara. Jedna od novijih ideja koje su zaživjele je “Dark souls” tip igre, značenje da je igra ultra teška, ali poštena. Zato je u ovaj projekt ubačeno da igrač umire na dva pogotka metka od neprijatelja ili instant ako se previše približi na lokaciju od igrača. Ovako se postavlja neki stupanj težine, dok je još uvijek moguće sa pokretima igraća izbjeći prijetnje koje se stvaraju.

3. Unity

Za izradu video igre odabran je Unity game engine, to je besplatan alat koji se može skinti na https://unity.com/download. Nakon instalacije može se krenuti sa izradom video igre. U ovom radu se koristi verzija 2021.3.23f1 Unity-a.

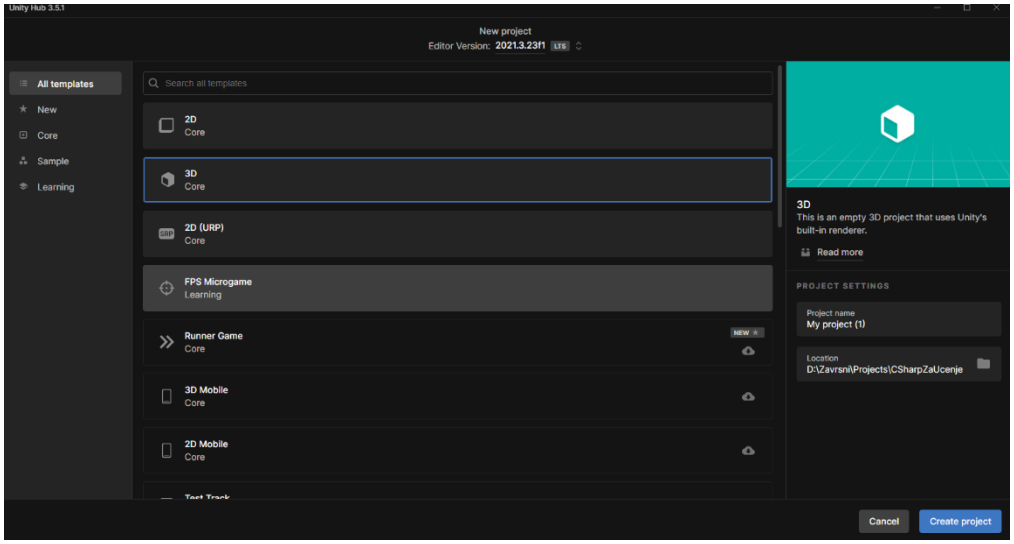
1.1. Ulazak u Unity

Paljenjem Unity-a ne ulazi se odmah u scenu i ekran za izradu videoigre. Prvi korak je izrada projekta preko “Unity Hub-a" (u ovom radu koristi se verzija 3.5.1). Unutar tog prozora moguća je prijava, izrada novog projekta, biranje projekta (ako postoje). Također, moguće je birati verzije Unity-a, mogućnost pronalaženja lekcija za učenje korištenja Unity-a ili videoigara, dio za korisnike Unity-a i pregled trenutnih aktivnih preuzimanja datoteka.



Slika 1 Unity Hub (Izvor: Unity Hub, 2023.)

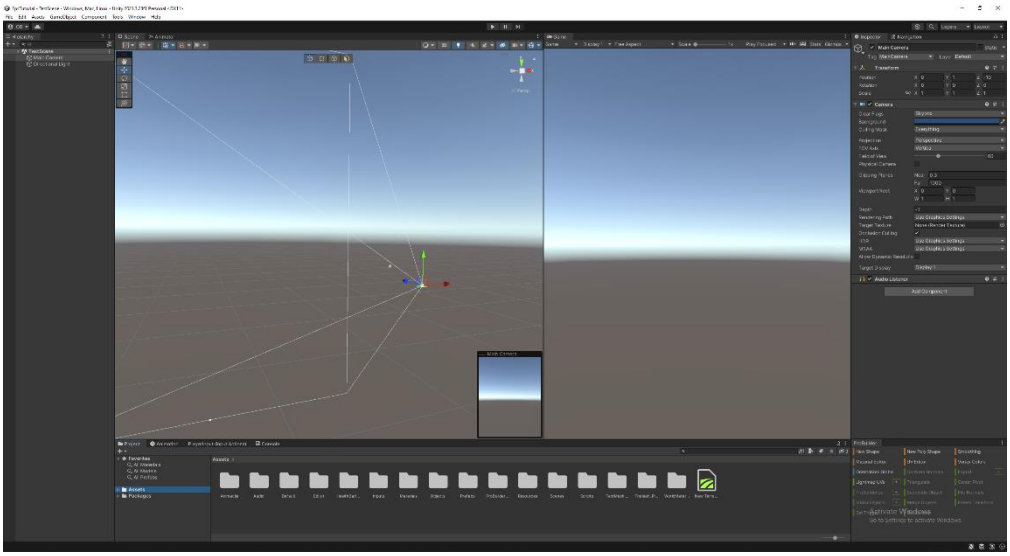
Za ovaj projekt je potrebno kliknuti na “New Project”, odabrati 3D Core unutar “templates”. S desne strane nalaze se projektne postavke te se unutar postavka postavlja ime projekta i lokacija gdje se želi pohraniti projekt. Nakon odabira se klikne na “Create project” i zatim se projekt izgrađuje.



Slika 2 Kreiranje projekta (Izvor: Unity Hub, 2023.)

2.1. Pregled scene

Nakon Izgradnje aplikacije uključit će se novi prozor koji se koristi za pregled scene. Tu se izrađuje videoigra.



Slika 3 Izgled početne scene (Izvor: Unity scene editor, 2023.)

Aplikacija za scenu izgrađena je od nekoliko dijelova koji su potrebni za izradu videoigre. Na prvu ruku ovo može izgledati nepojmljivo i komplicirano, ali uz dobro objašnjenje jednostavno je započeti i razumjeti što se događa. U sljedećim će se dijelovima objasniti sve komponente unutar aplikacije.

1.1.1. Scena

Prvo što je vidljivo u aplikaciji je veliki prozor u sredini na kojem s lijeve gornje strane piše “Scene”. U tom se prozoru postavljaju svi objekti za koje želimo da budu unutar igrice. Kako je odabran 3D projekt, u sceni je prikazan trodimenzionalni prostor sa x, y i z koordinatama. Početna pozicija svakog projekta, nije bitno je li 2d ili 3d, uvijek će biti x=0, y=0 i z =0. Prvi objekti koji će se nalaziti unutar projekta su “Main Camera” (hrv. Glavna kamera) i “Directional Light”. “Main Camera” služi za pogled unutar same igrice (o tome kasnije), dok Directional Light služi za svijetlost i od kuda dolazi (možemo reći da je kao sunce). Sa lijeve Scene, postoji alatni pojas sa različitim opcijama preko kojih se obavljaju različite stvari unutar scene. Na sljedećoj slici će biti objašnjeno značenje alata.

Slika 4 Tools (Alati) (Izvor: Unity “Scene” prozor, 2023.)

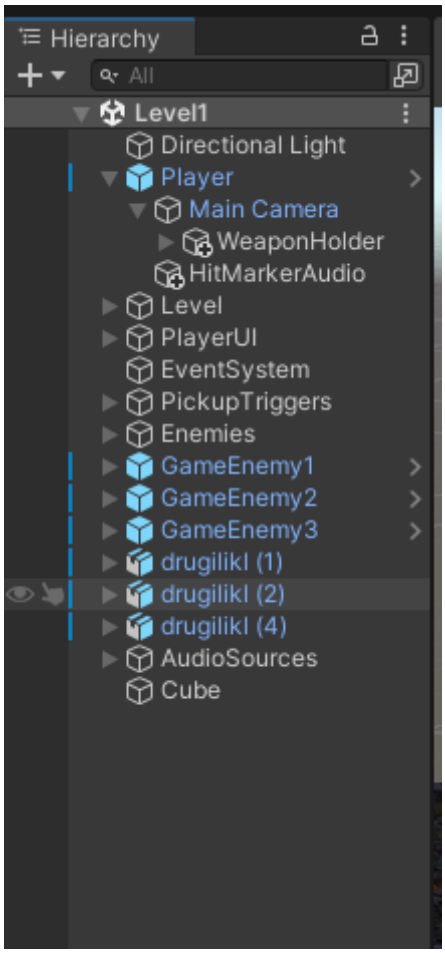
Alatni pojas sastoji se od:

- View Tool - Omogućuje pomicanje po 2D prostoru, pomicanje našeg pogleda na scenu.
- Move tool - Omogućuje pomicanje objekata unutar scene.
- Rotate tool - Omogućuje rotiranje objekata unutar scene (0-360 stupnjeva).
- Scale tool - Omogućuje skaliranje objekta unutar scene.
- Rect tool - Slično kao scale tool, ali povećava objekt prema obratnim kutevima, a ne prema sredini objekta.

Sa gornje desne strane moguće je vidjeti objekt za orijentaciju na kojoj možemo stisnuti bilo koju koordinatu koja nas potom prebaci na tu poziciju gledanja.

1.1.2. Hijerarhija

Prvi prozor s lijeve strane od scenskog pogleda je “Hierarchy”. Unutar tog prozora nalaze se svi objekti koji se koriste unutar videoigre.



Slika 5 Hijerarhija (Izvor: Unity “Scene”, 2023.)

Unutar hijerarhije na slici može se vidjeti da objekt može imati dijete objekt ili roditelj objekt. Djeca od jednog objekta mogu se zajedno pomicati unutar scene ako je odabran njihov roditelj. Također ako se kopira jedan objekt sa djecom, sva djeca će se kopirati zajedno sa roditeljem. Na vrhu (“Level1”) hijerarhijskog prozora nalazi se ime trenutne scene. Također, važno je napomenuti da ako su objekti kreirani i uvezeni iz drugih izvora, kada ih ubacite u hijerarhiju, bit će označeni plavom bojom. Klikom na strelicu s desne strane možete fokusirati kameru samo na taj objekt unutar scene, omogućavajući vam da se detaljnije posvetite njegovoj manipulaciji.

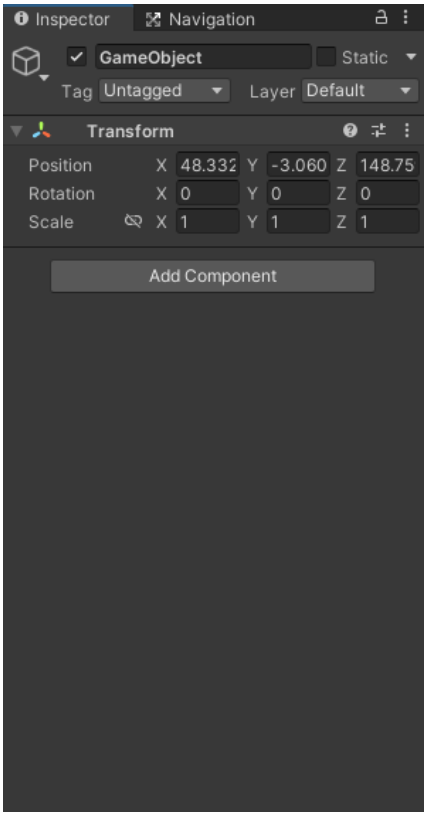
1.1.3. Projektna mapa

Projektna mapa nalazi se na dnu aplikacije i tu se može nalaziti bilo što osoba koja radi igricu postavi. To mogu biti “prefabs” (hrv. Montaže) (gotovi objekti koji su ponovno iskoristivi), skripte, animacije, kontroleri, objekti, itd. Svaki put kada se iz toga dijela ubacuje nešto unutar hijerarhije, ono stvara kopiju tog gotovog objekta.

Slika 6 Projektna mapa (Izvor: Unity “Scene”, 2023.)

1.1.3. Inspektor

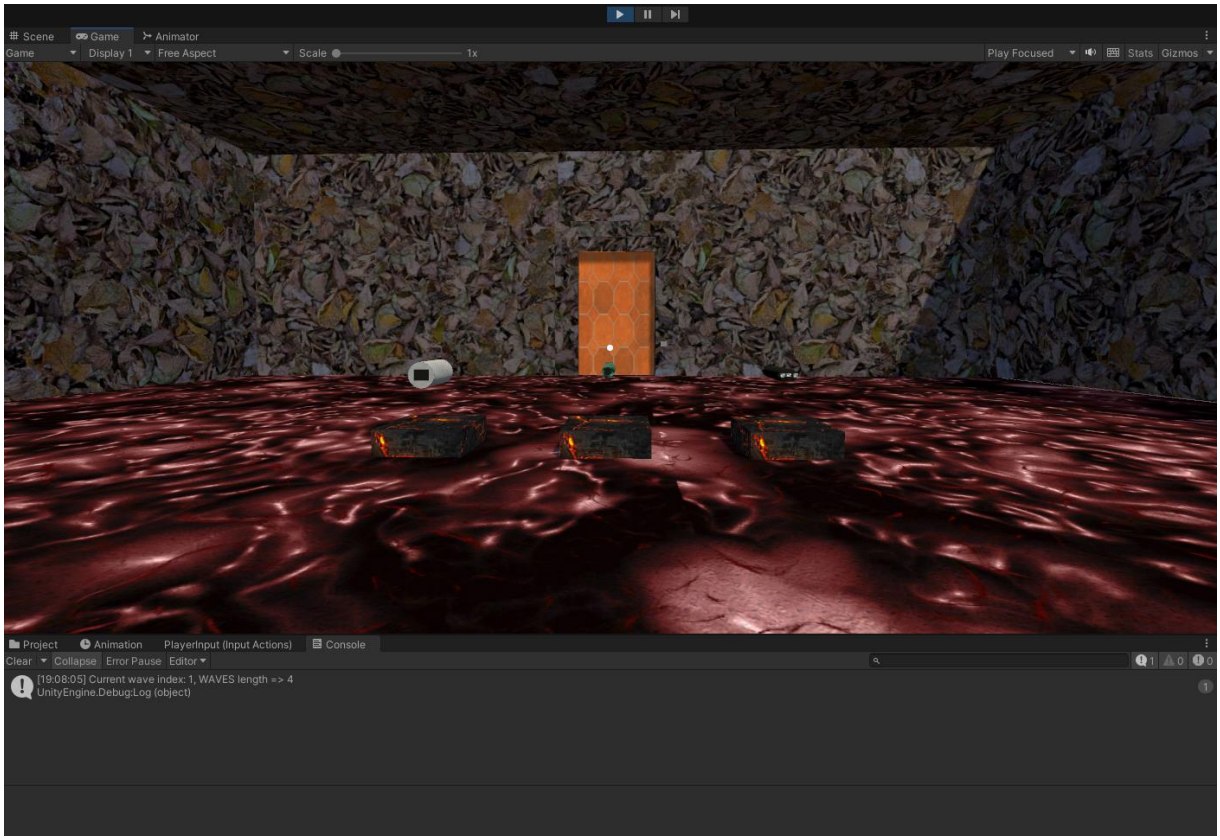
Inspektor se nalazi s desne strane ulaska u Unity i u njemu pišu sve osobine odabranog objekta.



Slika 7 Inspektor (Izvor: Unity “Scene”, 2023.)

Na slici devet može se vidjeti inspektor dok je odabran novi prazni objekt. Na vrhu unutar inspektora uvijek se vidi je li objekt uključen (Kvačica pored imena), ime objekta, je li statičan ili ne (ako je statičan to znači da se objekt za vrijeme pokrenute igre ne miče), tag (preko njega možemo u skriptama hvatat određene objekte i davat im posebne osobine), i zadnje je “Layer” i on služi za razdvajanje objekata unutar igre. Unutar svakog odabranog objekta, objekt će uvijek imati “Transform” svojstvo. Ono određuje poziciju, rotaciju i skalabilnost objekta u sceni. Ispod svih svojstva (ako ih je više) nalazi se “Add Component” gumb, on služi za dodavanje novih komponenti ili skripti na odabrani objekt.

1.1.4. Game

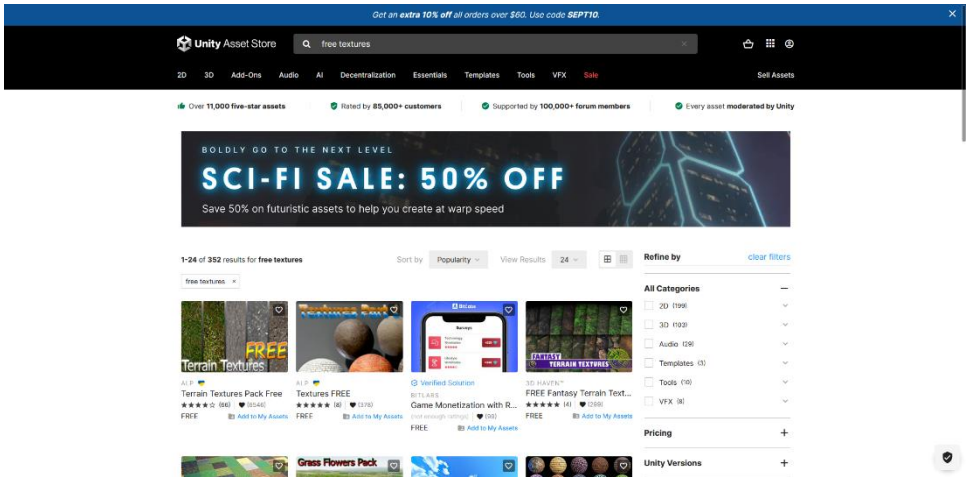


Slika 8 Game prozor (Izvor: Unity ”Scene”, 2023.)

Na slici osam vidi se “Game” prozor u kojemu su svi elementi do sad stavljeni u scenu i koji će biti vidljivi kada se započne izgrađena igrica. "Game" prozor se automatski prebacuje kad na vrhu iznad scene se pritisnete "Play button" ili najlijeviji gumb od tri dostupna gumba. U sredini se nalazi gumb za pauziranje aktivne igre, dok se desno nalazi gumb za preskakanje scena. Dok igra traje preporučeno je držanje konzole na dnu prikaza da se mogu gledati logičke greške i njihov mogući ispravak.

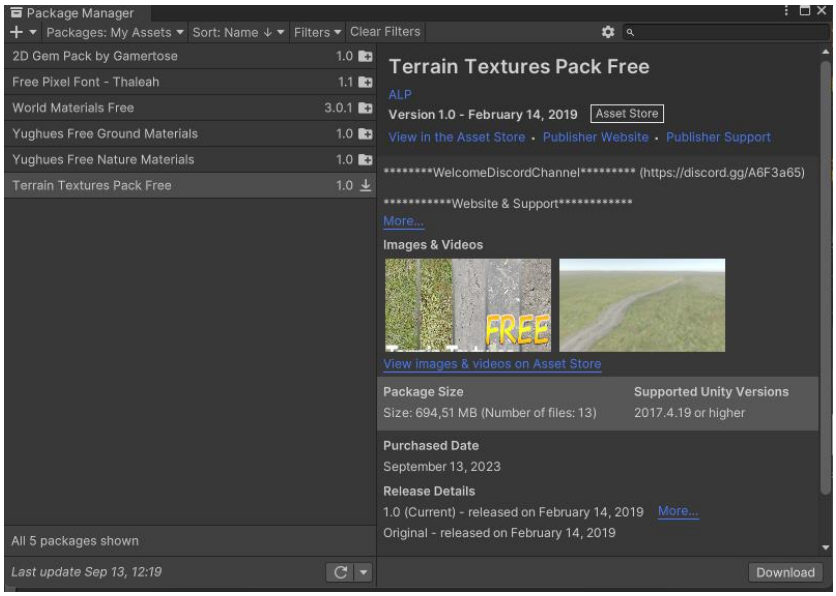
1.1.5. Asset Store i Package Manager

“Asset store” (hrv. Trgovina resursima) je web stranica na web pregledniku (prije joj je bilo moguće pristupiti preko aplikacije, ali to su promijenili u novijim verzijama Unity-a), preko koje je moguće preuzeti razne “Assets” (hrv. Resurse). To mogu biti 3D modeli, 2D modeli, teksture, skripte, itd. Većina resursa koji su postavljeni na web stranicu su izrađeni su od korisnika aplikacije. Na stranici je moguće kupiti resurse, no, postoje i besplatni resursi. U ovom ćemo radu koristiti samo besplatne resurse.



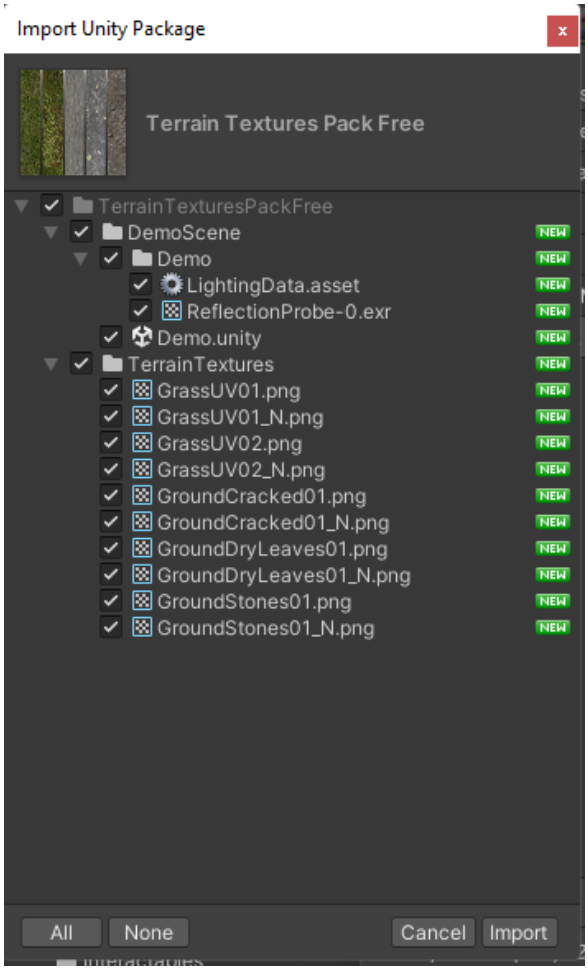
Slika 9 Unity Asset store (Izvor: https://assetstore.unity.com, 2023.)

Nakon što se pronašao neki resurs koji se želi preuzeti, na stranici se pritisne “Add to My Assets”, nakon čega se na “Terms of Service and EULA“ pristisne “Accept”. Na vrhu ekrana pisati će da se dodao resurs u naše “Assets-e”, s desne strane su dva gumba na kojima piše “Open in Unity” i “Go to My Assets”. Pritiskom na “Open in Unity”, vraćamo se u Unity aplikaciju gdje se otvara “Package Manager”.



Slika 10 Package Manager (Izvor: Unity aplikacija, 2023.)

Na slici deset prikazan je “Package Manager”, koji se koristi za upravljanje preuzetih vanjskih resursa. Također je važno primijetiti da se sada nalazi resurs koji smo preuzeli sa web stranice, i prva stvar koju je potrebno učiniti jest kliknuti na gumb "Download" (hrv. Preuzeti). Nakon toga, potrebno je pričekati da se resurs preuzme. Kada se preuzme pritisne se gumb “Import” i otvori se novi prozor sa novim opcijama.



Slika 11 Import Unity Package (Izvor: Unity aplikacija, 2023.)

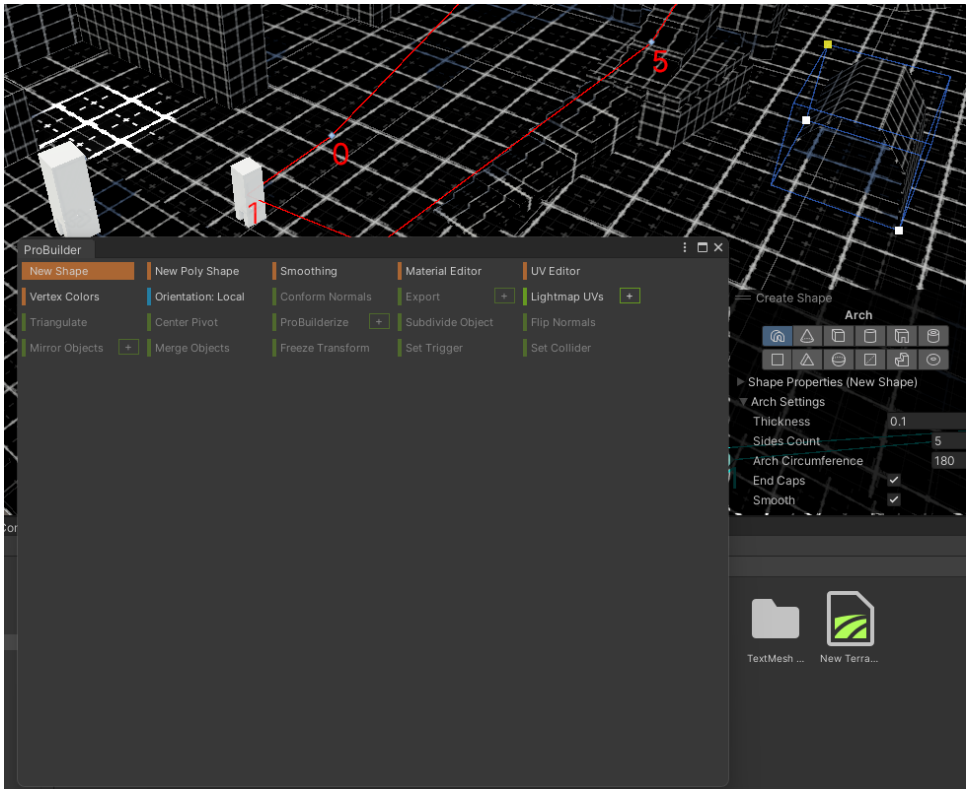
Na slici jedanaest, vidljiv je odabir svih resursa unutar preuzetog paketa. Ako se želi, neki se resursi mogu maknuti iz preuzetog paketa tako da maknemo kvačicu pored imena resursa. Nakon što smo napravili odabir, klinke se na gumb “Import”. Paket koji smo preuzeli na kraju je vidljiv na alatnog traci projekta. Važno je naglasiti da se “Packet Manager-u" može pristupiti preko Unity aplikacije na gumb “Window”, zatim na klik “Package Manager”.

3.1. Početak izrade igrice

Prije samog početka stvaranja videoigrice, potrebno je preuzeti par paketa iz “Package Managera”. To su (Izvor: Unity package manager, 2023):

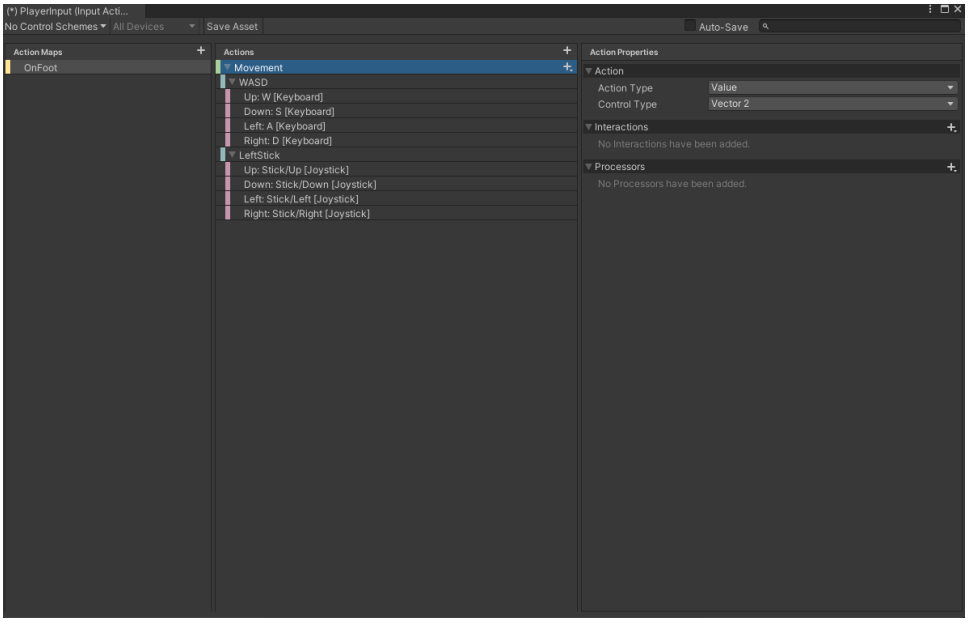
- ProBuilder: Služi za stvaranje, uređivanje i teksturiranje prilagođene geometrije unutar Unity-a. Koristi se za jednostavno stvaranje kompleksnijih oblika unutar videoigre. Na slici dvanaest vidljivi su ProBuilder prozor i s desne strane opcije koje su moguće kada se odabere nešto sa Probuildera. U ovome radu objasnit će se samo “New Shape” (hrv. Novi oblik). Kada je “New Shape” odabran, otvori se prozor na kojemu piše

“Create Shape” (hrv. Stvori oblik). Odabirom na željeni oblik, dobivaju se njegova svojstva koje je moguće promijeniti po želji. U Scenskom se pogledu nakon željenih odabira može napraviti oblik. Zavisno o obliku, prvo se stisne na jednu točku, zatim se držanjem miša povuče po prostoru za željenu veličinu. Ako postoji treća koordinata i ona se povuče. Novonapravljeni se objekt potom prikaže u Hijerarhijskom prozoru i zatim se s njime može raditi po želji.



Slika 12 Probuilder prozor i kreator (Izvor: Unity aplikacija, 2023)

- Input System: Prema opisu, input system novi je način za stvaranje kontrola za igrača i koristi se kao alternativa za klasični ulazni sustav. On se stvara unutar projektne mape sa pritiskom na desni klik, biranja na “Create” i ponovnim klikom na “Input Actions”. Nakon toga stvara se Input Action Asset. Klikom na njega otvara se novi prozor u kojem stvaramo nove komande. Vidljivo na slici trinaest.



Slika 13 Input Action prozor (Izvor: Unity aplikacija, 2023.)

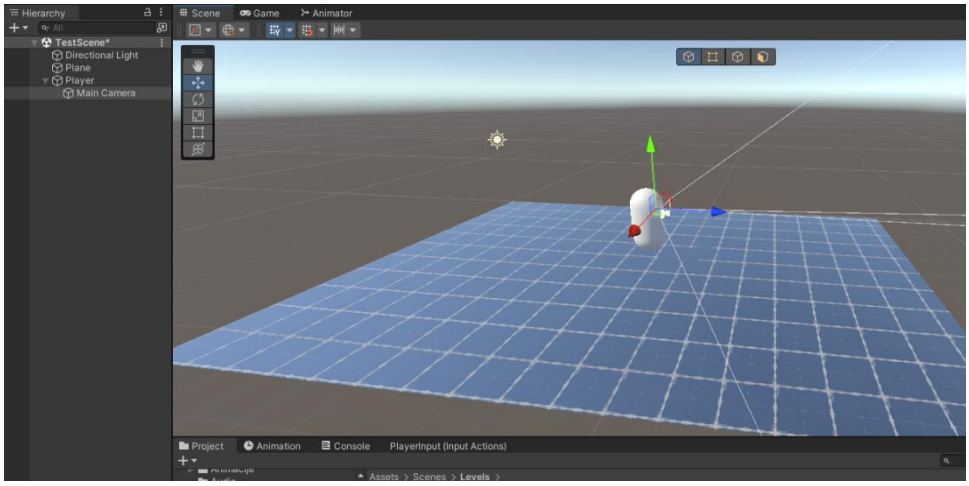
Sa lijeve strane je ime metode koja će se koristiti i koja se dodaje preko plusa na vrhu kolone. U sredini se dodavaju akcije koje se želimo izvoditi, desnim klikom na svaku akciju stavljamo “binding” (hrv. Poveznica) preko koje se dodaju željene radnje. S desne strane nalaze se “Binding properties” (hrv. Svojstva poveznice) preko kojih određujemo kako će se radnje ponašati. Nakon što su se stavile sve poželjne opcije, klikne se na “Save Asset”. Kada smo spremili ulazne akcije vratimo se u mapu gdje se nalazi “Input asset“ datoteka. Zatim u inspektoru možemo vidjeti da postoji “checkbox” (hrv. Potvrdni okvir). Kada ga potvrdimo, možemo napraviti C# klasu u kojoj preko programskog koda određujemo radnje unutar igre.

- TextMeshPro: Koristi se kao zamjena za Unity Ui Text i Text Mesh. Jednostavno se koristi unutar koda i ima velik izbor mijenjanja fontova, stila, veličine, itd.

Nakon instalacije svega potrebnog može se krenuti. Prva stvar koju želimo napraviti je lik za igrača igre.

4.1. Izrada lika u igri

Započinjemo razvoj igre tako da u sceni preko ProBuildera (otvaramo prozor preko “Tools” -> “ProBuilder” -> “ProBuilder Window”), odaberemo “New shape” i u scenu postavimo neki objekt na kojemu želimo testirati početne skripte i promijene. Nakon što smo postavili objekt kao pod, možemo ubaciti svog lika. U ovome radu koristit ćemo “Capsule” 3D objekt (desni klik u hiearchy -> create 3d object -> capsule). Nakon što se dodala kapsula, ili neki drugi objekt za igračevog lika u Hijerarhiji, kliknut će se desnim klikom na njega i promijenit ime u “Player”. Također, u hijerarhiji treba postaviti “Main Camera” kao dijete objekta od igračevog lika. To se može jednostavno učiniti sa “drag and drop” kamere ispod igračevog lika. Kada se ubacila kamera, treba namjestiti kameru da kreće iz primjerenog mjesta. Kako je ovo FPS igrica, odabere se kamera i zatim se pozicija kamere namješta tako da je ona na poziciji igračeve glave.



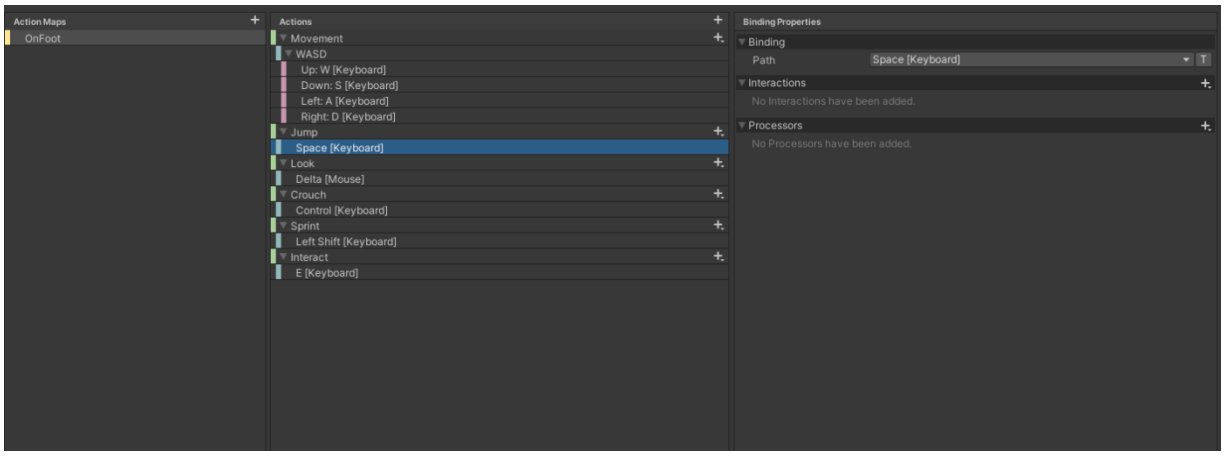
Slika 14 Namještanje kamere (Izvor: Unity aplikacija, 2023.)

Sada kada se pomiče “Player” objekt, sva njegova djeca pomiču se s njime. Kada se stisne gumb play, vidjet će se iz perspektive kamere koja je sad dijete kapsule.

1.1.6. Izrada skripte za pomicanje i gledanje lika u igri

Kada se kreira skripta, pametno ih je razvrstati po njihovoj upotrebi, što znači, ako se radi o skriptama, pametno je napraviti mapu imenom “Scripts” i u taj folder ubacivat samo skripte. Mogu se napraviti mape prema njihovom radu. Ako se napravila mapa za neprijatelja onda sve što je vezano za neprijatelje će se spremati u tu mapu. U ovom će se radu koristiti prva metoda. Također nije bitan naziv tih mapa, nego će se u ovom radu samo dati primjer kako je osoba koja je napisala ovaj rad postavila imena.

U novu mapu, pod imenom “Inputs”, kreira se novi Input Action (desni klik u mapi -> create -> input actions). Kada se stvorio novi “Asset”, postavlja se njegovo ime (Najčešće ime je povezano sa radnjom te skripte/radnje). U ovom će se radu zvati “PlayerInput”. Duplim klikom odabrati novo napravljeni “Asset” i otvara se prozor od “Input Actions-a”. U actions map stupcu postavit će se nova akcijska mapa pod imenom “OnFoot”, a zatim se pod Akcije postavljaju sve poželjne radnje. Na sljedećoj slici biti će pokazane sve radnje koje će biti uključene za igrača u ovome radu.



Slika 15 Input actions za igrača (Izvor: Unity aplikacija, Input Actions, 2023.)

Svaka se akcija postavlja pomoću plusa na “Actions” stupcu. Ovo će biti sve radnje koje će igrač moći raditi. Objašnjenje slike i postupka izrade #1 FPS Movement: Let’s Make a First Person Game in Unity!, Natty GameDev, 8. 12. 2023.:

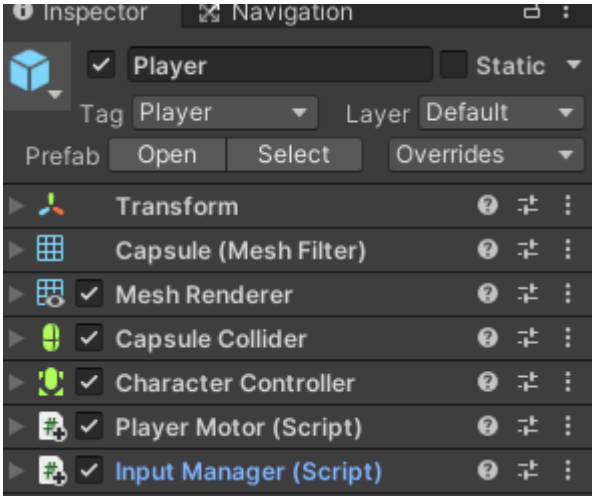
- Movement: Služi za pomicanje igrača. U svojstvima pod “Action Type” postavlja se na Value (jedinstvena vrijednost, najbolje za integer ili float vrijednosti), a “Control Type” na Vector2 (x i y vrijednosti, što znači da će se po dvije dimenzije moći kretati). Stisne se plus na movement i odabere “Add Up/Down/Left/Right Composite”. U programu će se stvoriti struktura kao na slici. Prvi element postaviti će se na “WASD” i u svojstvima “Composite type” postavi se na “2D Vector”, dok “Mode” na “Digital Normalized”. Ispod WASD su četiri akcije u kojima će se neko slovo na tipkovnici postaviti na “Path”, dok se na “Composite Part” postavi vrijednost za to slovo, to jest u kojem smjeru da se igrač kreće. U ovom slučaju je to klasični WASD kontrolna shema gdje je W-gore, A-lijevo, D-Desno, S-Dolje. Bitno je također da kada se bira slovo da piše pored slova [Keyboard].
- Jump: Služi za skakanje i “Action Type” se postavlja na “Button”, klikne se plus i stisne se na “Add Binding”, unutar veze na “Path” se postavi “Space [Keyboard]”.

- Look: Služi za pogled, i “Action Type” se postavlja na “Value” i “Control Type” na “Vector2”, klikne se plus i stisne se na “Add Binding”, unutar veze na Path se postavi “Delta [Mouse]”.
- Crouch: Služi za čučanj, i “Action Type” se postavlja na “Button”, klikne se plus i stisne se na “Add Binding”, unutar veze na “Path” se postavi “Control [Keyboard]”.
- Sprint: Služi za šprintanje, i “Action Type” se postavlja na “Button”, klikne se plus i stisne se na “Add Binding”, unutar veze na “Path” se postavi “Left Shift [Keyboard]”.
- Interact: Služi za interakciju sa interaktabilnim elementima u igri, i “Action Type” se postavlja na “Button”, klikne se plus i stisne se na “Add Binding”, unutar veze na “Path” se postavi “E [Keyboard]”.

Nakon što su se unijele sve željene vrijednosti klikne se na “Save Asset” i stvori se novi Asset unutar mape. Nakon toga pritisnemo na promijenjeni Asset i u inspektor pritisnemo kućicu da želimo generirati C# klasu. U prvom redu odaberemo lokaciju gdje želimo spremiti skriptu, u sljedećem redu odaberemo njeno ime i zadnji red nije bitno upisati. Na kraju se klikne “Apply” i dobit će se nova skripta.

Da se mogla koristiti nova napravljena klasa, treba napraviti novu skriptu koja će moći koristiti novo napravljene klase. Za to će trebati napisati nove skripte po nazivom “InputManager” koja će služiti kao centralna točka za prolaženje svih stisnutih inputa. Kod za ovu skriptu se nalazi pod naslovom “5.5.1 InputManager”. Sljedeća skripta koja je potrebna je skripta za micanje. U istu mapu može se napraviti nova C# skripta pod nazivom “PlayerMotor”, u kojoj se nalazi sva potrebna logika da se igrač može pomicat. Ona se spaja sa InputManager skriptom i očekuje pritisnute tipke kako bi izvršila određenu akciju. Kod za ovu skriptu nalazi se pod naslovom “5.1.2 PlayerMotor”.

Nakon što su se napravile skripte, treba ih ubaciti na “Player” objekt da bi zapravo radile. To se može napraviti na dva načina. Prvi je da se stisne na “Player” objekt u hijerarhiji i mišem se odradi “drag and drop” odabranih skripti u inspektor tog objekta, ali na neki prazni dio. Drugi je način da se na odabrani objekt u inspektoru stisne “Add Component” i upiše se ime skripte koja se želi dodati. Kada “Player” objekt izgleda ovako:



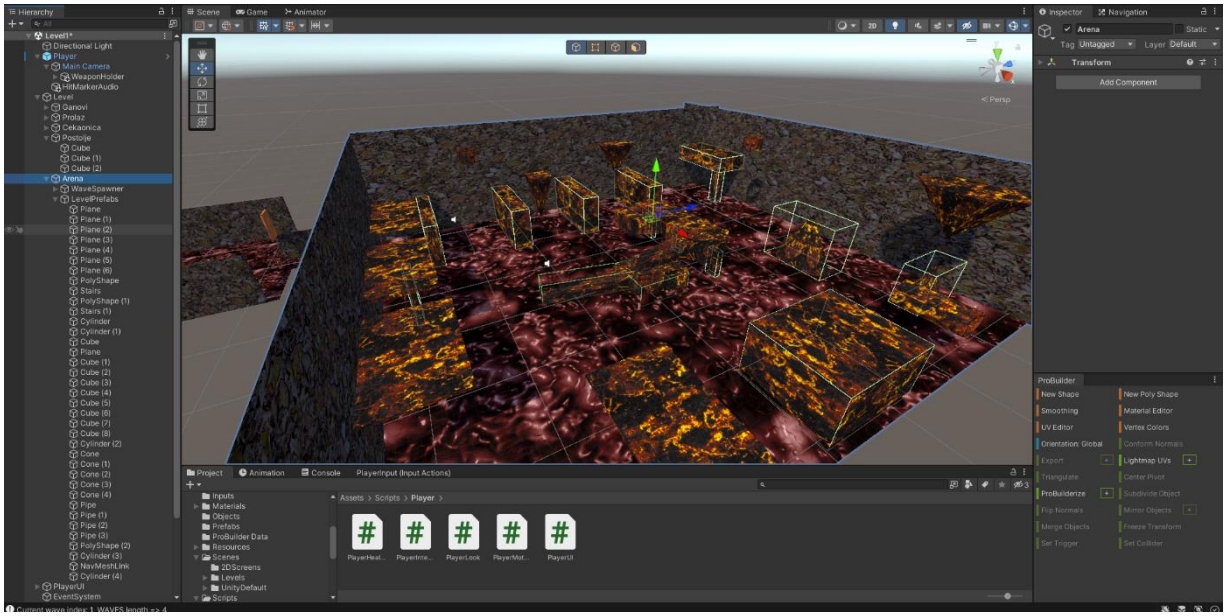
Slika 16 Izgled objekta sa ubačenim skriptama (Izvor: Unity aplikacija, Inspector, 2023.)

Također je potrebno dodati “Character Controller” koji govori neke posebne informacije o objektu na kojem je postavljen. Također, moguće su promjene nad tim objektom koji će mijenjati ponašanje kako se objekt pomiče. Kada je sve postavljeno može se pokrenuti scena i isprobati. Objekt bi se trebao micati pomoću tipki W,A,S,D. Također, vrijedno je napomenuti da sve skripte koje u sebi imaju “[SerializeField]” ili deklaracije varijabli kao “public” omogućavaju da se takve varijable vide unutar inspektora i moguće je promijeniti njihove vrijednosti po potrebi.

Sljedeće što je bitno je napraviti skriptu za pomicanje pogleda unutar igre. Opet se radi nova skripta koja će biti postavljena u inspektoru pod objektom Player i nazvat će se “PlayerLook”. Kod za tu skriptu nalazi se pod naslovom “5.1.4. PlayerLook”. Nakon što je kod napisan, u inspektoru pod varijablu “Cam” potrebno je ubaciti neki objekt. U ovom slučaju to je “Main Camera” od našeg “Player” objekta, zato što “Main Camera” u igrici služi za gledanje u svijet, a poanta skripte je da se pomiče pogled u igri. Stoga treba samo“drag and drop” Main Cameru u to polje, i moguće je namjestiti x i y senzitivnost za brzinu miša unutar igre. Nakon što se to odradilo, pritišće se “Play” i provjerava se može li se okretati kamera pomoću miša. Sada bi trebali raditi pokreti i gledanje igračevog lika. Sljedeće na čemu će se raditi je level.

1.1.7. Izgradnja levela

Levele je jednostavno izgraditi jednom kada se shvati kako se objekti grade, a uz pomoć Probuildera vrlo je jednostavno. U hijerarhiji je najpametnije raditi strukture gdje će, naprimjer: roditelj biti soba i sve komponente unutar sobe su njegova djeca. Tako napravljena hijerarhija omogućuje ponovno korištenje elemenata.



Slika 17 Gotova arena (Izvor: Unity aplikacija, 2023.)

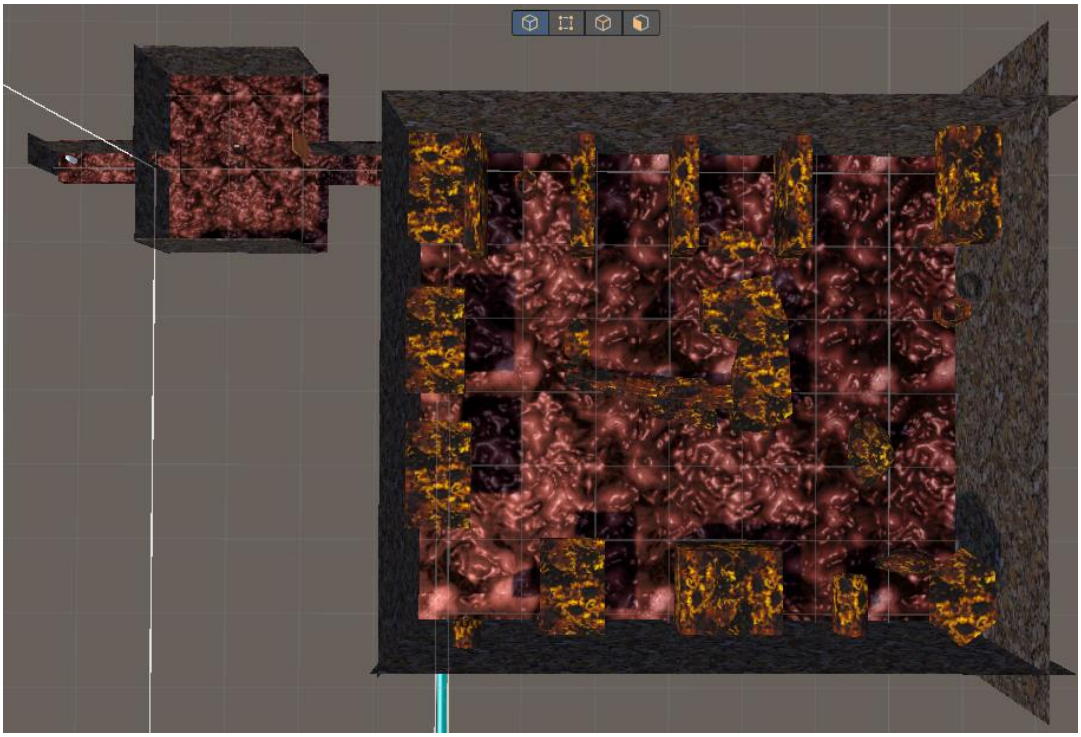
Na slici sedamnaest vidljiv je jedan cijeli izgrađeni level. Na hijerarhijskog je strani odabrana Arena objekt koja je roditelj mnogo pojedinačnih objekata. Klikom na roditelja označiti će se svi objekti koji su djeca tog roditelja. To je vrlo korisno ako se želi kopirati veliki dio objekata odjednom.

Za ovaj je rad odabrano da se igrica sastoji od tri lokacije u jednom levelu.

1. Spawn Point: Mjesto koje služi kao početak svakog levela i od kuda kreće igra.
2. Čekaonica: Mjesto gdje se bira oružje koje će se koristiti unutar arene.
3. Arena: Mjesto gdje se odvija glavni dio igre i koji se mora preživjeti da se može proći na sljedeći level.

Ako se želi spremiti neki dio levela za ponovno korištenje, moguće je povući iz hijerarhije cijeli dio koji je poželjan i prebaciti ga u neku mapu po želji. Nakon toga je moguće cijeli taj dio ponovno iskoristiti u bilo kojoj sceni.

Cijeli dizajn levela “iz zraka” moguće je vidjeti na slici osamnaest.



Slika 18 Nivo s gornje strane (Izvor: Unity aplikacija, scena, 2023.)

Ako se želi već u ovom stadiju raditi novi level, moguće je napraviti novu scenu unutar mape “Scenes”. To se radi desnim klikom unutar mape, zatim na “Create” i stisak na “Scene”. Upalit će se nova prazna scena gdje je moguće ubacivanje novih ili spremljenih objekata.

1.1.8. Interaktivnost i animacije

Između arene i čekaonice nalaze se vrata koja su zatvorena kad se uđe u novi level. Da se vrata otvore potrebno je stisnuti gumb pored vrata, a kada se prijeđe u arenu postoji “Trigger” (hrv. Okidač) koji, kada se prođe kroz njega, opet zatvori vrata da se ne može vratiti nazad. To je napravljeno preko skripti i ugrađenog dijela Unity-a koji omogućuje radnju animacija. Animacija je svaki pokretni dio levela koji na neku interakciju mijenja svoja svojstva. U igrici je to iskorišteno za vrata i još par dijelova do kojih će se kasnije doći.

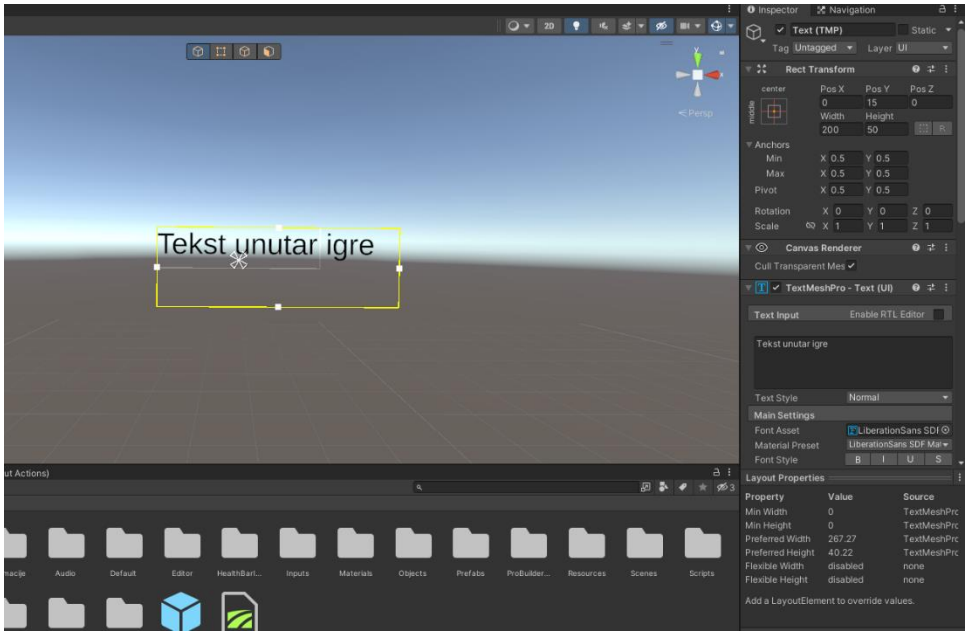
Slika 19 Zatvorena vrata (Izvor: Unity aplikacija, scena, 2023.)

Slika 20 Otvorena vrata (Izvor: Unity aplikacija, scena, 2023.)

Prvi korak u izgradnji interaktivnih dijelova je izrada nove skripte za interaktivnost. Prvo se radi skripta koja će funkcionirati kao šablona za sve interaktivne objekte koji se žele napraviti. U ovom slučaju to su vrata. Skripta koja je potrebna je “Interactable” i njen kod se nalazi pod naslovom “5.2.1. Interactable”. U njoj se nalazi kod koji će pozvati poruku koja će

se prikazati na ekranu, i na kojoj piše što je potrebno učiniti da se neki događaj odvije. Također, potrebno je napraviti novu skriptu za interakciju od strane igrača. Skripta će se zvati “PlayerInteract” i kod za tu skriptu nalazi se na “5.2.2 PlayerInteract”. Ta skripta omogućuje interakcija s interaktivnim elementima, ali također daje uvjet da objekt mora biti na određenoj udaljenosti da bi mogao nad njime raditi akciju. Potrebno je unutar inspektora napraviti novi “Layer” (hrv. Sloj) koji se nalazi na vrhu s desne strane te dodati novi Layer imenom “Interactable”. Nakon što se skripta izradila postavlja se na igračev objekt i ide se dalje. Također, bitno je da se u inspektoru kod skripte, kod opcije “Mask”, postavi izbor “Interactable”.

Na slici devetnaest i dvadeset moguće je s desne strane vidjeti “gumb” objekt koji služi za otvaranje vrata. Da on postane gumb preko kojeg se može odvijati radnja, potrebno mu je u inspektoru postaviti Layer u “Interactable”, da skripta može prepoznati koji je dio za interaktivnost. Bitna stvar je znanje kada je neki objekt interaktivan, a kada nije. To se može napraviti preko texta na ekranu. To se radi pomoću “Canvas” (hrv. Platno). Prva stvar koja se napravi je desni klik u Hijerarhiji i odabir na “UI” (eng. User Interface, hrv. Korisničko sučelje) i klikom na “Canvas”. Unutar Canvas ponovo se klikne desni klik i odabere se “Text (TMP)”. Na slici dvadeset i jedan moguće je vidjeti kako izgleda u sceni.

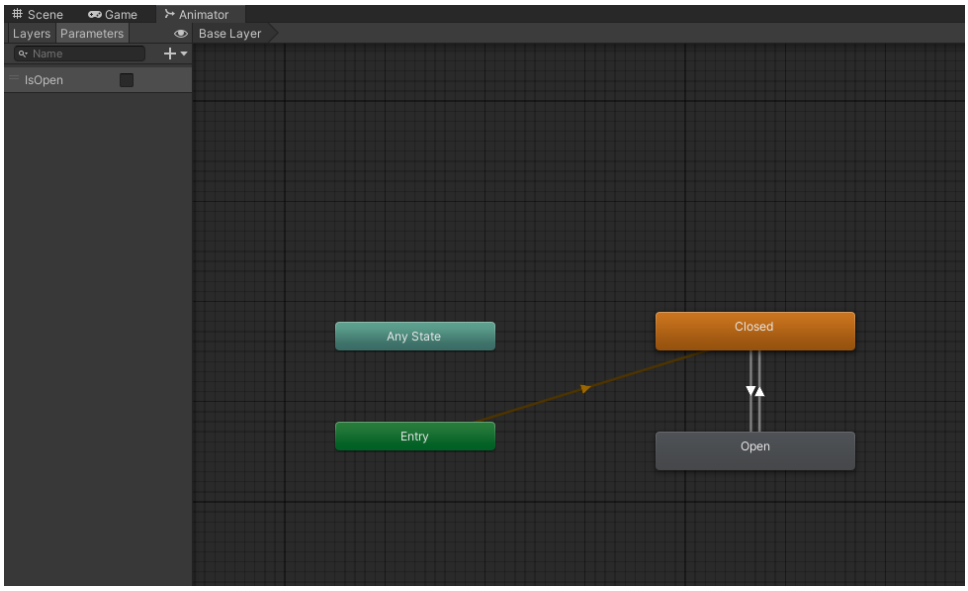


Slika 21 Tekst unutar scene (Izvor: Unity aplikacija, scena, 2023.)

Postoje već napravljene skripte od Input Managera koje se nalaze unutar instalacijske mape Input Systema, pa je dobro kopirati već gotove metode, imenom “Default Input” i napraviti novu skriptu od nje koja će se koristiti za interakciju u igri. Također, pametno je unutar Canvasa napraviti “Crosshair” (hrv. Nišan) koji služi za ciljane unutar igre. Isti je postupak kao kod pravljenja teksta na ekranu; za sliku u radu koristila se slika kruga, a boja bijela. Nakon toga se radi nova skripta imenom “PlayerUI” koja će sadržavati sve elemente koji se koriste za igračev pogled: Nišan, HP, Tekst na ekranu, itd. Kod za tu skriptu moguće je vidjeti ispod

naslova “5.1.5. PlayerUI”. Nakon što se kod postavio, kada se upali igra unutar Unity-a trebalo bi se moći vidjeti tekst kada se priđe gumbu za vrata.

Unutar igre je postavljeno da su vrata objekti koji imaju svoje karakteristike. Sad se želi da se na stisak gumba vrata otvore. To se radi preko animacije. Animacijski se prozor otvara preko gumba Window. Zatim Animation i klik opet na Animation. Kada je prozor za animaciju odabran bitno je odabrati objekt na kojem se želi raditi animacija. Nakon što je odabran objekt (u ovome radu to je “Door Parent”) klikne se na “Create”. Animacija se može spremiti bilo gdje u mapi, ali preporučuje se da se napravi nova mapa za animacije. Kada se odabrala mapa i ime, stisne se “Add Property”, preko kojeg se bira koje svojstvo se želi mijenjati u animaciji. U ovom će radu to biti “Position” (hrv. Pozicija). Nakon što se odabrala pozicija, u prvom se dijelu brišu zadnje točke i radi se nova animacija tako da se klikne na ime animacije i stisne “Create New Clip”. Potom se opet odabire na kakav se način želi promijenit objekt i obrišu se zadnje točke. Sada kada se odabrao “Position” odaberemo drugačiju koordinatu da se animacija može pomaknuti s jedne točke na drugu. Nakon što se odabrala animacija, otvara se “Animator” (Windows -> Animation -> Animator).



Slika 22 Animator za vrata (Izvor: Unity aplikacija, animator, 2023.)

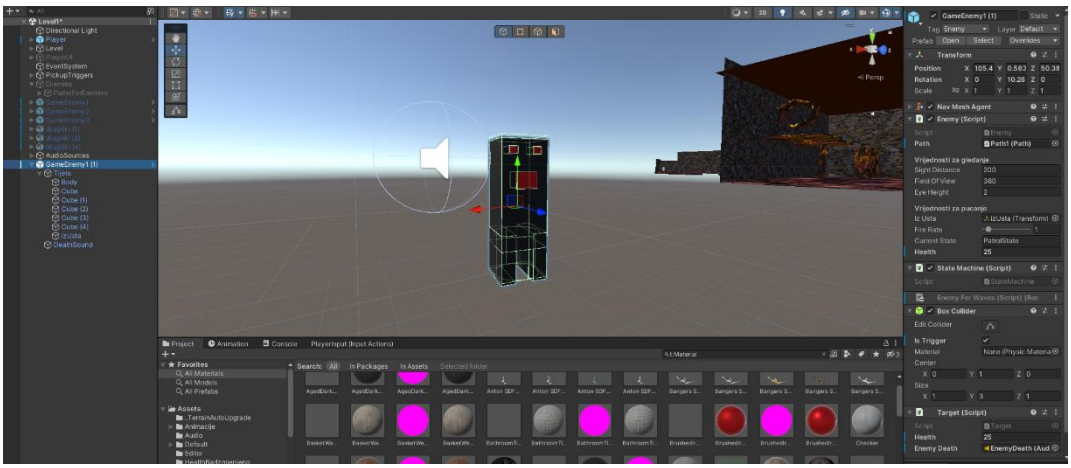
Dok je odabran objekt nad kojim su se radile animacije, u Animatoru će se pokazati sve animacije koje su izrađene. U slici dvadeset i dva vide se nekoliko stanja. U ovom primjeru “Entry” znači ulaz u igru, a ulaz u igru je spojen na “Closed”, klosed u ovom slučaju su zatvorena vrata. Drugi dio je “Open” a to je animacija dok su vrata otvorena. Između Open i Closed je dupla veza koja se okida kada je neki parametar zadovoljen. U ovom slučaju to je “IsOpen” koji se vidi na lijevoj strani. On se radi pomoću “Parameters”, no ako ne postoji klikne se na plus sa desne strane i odabere se kakva će biti varijabla kojoj će se iz koda moći pristupiti. U ovom slučaju je zadan kao bool vrijednost. To će se odraditi pomoću skripte “Keypad” koja se primjenjuje na Gumb unutar scene koja funkcionira kao okidač. Kada se

pritisne tipka “E” na tom gumbu, promijenitt će se stanje varijable “IsOpen”. Kod za skriptu “Keypad” je moguće vidjeti ispod naslova “5.2.3 Keypad”.

1.1.9. Neprijatelji

Poanta cijele igrice je borba protiv neprijatelja i prijelaz na sljedeći level. To znači da treba napraviti neprijatelje. U ovome će se radu objasnit dva načina za izradu objekta neprijatelja, jedan je kako u Unity-u od jednostavnih objekata stvoriti neprijatelja, a drugi način će biti objašnjen u Blenderu.

Početak radnje neprijatelja svodi se na kreiranje objekta koji će, nakon spajanja, dobiti svoje karakteristike i time mu biti dodijeljena svojstva. Prvi korak je napraviti prazni objekt i nazvat ga “Enemy” (ili po želji), nakon čega se mogu počet dodati 3D objekti po želji u bilo kojem obliku i bilo koliko njih (ovo je po želji stvaratelja igre, svako želi nešto drugačije stvoriti). Kada se stvorio željeni oblik možemo početi raditi skripte za neprijatelja.



Slika 23 Objekt neprijatelja (Izvor: Unity aplikacija, 2023.)

Poanta svakog neprijatelja u igrici je da stvara neki izazov igraću i, ovisno o igrici, da mu mogu nauditi. U ovom radu neki neprijatelji će trčati prema korisniku i ako ga dotaknu korisnik prima štetu. Drugi će tip neprijatelja biti koji čeka da mu u domet dolazi igrač i nakon toga počinju ispaljivati projektile u smjeru igrača. Oba neprijatelja će se objasniti i napisati će se koje su im potrebne skripte.

1. Pucajući neprijatelji: “Animator”: Potreban ako je neprijatelj napravljen unutar Blendera da se animacije mogu koristiti. “Box Collider”: Potreban da objekt može imati detekciju za granice, fiziku u igrici, detekcija za sudar, itd. “Enemy” (kod je ispod naslova “5.3.1 “Enemy”): Ovdje se nalazi cijela logika za neprijatelja. Koga će napadat, u kojem je stanju i iz koje točke će napast. Kod za cijeli neprijateljski AI (eng. Artificial intelligence, hrv. Umjetna inteligencija). Objašnjenje: postoji jedna skripta za pratnju trenutnog aktivnog stanja neprijatelja, jedna za mijenjanje njegovog stanja i tri skripte za tri

različita stanja od neprijatelja. 1. “Search State”: Neprijatelj traži igrača, 2. “Attack State”: neprijatelj napada igrača. 3. “Patrol State”: Neprijatelj ima svoju default putanju dok nema igrača u njegovom vidnom polju. Također u skripti se nalazi “Path” varijabla koja, kada se postavi, omogućuje postavljanje “rute” neprijatelju koju prati dok je u “PatrolState” prije nego što uđe u “AttackState”. U polje iz usta dodamo jednu praznu lokaciju iz koje je poželjno da izlaze projektili. Tu točku postavimo kao dijete cijelog objekta. Također u objektu mora biti “StateMachine” skripta. Sljedeća skripta je “Target”. U njoj se nalazi jednostavni kod koji se koristi za životne bodove tog objekta. Kada uđe u nulu, uništavamo taj objekt (tako funkcionira smrt neprijatelja). Također, postavljen je zvuk za njegovu smrt. Kod za skriptu može se pronaći ispod naslova “5.3.3 Target”.

2. Trčeći neprijatelji: Ova vrsta neprijatelja slična je kao i i pucajući, ali razlika između njih je u razlikama od jedne skripte. Umjesto “Enemy” skripte na ovoj vrsti neprijatelja nalazi se skripta “Enemy For Running” koja samo dobiva trenutnu lokaciju igrača i govori objektu da se treba izjednačiti s njegovom pozicijom na karti. Također, može mu se zadati brzina i s koje udaljenosti može opaziti igrača. Kod za skriptu moguće je naći ispod naslova “5.3.4 EnemyForRunning”.

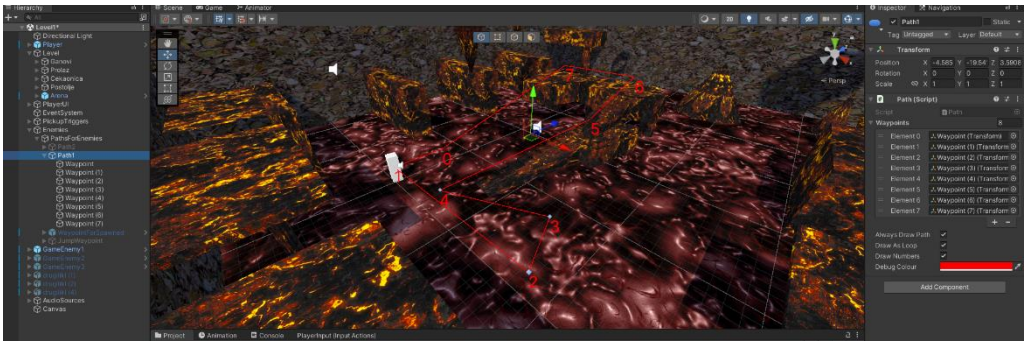
1.1.10. NavMesh

Za objašnjenje ponašanja neprijatelja i postavljene skripti koje im omogućuju kretanje po nivou, potrebno je dodati dodatne komponente unutar same scene kako bi se omogućilo kretanje neprijatelja. To je navigacija. Izgrađeni level potrebno je “ispeći”. To znači da je potrebno na sve dijelove levela postaviti dopuštene zone. Navigacija se postavlja tako da se pritisne na “Window”, zatim na “AI” i zadnje se klikne na “Navigation”. Otvara se novi window koji se zove “Navigation” i ima četiri dijela, ali u ovome radu su potrebni “Bake” i “Object”. U “Bake” dijelu se označuje kakve postavke se žele primijeniti nad “Agentima” kako bi se micali po odabranim dijelovima levela. Agenti su svi objekti koji imaju unutar karakteristika postavljeni “Nav Mesh Agent”. Bez te karakteristike objekt se ne bi mogao kretati unutar levela (to je ugrađena karakteristika unutar samog Unity-a). U navigacijskom je prozoru moguće namjestiti prije “pećenja” koliko se želi da neprijateljski objekt može prići blizu zidovima, naprimjer, može se namjestiti kolika je njegova visina ili pod kojim maksimalnim kutem se može penjati. Također, moguće je postaviti da neprijatelj može “skakati” između neke rupe preko “Jump Distance” opcije ili koliko daleko može pasti s visine preko “Jump Distance”. Da se postave sve potrebne lokacije za hodanje, potrebno je, dok je upaljen “Navigation” prozor, odabrati sve

objekte po kojima se želi da hodaju neprijatelji. Ako postoje neki objekti za koje se žele da se može skakati po njima prije nego što se ispeće prostor treba se otići u “Object”, odabrati ih i pod opcijom “Navigation Area” odabrati “Jump”. Nakon toga može se pritisnuti “Bake” (na sljedećoj slici se može vidjeti rezultat).

Slika 24 Prostor za hodanje (Izvor: Unity aplikacija, navigacija, 2023.)

Za navođenje neprijatelja po nekoj putanji dok je u PatrolState-u napravit će se nova skripta imenom “Path”. Kod za skriptu je moguće vidjeti ispod naslova “5.3.5 Path”. Ovom skriptom omogućeno je neprijatelju davanje koordiniata za patroliranje (ovo se napravilo da se osjećaju više živim). Da se napravi jedna putanja za neprijatelja, u hijerarhiji se napravi prazni objekt kojem se dodijeli skripta “Path”. Nakon toga se naprave djeca tog objekta koja su prazni objekti samo s nekom lokacijom. Kada su se odabrale sve lokacije, vratimo se na roditelja i u “Waypoints” listi postavimo sve “child” objekte. One su sada put koji će neprijatelj prijeći dok patrolira. Na sljedećoj slici se može vidjeti kako to izgleda.



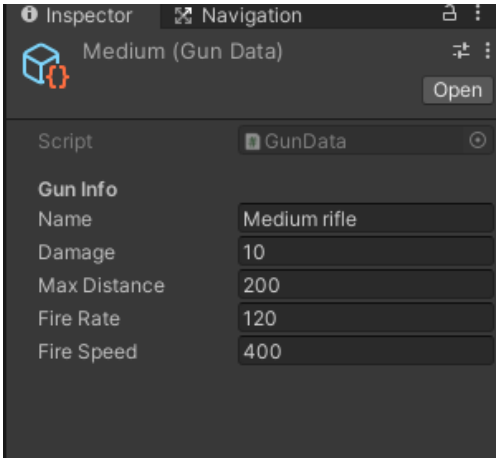
Slika 25 Putanja neprijatelja (Izvor: Unity aplikacija, scena, 2023.)

1.1.11. Oružje

Poanta ove igre su prolazak kroz arene u kojima se generiraju neprijatelji koje je potrebno uništiti. To će se raditi preko oružja koje igrač može odabrati i koristiti jednom kada su na

njemu. U ovome radu napravit će se tri oružja koje će na početku svakog nivoa moći pokupiti u čekaonici. Jednom kad se oružje izabere, ne može se više birati. Tri oružja su na raspolaganju: “Heavy” (Teško oružje), “Medium” (Srednje oružje), “Light” (Lako oružje). Kao u klasičnim igricama ovakvog tipa teško oružje ima visoku snagu, ali je sporo, dok je lako oružje brzo, ali ima najmanju snagu. Srednje je oružje balans između ta dva.

Oružje koje će se koristiti u ovom radu će biti napravljeno slično kao i “Enemy“ objekti u smislu da smo kombinirali 3D objekte u jedan cijeli objekt. Nakon što su se po želji izradila oružja ili skinula sa “Asset store”, radi se nova skripta koja će služiti kao brzi stvaralac karakteristika za puške. Skripta će se zvati “GunData” i kod za skriptu moguće je vidjeti ispod naslova “5.4.1 GunData”. Ovom skriptom sada je moguće stvarati karakteristike oružja. Desnim klikom u novoj mapi za oružje otvorimo na vrhu “Weapon” i stisnemo “Gun”, otvara se nova “Gun Data" objekt unutar inspektora u kojem možemo mijenjati GunInfo po želji.

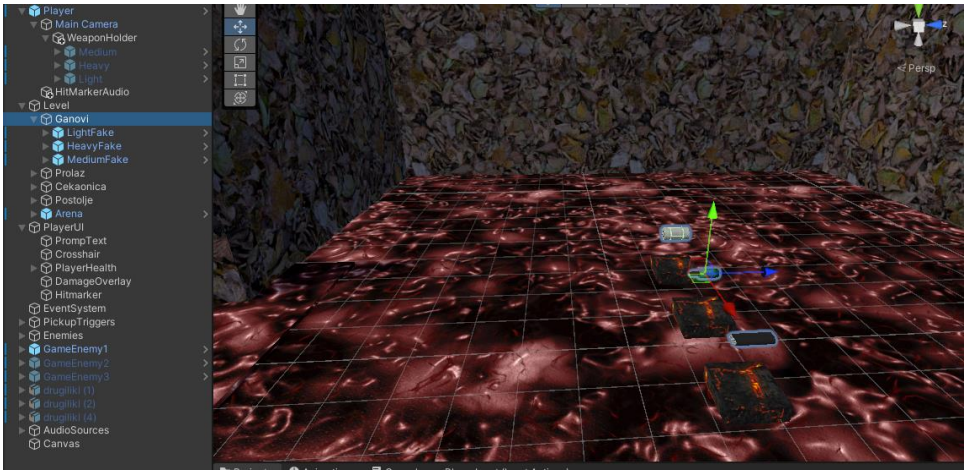


Slika 26 Karakteristike oružja (Izvor: Unity aplikacija, inspektor, 2023.)

Sljedeća skripta će biti potrebna za oružje da postane funkcionalno. Skripta će imat naziv “Gun” i može se pronaći ispod naslova “5.4.2 Gun”. Skripta se postavlja u izrađeno oružje i mogu se dodati karakteristike od koje su se izgradile od GunData skripte, u “Muzzle” je bitno staviti poziciju iz koje kreće paljba, to znači prazan objekt koji služi kao točka iz koje će biti ispaljeni metci. Ima još par opcija koje je moguće stavit - kao što su širenje metka, koliko veliko širenje metka, sustav udarnih čestica gdje se može dodati bilo kakav efekt kad metak udari u zid ili objekt i trag metka koji također može biti neki efekt. Pod “Mask” se samo odabere “Default”. Druga bitna skripta za pucanje je “PlayerShoot” koja će se baviti vidljivim dijelom pucanja, što znači da kada smo pogodili neprijatelja pokazat će se “hitmarker” (hrv. Marker pogotka). Skripta se postavlja na igračev objekt i može se vidjeti ispod naslova “5.4.5. PlayerShoot”. Da se vidi marker pogotka radi se ista stvar kao i kod ciljnika (Canvas-> Mijenjanje slike u poželjnu za prikazivanje pogotka neprijatelja). Sada funkcioniraju oružja i mogu se testirati na neprijateljskim objektima. U skripti “Target” bi se trebalo vidjeti da im se

umanjuje broj svakim pritiskom miša na njih za onoliko koliko je u karakteristikama oružja pod “Damage” zadano.

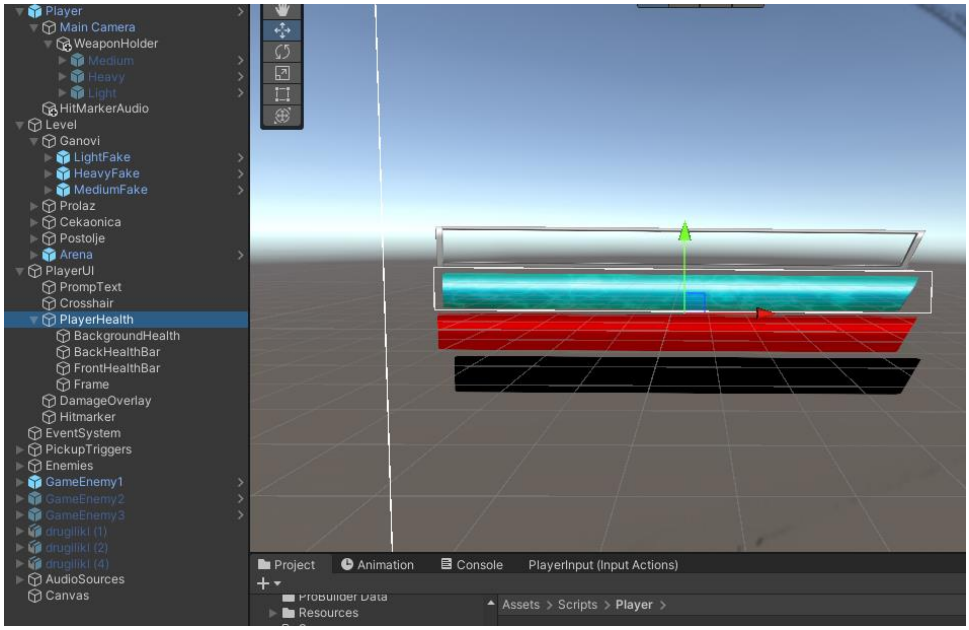
U ovom je radu napravljeno da ulaskom u čekaonicu postoje tri oružja koja se, kad se prođe preko njih, stvori u ruci igrača. Za to je potrebno napraviti “lažne” puške koje će samo biti korisne za prelaženje preko njih i aktiviranje prave puške u ruci od igrača. Prvo ih je potrebno postaviti negdje u level kuda će igrač proći i pokupiti jedno od oružja. Nakon toga je potrebno napraviti oko tog oružja “Box collider” koji se ponaša kao okidač. Radi se nova skripta imenom “PickupWeapon” (kod se može vidjeti ispod naslova “5.4.3 PickupWeapon”). Također, potrebna je skripta koja prati da li postoji već oružje koje je aktivirano na igraču ili ne. Ta će se skripta postaviti na prazan objekt koji će sadržati sva oružja koje je moguće pokupiti. Skripta će se zvati “WeaponState” i kod se može pronaći ispod naslova “5.4.4 WeaponState”. Skripta prati da li je prilikom učitavanja oružje aktivirano, ako je, onda ga ugasi. Dok drugi dio uključuje oružje koje je se aktivira prilikom aktiviranja skripte “PickupWeapon”.



Slika 27 Hijerarhija prvog i lažnog oružja (Izvor: Unity aplikacija, hijerarhija i scena, 2023.)

1.1.12. Život igrača

Sve su funkcionalnosti postavljene osim jedne da zapravo igrica može funkcionirati, a to je životni bodovi igrača. Prvo će se napraviti “PlayerHealth” skripta. Kod za skriptu može se naći ispod naslova “5.1.3. PlayerHealth”. U skripti se nalazi kod kojim se postavlja kojom brzinom izgubimo dio života, maksimalni broj životnih bodova koje igrač ima, slike koje će se mijenjati kada se primi šteta. U hijerarhiji se stvori novi element koji se zove “PlayerHealth” i u njega se ubacuju slike koje su željene da preko kojih će u igri na ekranu biti prikazan život.



Slika 28 Izgled zdravstvene trake (Izvor: Unity aplikacija, hijerarhija i scena, 2023.)

U ovome radu od komponenti na slici dvadeset i šest su komponente od kojih je sačinjen život od igrača. U “PlayerHealth” skripti postave se slike na “Front Health Bar” i “Back Health Bar”.

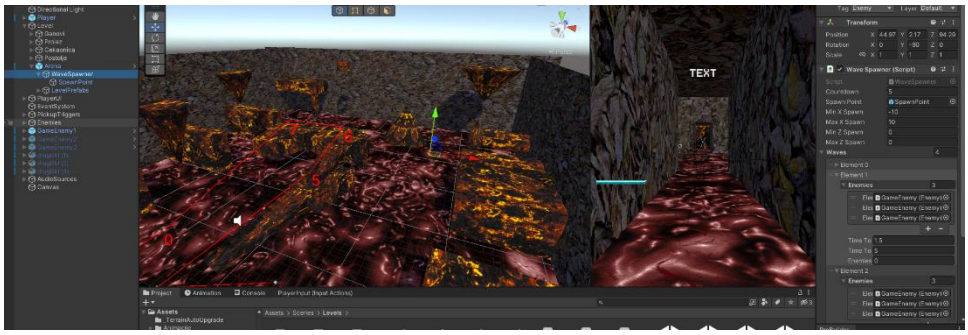
1.1.13. SpawnPoint i prolaženje igrice

Svi dosadašnji elementi su kreirani kako bi se dodale funkcionalnosti igri, no sada je potrebno te elemente povezati kako bi se oblikovala ukupna funkcionalnost igre. U ovome radu to se radi sa preživljavanjem i uništavanjem svih valova neprijatelja koji se stvaraju u određenom levelu.

Prva stvar koja se treba napraviti je nova skripta koja će imati funkcionalnost kao “spawner” za valove. Poanta igre je da će se iz određenih točaka stvarati neprijatelji i ti trebaš uništiti sve neprijatelje da se može otići na sljedeći level. Skripta je naziva “EnemyForWaves” i nalazi se ispod naslova “5.5.2 WaveSpawner”. Prvi korak je napraviti prazno dijete koje će se koristiti kao pozicija iz koje se neprijatelji stvaraju. Skripta funkcionira na način da se unutar inspektora gdje je zalijepljena skripta postavi broj valova koji se želi ugraditi u level. Unutar skripte moguće je mijenjati brojač preko kojeg se stavlja kada da počinju valovi i pozicija od kuda kreću valovi unutar arene. Moguće je dodati pomak od pozicije koja je postavljena tako da izgleda kao da je statično mjesto od kuda izlaze neprijatelji (na x i z kordinatama). Zatim se nalazi lista u koju se dodaju neprijatelji za igrača. Moguće je dodati bilo koliko neprijatelja i valova. Na kraju kada su uništeni svi neprijatelji prelazi se na sljedeći level.

Također, potrebna je skripta koja će ići na neprijatelja imenom “EnemyForWaves”. Kod za skriptu se može pronaći ispod naslova “5.5.1. EnemyForWaves”.Skripta je potrebna kako bi skripta za valove mogla identificirati određenog neprijatelja i smanjiti njihov broj u trenutnom

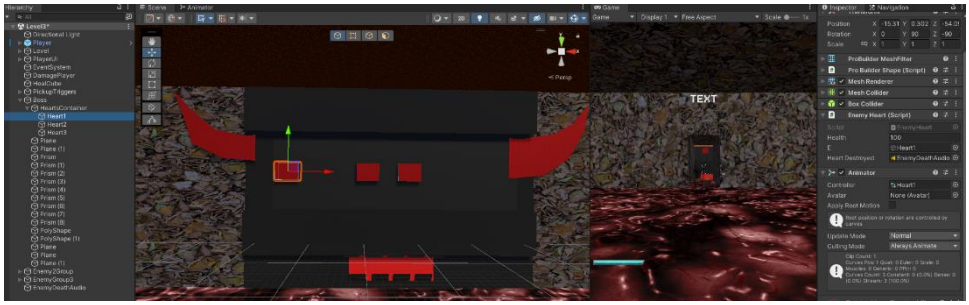
valu tako da se oduzmu od ukupnog broja neprijatelja u tom valu. Kada su svi neprijatelji uništeni zove se linija koja prebacuje igrača na sljedeću scenu.



Slika 29 Spawner za neprijatelje (Izvor: Unity aplikacija, hijerarhija i scena, 2023.)

1.1.14. Neprijateljski šef

Igrica se sastoji od tri levela, od kojih su prva dva “Arena survival” (hrv. Preživljavanje u areni), dok se treći level sastoji od “Boss” (hrv. Glavnog šefa) neprijatelja kojeg je potrebno uništiti da igrica završi.



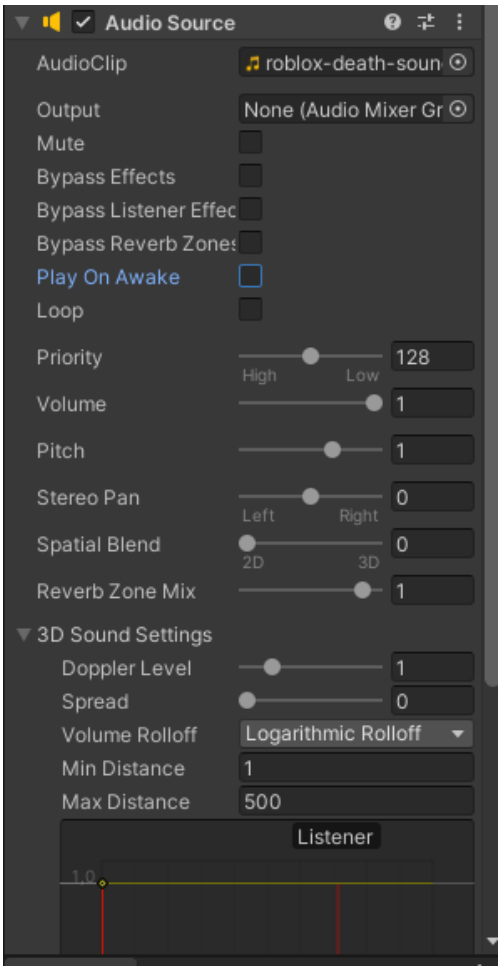
Slika 30 Izgled zadnjeg neprijtelja (Izvor: Unity aplikacija, hijerarhija i scena, 2023.)

Zadnji neprijatelj se sastoji od svega što se naučilo od ranijih skripti i radnji u Unity-u. Da se pobijedi potrebno je uništiti tri oka u njegovoj glavi. Oni se pomiču pomoću animatora, gdje mu se konstantno pomiču oči u neku stranu, dok njegove oči prekrivaju kapci koji se također pomiču gore/dolje kako bi zaštitili bolne točke neprijateljskog šefa. Napravljena je nova skripta za njegove oči da se može brojati koliko je očiju uništeno i da se onda može napraviti kraj igre. Kod od skripte se nalazi ispod naslova “5.6.1. EnemyHeart”. Svako oko ima istog roditelja koji prati njihovo uništenje. Taj roditelj ima skriptu “ParentOfTheEnemy” i kod se može pronaći ispod naslova “5.6.2. ParentOfTheEnemy”. Ta skripta dobiva broj uništenih očiju i kad se svi unište prebacuje na novu scenu gdje piše da je igrica gotova.

1.1.15. Audio

Dodavanje Audia unutar igre vrlo je jednostavno. Samo je potrebno imati mp3 datoteku. Unutar hijerarhije na bilo koju poziciju se napravi prazni objekt u kojemu se preko “Add Component” doda “Audio Source”. Unutar Audio Sourca, u prvi red gdje piše “AudioClip”, samo

se treba dodati mp3 datoteka. Sada postoji Audio objekt koji se može dodati u bilo koju skriptu pomoću linije: `public AudioSource primjer`. Sada se u kodu na bilo koje mjesto može dodati linija: `primjer.Play();` , u bilo koji dio izvedivog koda, i taj zvuk će se čuti. Također u “AudioSrouce" je bitno ugasiti opciju “Play on awake” da se prilikom paljenja igre ne čuje taj zvuk.



Slika 31 Audio Source komponenta (Izvor: Unity aplikacija, inspektor , 2023.)

1.1.16. 2D scene

Za izradu ekrana koji se koriste kao zasloni za početak igre, izlazak iz igre ili ponovno pokretanje igre kada igrač umre ili pobijedi, potrebno je stvoriti nove scene. U toj svrsi, stvara se nova scena u game engineu koja sadrži "Canvas" (platno) i "Event System" (sustav događaja) u hijerarhiji objekata.

Nakon dodavanja "Panela" (ploče) unutar "Canvasa," korisnik može dodati gumbe i tekst po želji. Gumbovi služe za omogućavanje igračima ulaska u igru, izlaska iz nje ili ponovnog pokretanja igre.

Za upravljanje funkcionalnostima ovih ekrana, potrebno je kreirati dvije nove skripte pod nazivima "StartMenu" i "EndMenu." Te skripte se povezuju s odgovarajućim gumbovima u inspektoru, gdje se također postavljaju funkcije koje će se izvršiti prilikom klika na gumbe.

Skripta "StartMenu" implementira funkcije za prelazak unutar igre, dok "EndMenu" skripta sadrži funkciju za gašenje igre na klik. Na taj način, igrači mogu lako navigirati između različitih zaslona igre, ulaziti u igru, izlaziti iz nje ili je ponovno pokretati.

1.1.17. Mijenjanje scene

Scene u igri se mijenjaju kada se izvrši neka veća radnja. To može biti smrt Igrač, pa je zato prebačen na “Game over” prozor, ili ako su uništeni svi neprijatelji, prebaci igrača na sljedeći level, Početni zaslon, itd.
Prvo što je potrebno za mijenjanje scena su više scena. U ovome radu napravljeno je tri scena koje je moguće igrati i tri scene koje su 2D i koje se koriste kao Menu scene. Da bi igrica funkcionirala treba dodati sve scene unutar “Build-a” (hrv. Izgradnje) da se mogu pristupiti iz jedne u drugu. Za njihovu povezanost, potrebno je biti u sceni koja nije uključena u izgradnju igre, zatim kliknuti na “File”, “Build Settings” i na gumb “Add Open Scenes”. Sada je moguće iz bilo kojeg dijela koda pristupiti toj sceni sa linijom:

```
SceneManager.LoadScene(#indeksni broj scene)
```

Dok je na vrhu potrebno napisati:

```
using UnityEngine.SceneManagement;
```

Kada su se dodale sve scene na kraju se može pritisnuti “Build” i zatim se dobiva gotovi File koji pokreće igricu.

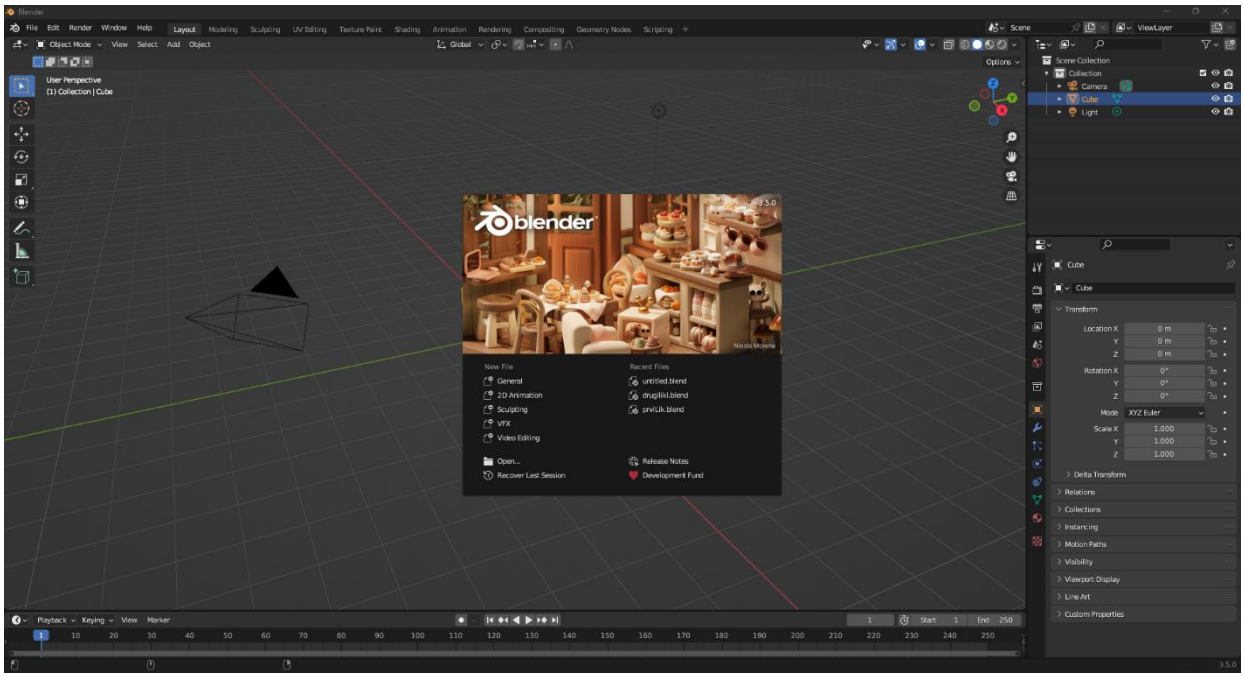
4. Blender

Blender je besplatni i “Open source” (hrv. Otvoreni izvor) 3D paket za stvaranje. Blender podržava cijeli cjevovod što u značenju znači da podržava modeliranje, namještanje, animacija, simulacija, itd. (“The software”, 2023). Blender je odličan program za stvaranje novih 3D modela, ali za početnike je užasno opširan i kompliciran. Možda nije najpametnije za

početne korisnike da kreću sa njime, ali u ovom radu će se napisati postupak preko kojeg će se objasniti postupak za kreiranje modela i animacije.

5.1. Početak modeliranja

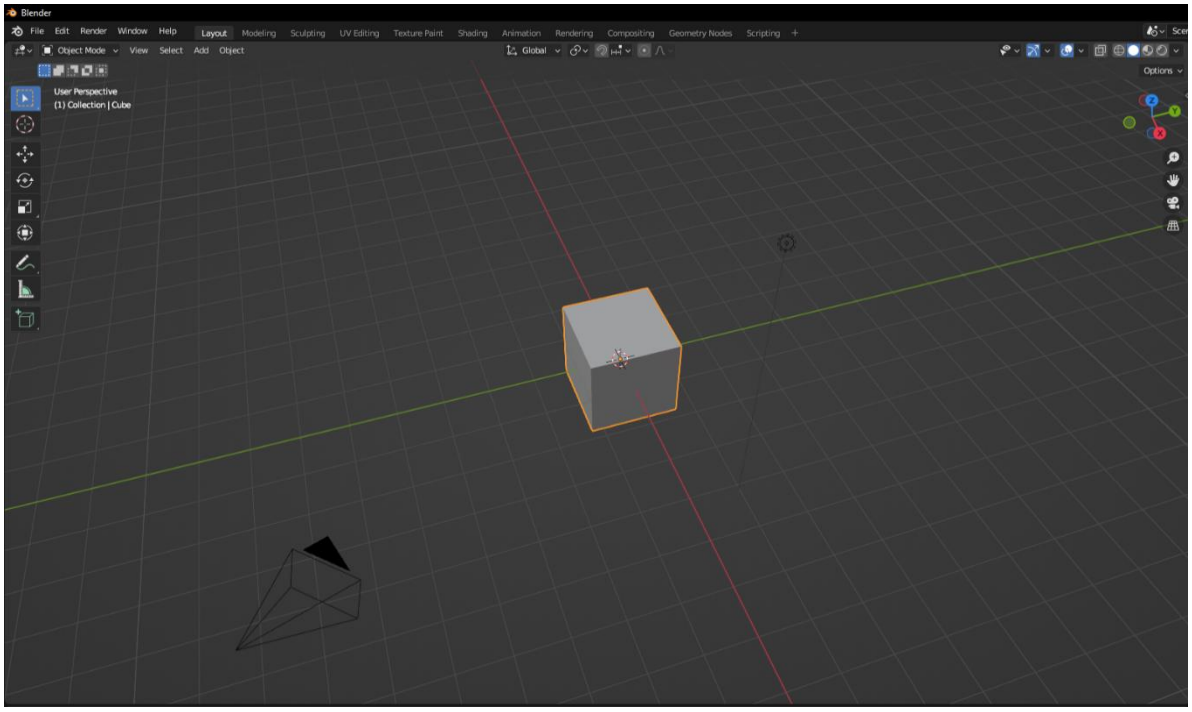
Nakon instalacije i kada se otvori program, prva stvar što je vidljiva je odabir projekta.



Slika 32 Blender (Izvor: Blender aplikacija, 2023.)

Na slici je moguće vidjeti početni ekran koji prikazuje opcije za odabir početka rada. Moguć je odabir novog projekta ili već postojećeg. U ovom slučaju će biti pritisnut “General” za nastavak.

Nakon odabira na General, prikaže se model kocke na kojemu se može početi radit.



Slika 33 Početni prozor (Izvor: Blender aplikacija, 2023.)

U ovom se radu neće objasniti svaki dio programa, jer je Blender ogroman, nego će se samo objasniti oni potrebni dijelovi za modeliranje preko kojeg će se napraviti model za videoigru, te kako ga animirati da se može prebaciti u Unity kao objekt za korištenje.

1.1.18. Alatna traka

Slika 34 Alatna traka (Izvor: Blender aplikacija, 2023.)

Na lijevoj strani ekrana će se koristiti alatna traka koja ima opcije potrebne za kreiranje prvog modela. Po redu opcije su (“Editors”, 2023):

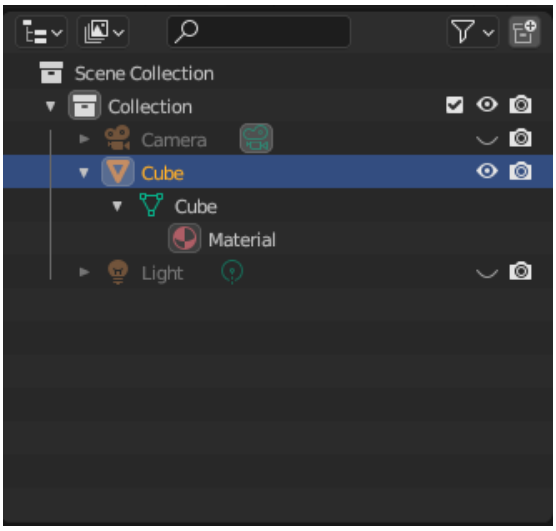
- Select Box: Opcija za odabir predmeta unutar programa (moguće mijenjanje načina odabira sa klikom “w” na tipkovnici).

- Cursor: Točka u prostoru koja ima lokaciju i rotaciju, ima više svrha, ali jedna od njih je da definira gdje su se postavili novi objekti.
- Move: Koristi se za pomicanje objekta u prostoru po x, y i z koordinati.
- Rotate: Koristi se za okretanje objekta u prostoru po x, y i z koordinati (0-360 stupnjeva).
- Scale: Promjena veličine objekta.
- Transform: Kombinacija “Move” i “Rotate”.
- Annonate: Pisanje primjedba na aktivnim podacima.
- Meassure: Pomicanjem miša može se nacrtati linija u prostoru koja će napisati kolika je dužina između početne i krajnje točke.
- Add Cube: Opcija za dodavanje nove kocke u prostor.

1.1.19. Hijerarhijski prozor i inspektor

S desne strane programa nalaze se hijerarhijski prozor i inspektor preko kojih se mogu mijenjati karakteristike objekta.

1.1.1.1. Hijerarhijski prozor



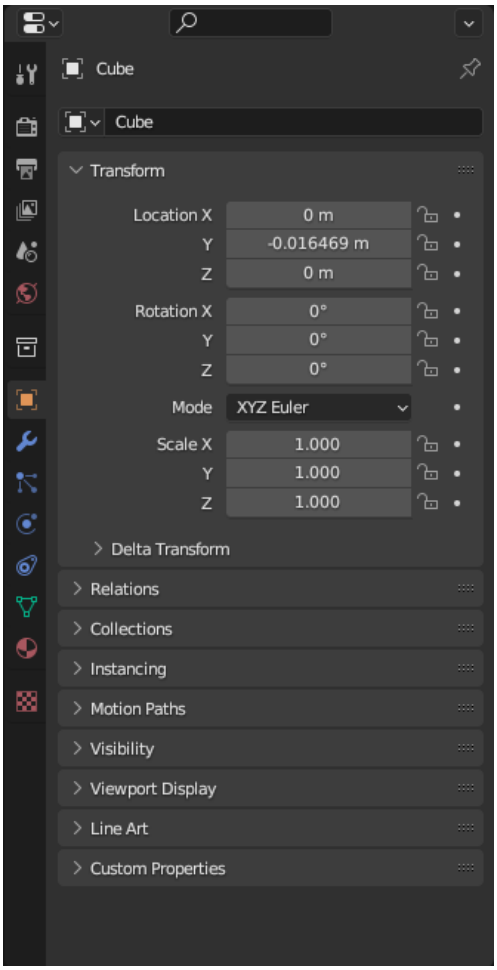
Slika 35 Hijerarhijski prozor (Izvor: Blender aplikacija)

Slično kao i u Unity-u ovdje se mogu vidjeti svi objekti ili predmeti koji su unutar scene. Na početku modeliranja u prozoru su prisutni "Camera”, “Cube” i “Light”. U kontekstu ovog rada bitan je samo “Cube” objekt nad kojim će se raditi promjene. S lijeve strane svakog objekta, ako postoji, može se kliknuti na strelicu da vidimo djecu tog objekta. S desne strane

sličica oka predstavlja vidljivost unutar prostora. Kada se stisne na nju i “oko” izgleda kao da je zatvoreno, znači da se objekat ugasio unutar prostora. Desno od tog je “kamera” koja služi za isključivanje objekta iz “renderanja” (hrv. Prikazivanje).

1.1.1.2. Inspektor

Inspektor unutar Blendera je sličan kao u Unity-u, u smislu da se u njemu također vrši radnja nad odabranim objektom i postavljaju mu se karakteristike koje mi želimo.



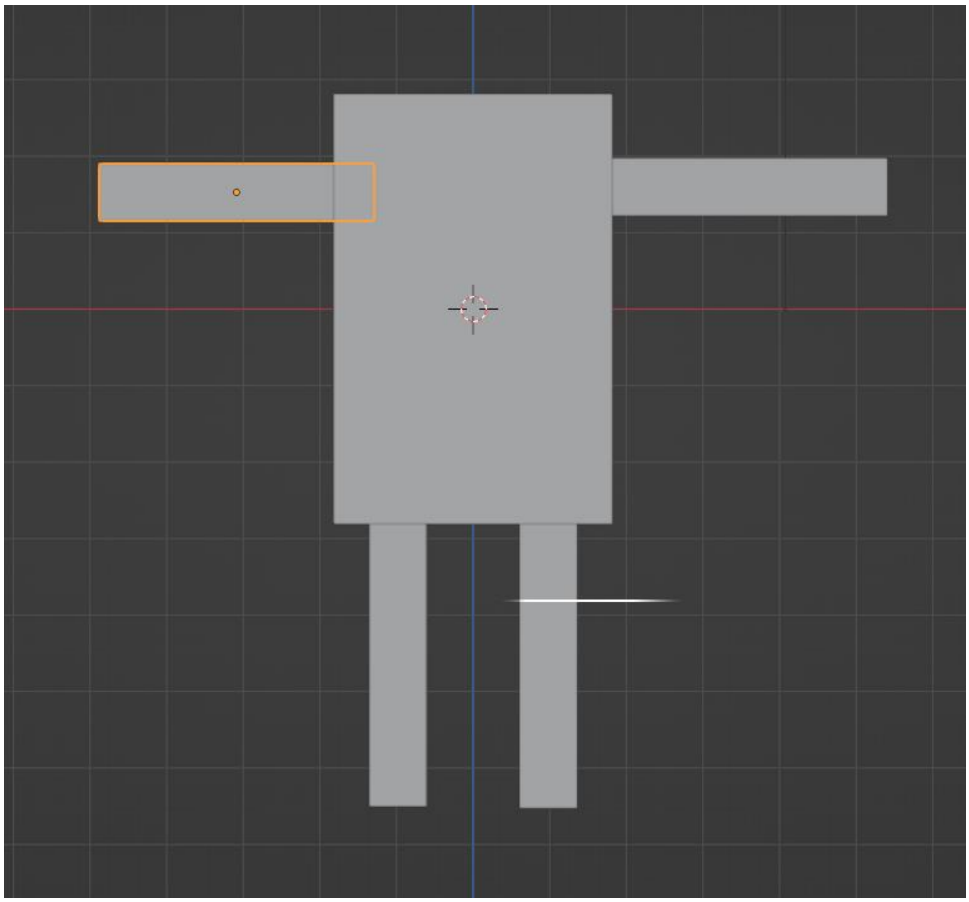
Slika 36 Inspektor za objekt (Izvor: Blender aplikacija, 2023)

Inspektor za karakteristike nad objektom su vrlo opsežne i nisu sve bitne kada se želi napraviti jednostavan model za videoigru. Zato je u ovome radu samo bitan “Object Properties”. U tom prostoru najbitnije će biti namještanje objekta i njegovo okretanje.

6.1. Prvi model

Prvo što se vidi u aplikaciji je kocka na kojoj su moguće promijene. Na korisniku je kako želi da mu izgleda prvi model. Za ovaj rad napravit će se jedan jednostavan model u humanoidnom obliku koji će nekako biti najbliži čovjeku, pa je nekako najjednostavnije razmišljati gdje mu se nalaze kosti i kako bi se trebao pomicati. Bitno je da se u gornjem desnom kutu namjesti pogled iz –Y koordinate.

U inspektoru se može oblikovati kocka po izboru i, ako je potrebno, sa opcijom “Add Cube” dodaje se još kocka na prvu kocku. Za početnike je najbolje onda tu novo napravljenu kocku kopirati (shift + D) i postaviti sa druge strane tako da se čini kao da objekt ima dvije noge. Nakon toga mogu se dodati ruke, ako korisnik želi.

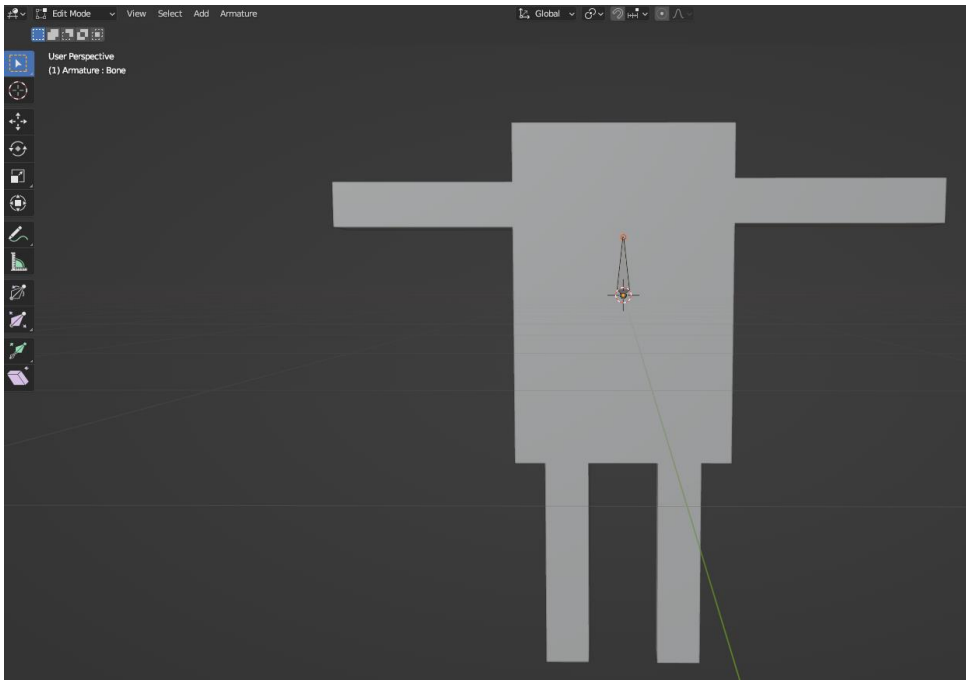


Slika 37 Početni lik (Izvor: Blender aplikacija, 2023)

Kada je korisnik zadovoljan s likom, potrebno je odabrati sve dijelove tijela i kliknuti ctrl + A i izabrati “All Transforms” i nakon toga ctrl + J da se svi objekti lika spoje u jedan objekt.

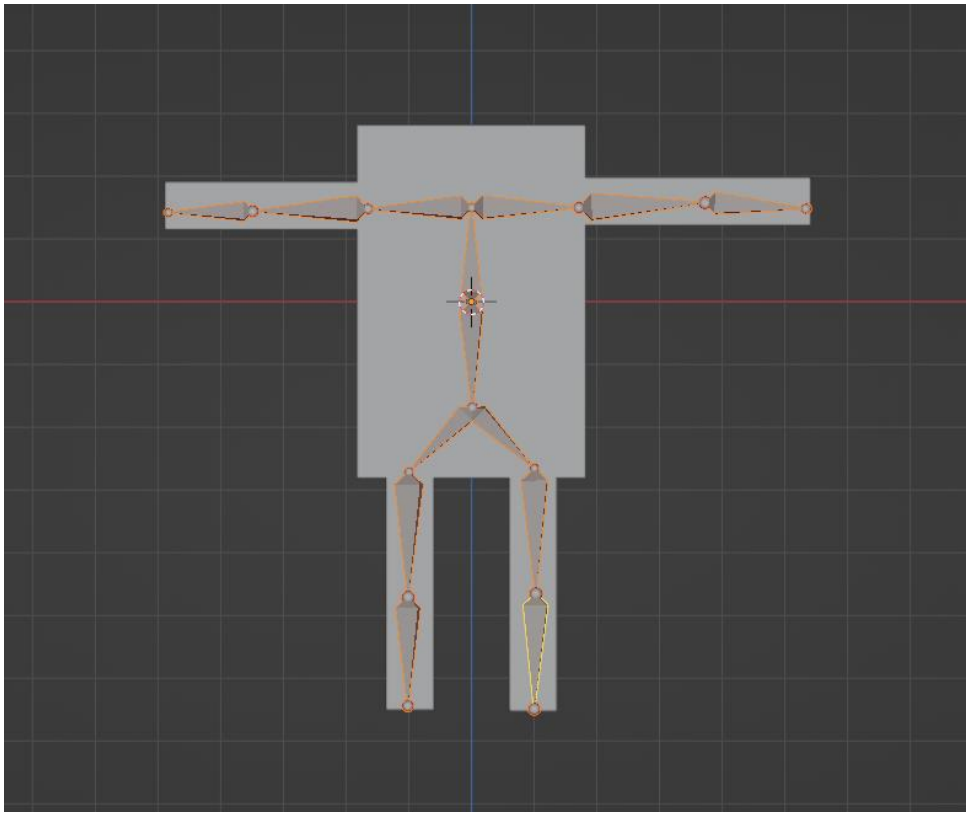
7.1. Dodavanje kostiju

Nakon što je korisnik zadovoljan svojim likom dodaju se kosti. Kosti se dodaju sa pritiskom na “Add” koji se nalazi na gornjoj strani programa, zatim “Armature” i onda “Add Bone”. Nakon što je dodao kost bitno je promijeniti način interakcije iz “Object mode” u “Edit Mode” (gornji desni kut programa). Kada se doda nova kost također je pametno u inspektoru otići u “Object Data Properties” i pod “Viewport Display” kliknuti na opciju “In Front” da se kosti mogu vidjeti ispred modela.



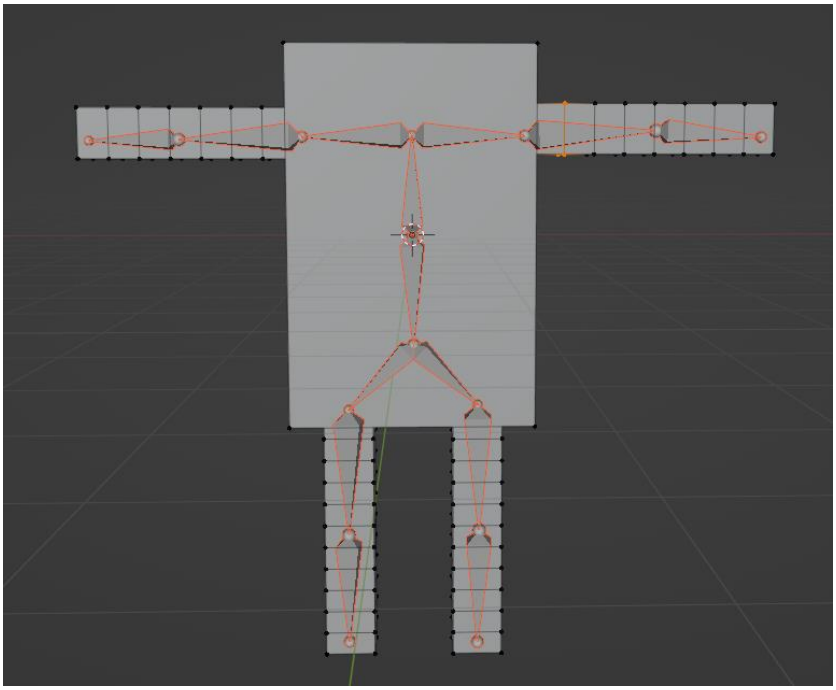
Slika 38 Kost u liku (Izvor: Blender aplikacija, 2023)

Kost se koristi tako da je manji dio onaj iz koje se radi rotacija, dok je veći dio onaj gdje kost kreće. Pa tako kad se postavi prva kost iz samog vrha može se pritisnuti slovo E na tipkovnici i iz toga vrha će izaći nova kost. Pametno je nakon svake postavljene kosti okružiti cijeli objekt i vidjeti da li je dobro postavljena, ako nije može se sa pritiskom na slovo R ispraviti njena pozicija.



Slika 39 Cijeli postavljeni kostur (Izvor: Blender aplikacija, 2023)

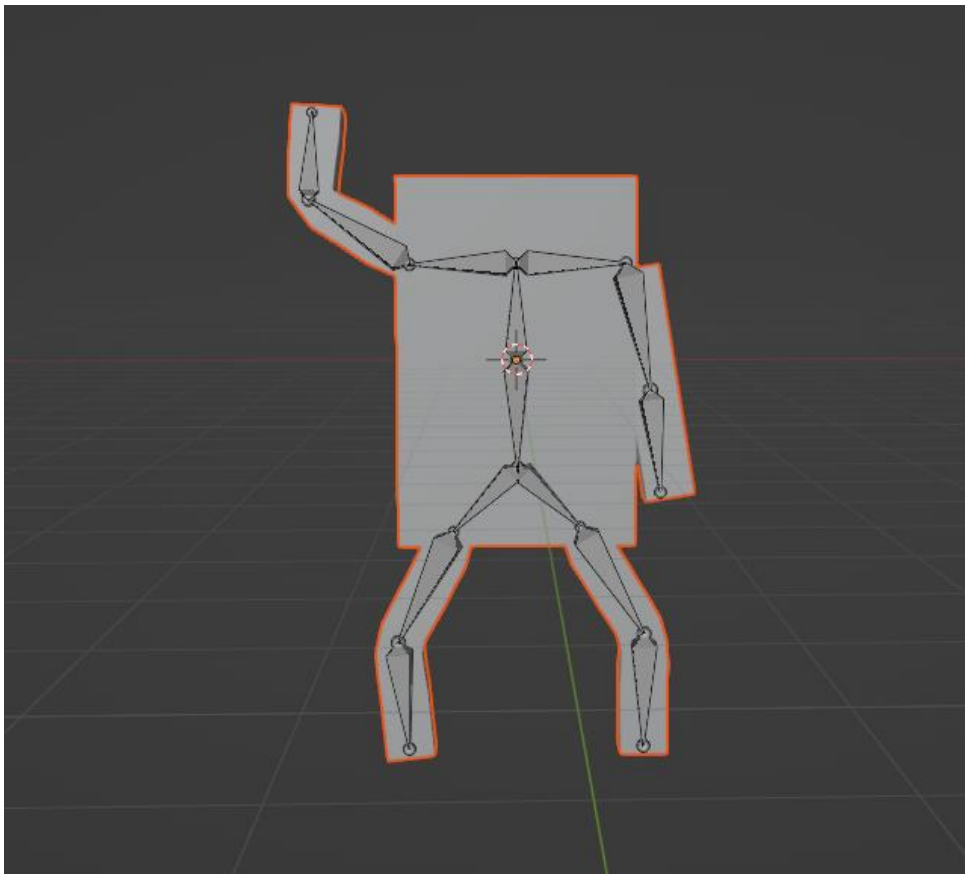
Kada je postavljen cijeli kostur, potrebno je otići u “Layout” tab i odabrati “Loop Cut”. Nakon što je odabrano, trebaju se staviti svugdje izrezi gdje se očekuje da će se kosti okretati.



Slika 40 Objekt sa izrezima (Izvor: Blender aplikacija, 2023)

Zatim je potrebno spojiti kostur sa objektom da kosti mogu upravljati objektom. U “Layout” tabu se ponovno odabere “Object Mode” i zatim se odaberu objekt PA kosti. Vrlo je

bitno da se prvo klikne na objekt pa se zatim sa shift klikom stisne na kostur! Stisne se desni klik, klikne se na “Parent” i odabere opcija “With Automatic Weights”. Ako se sve dobro napravilo, moguće je provjeriti hoće li se objekt dobro pomicati u “Pose Mode”. Klikne se na kosti i u gornjem lijevom kutu sa “Object mode” prebaci se na “Pose mode”. Odabere se jedna kost i stisne se na tipkovnici slovo R.

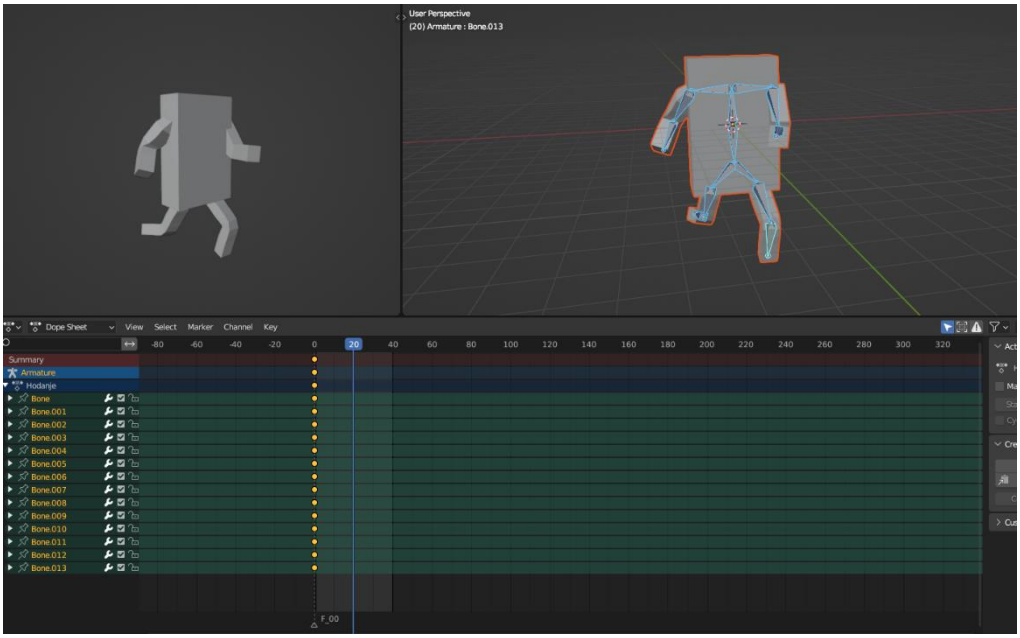


Slika 41 Objekt s dobro postavljenim kosturom (Izvor: Blender aplikacija, 2023)

8.1. Stavljanje animacije

U gornjem dijelu aplikacije klikne se na “Animation”, i onda se dolje lijevo odabere umjesto “Dope sheet” -> “Action Editor”. Sada se može nazvati svaka koja će se napraviti. Za ovaj primjer će se napraviti hodanje. U inspektoru se odabere “Output properties” i u “Frame Range” se odabere koliko dugo da traje animacija. U ovom primjeru će biti 60 frameova.

Da bi se krenulo animirati, prvo je potrebno odabrati kosti koje se žele mijenjati. Zatim ih sa slovom R treba namjestiti u pozu koju korisnik želi. Nakon što je model postavljen u željenu poziciju, označe se sve kosti i stisne se slovo I na tipkovnici i odabere se opcija “Location, Rotation & Scale”.



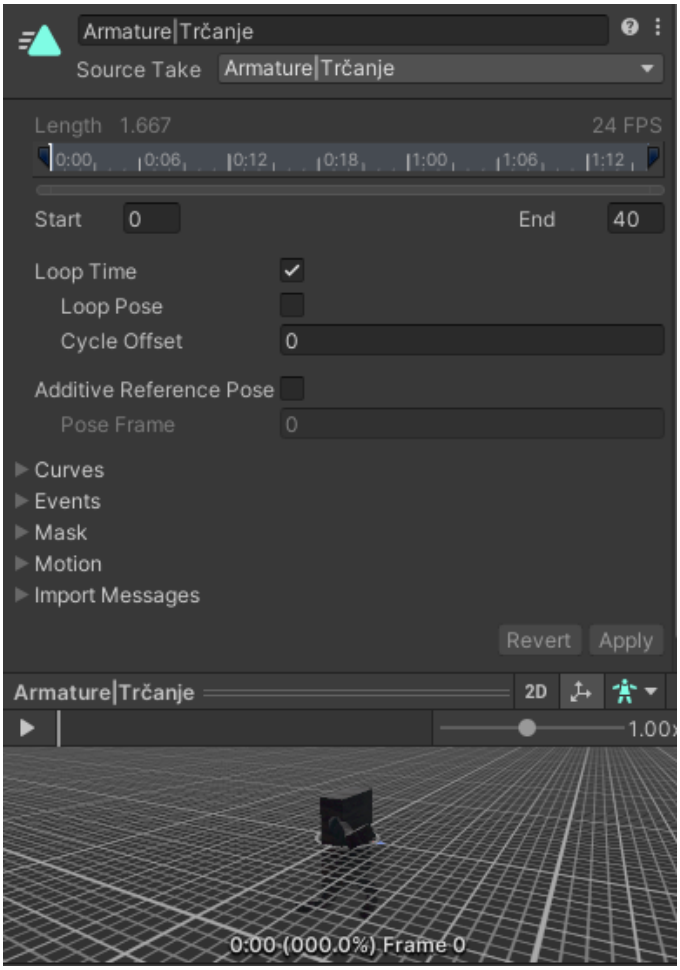
Slika 42 Objekt s dobro postavljenim kosturom (Izvor: Blender aplikacija, 2023)

Na ekranu bi trebalo ovako izgledati, nakon toga se pomakne plava linija (u ovom slučaju za 20 frameova) i radi se sljedeća poza. Nakon kada se odradila druga poza, ako se želi stvoriti fluidna animacija, dobro je kopirati prvu pozu i staviti je na zadnji Frame da imamo prividnost trčanja. Sada je izrađen lik sa jednostavnom animacijom.

9.1. Izvoz lika u Unity

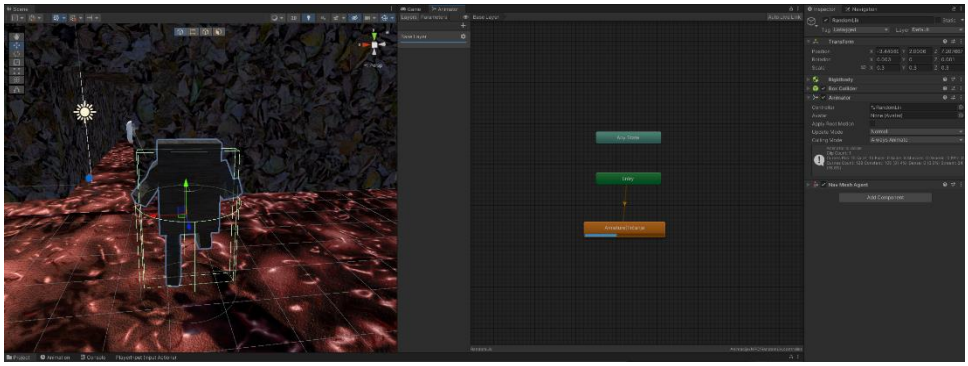
Kada je korisnik zadovoljan sa napravljenim likom, može ga izvest u Unity kao objekt. To se radi sa odlaskom na “File” (gornji desni kut), “Export”, “FBX”. Odabere se putanja tog objekta (pametno je u izravno u projekt da se ne izgubi), i u opcijama treba izmijeniti par stvari. S desne strane unutar “Include“ odabrati samo “Armature” i “Mesh” (ostalo je već postavljeno unutar Unity-a), u “Transform” odabrati za “Forward”: “-Z Forward” i “Up”: “Y Up” i u “Armature” samo ostaviti “Add Leaf Bones”, ostaviti upaljeno “Bake Animations”.

Sada kada je model ubačen unutar Unity-a, potrebno je obaviti par stvari. Prvo, ako se želi da se animacija ponavlja, treba unutar lika ući u “Animations”, odabrati animaiju koju bi željeli ponavljat i stisnuti na kučicu “Loop Time”.



Slika 43 Postavljanje da se animacija ponavlja (Izvor: Unity aplikacija, 2023)

Unutar samog modela potrebno je dodati sve skripte za neprijatelja ako se koristi kao neprijatelj. Također, ako se želi upaliti animacija bitno je dodati pod komponente “Add Component”, i odabrati “Animator Control”. Ovdje će se staviti novi “Animator Controler”. Da bi animacija proradila napravi se “Animator Controler” u neku mapu, koju je najbolje nazvati “Animacije” i unutra napraviti novi kontroler za tog lika. Unutar kontrolera potrebno je staviti animaciju za koju se želi da se izvodi. U ovom slučaju to je “Trčanje”. Kada se animacija ubacila trebala bi biti spojena sa “Entry”. Entry znači da kada se uđe u igru da će to biti prva stvar koja će se pokrenuti. Spremi se kontroler i u inspektoru od ubačenog lika u scenu, unutar Animatora, u “Controller” ubaci se novo izrađeni kontroler i na pritisak “Play” unutar Unity-a, trebali bi vidjeti da taj objekt izvodi animaciju.



Slika 44 Izvođenje animacije (Izvor: Unity aplikacija, 2023)

5. Skripte i kodovi

10.1. Skripte za igrača

1.1.20. InputManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
public class InputManager : MonoBehaviour
{
    private PlayerInput playerInput;
    public PlayerInput.OnFootActions onFoot;
    private PlayerMotor motor;
    private PlayerLook look;

    void Awake()
    {
        playerInput = new PlayerInput();
        onFoot = playerInput.OnFoot;
        motor = GetComponent<PlayerMotor>();
        look = GetComponent<PlayerLook>();
        onFoot.Jump.performed += ctx => motor.Jump();
        onFoot.Crouch.performed += ctx => motor.Crouch();
        onFoot.Sprint.performed += ctx => motor.Sprint();

    }
    void FixedUpdate()
    {
        motor.ProcessMove(onFoot.Movement.ReadValue<Vector2>());
    }
    private void LateUpdate()
    {
        look.ProcessLook(onFoot.Look.ReadValue<Vector2>());
    }
    private void OnEnable()
    {
        onFoot.Enable();
```

```
    }
    private void OnDisable()
    {
        onFoot.Disable();
    }
}
```

Ova skripta je preuzeta od “#1 FPS Movement: Let’s make a First Person Game in Unity!, Natty GameDev, 08.12.2012.” Skripta omogućuje kretanje i kontrolu gledanja igrača koristeći “PlayerInput” komponentu i ostale komponente “PlayerMotor” i “PlayerLook”. “playerInput” varijabla zasrži referencu “PlayerInput” komponente i onda pomaže pri ulazu igrača. “onFoot” varijabla sadrži referencu “onFoot” komponente koja upravlja kretanjem igraća. Varijabla “look” sadrži referencu na komponentu “PlayerLook” koja upravlja kontrolom gledanja igrača. Awake metoda se izvršava kada se pokrene igra. Ostale Update metode se izvršavaju tijekom svake sekunde upaljene igre.

1.1.21. PlayerMotor

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

public class PlayerMotor : MonoBehaviour
{
    private CharacterController characterController;
    private Vector3 playerVelocity;
    public float speed = 10f;
    private bool isGrounded;
    public float gravity = -9.8f;
    public float jumpHeight = 3f;
    bool crouching = false;
    float crouchTimer = 1;
    bool lerpCrouch = false;
    bool sprinting = false;

    void Start()
    {
        characterController = GetComponent<CharacterController>();
    }
    void Update()
```

```
{
    isGrounded = characterController.isGrounded;
    if (lerpCrouch)
    {
        crouchTimer += Time.deltaTime;
        float p = crouchTimer / 1;
        p *= p;
        if(crouching){
              characterController.height=Mathf.Lerp(characterController
        .height, 1, p); }
  else{
  CharacterController.height = Mathf.Lerp(characterController.height,
  2, p);  }
        if (p > 1)
        {
            lerpCrouch = false;
            crouchTimer = 0;
        }
    }
}
public void ProcessMove(Vector2 input)
{
    Vector3 moveDirection = Vector3.zero;
    moveDirection.x = input.x;
    moveDirection.z = input.y;
   characterController.Move(transform.TransformDirection(moveDirection)
        * speed * Time.deltaTime);
    playerVelocity.y += gravity * Time.deltaTime;
    if(isGrounded && playerVelocity.y < 0)
    {
        playerVelocity.y = -2f;
    }
    characterController.Move(playerVelocity * Time.deltaTime);
}
public void Jump()
{
    if(isGrounded)
    {
        playerVelocity.y = Mathf.Sqrt(jumpHeight * -3.0f * gravity);
    }
}
```

```
    public void Crouch()
    {
        crouching = !crouching;
        crouchTimer = 0;
        lerpCrouch = true;
    }
    public void Sprint()
    {
        sprinting = !sprinting;
        if (sprinting) speed = 30;
        else speed = 15;
    }
}
```

Ova skripta je preuzeta od “#1 FPS Movement: Let’s make a First Person Game in Unity!, Natty GameDev, 08.12.2021.”. Ova skripta je odgovorna za upravljanje igrača unutar igre. Preko ove skripte dobiva se sposobnost kretanja, skakanja, čučnja i sprintanja. Za ovu skriptu potrebno je da igrači objekt ima na sebi “RigidBody” karakteristiku. Varijable koje su prisutne rade ono što im se u imenu kaže ali bitno je naglasti da varijabla “lerpCrouch” služi za gledanje da li je trenutno aktivna animacija skakanja ili čučnja. Dok funkcije imenima za neku radnju upravo rade to kako se zovu, Update metoda gleda da li je igrač na tlu da može izračunavati sljedeće akcije.

1.1.22. PlayerHealth

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

public class PlayerHealth : MonoBehaviour
{
    private float lerpTimer;
    private float health;
    [Header("Health Bar")]
    public float chipSpeed = 1;
    public int maxHealth = 2;
    public Image frontHealtBar;
    public Image backHealthBar;
```

```
    [Header("Damage Overlay")]
    public Image overlay; //overlay objekt
    public float duration;
    public float fadeSpeed;
    public AudioSource playerHitSound;

    private float durationTimer;

    void Start()
    {
        health = maxHealth;
    overlay.color = new Color(overlay.color.r, overlay.color.g,
overlay.color.b, 0);
    }
    void Update()
    {
        health = Mathf.Clamp(health, 0, maxHealth);
        UpdateHealthUI();
        if(overlay.color.a > 0)
        {
            if(health <= 1)
            {
                SceneManager.LoadScene(5);
            }
            durationTimer += Time.deltaTime;
            if(durationTimer > duration)
            {
                float tempAlpha = overlay.color.a;
                tempAlpha -= Time.deltaTime * fadeSpeed;
                overlay.color = new Color(overlay.color.r, overlay.color.g,
overlay.color.b, tempAlpha); //za svaki tick updejta se u alpha channel
dodaje slika sa manjim alphom
            }
        }
    }
    public void UpdateHealthUI()
    {
        //Debug.Log(health);
        float fillF = frontHealtBar.fillAmount;
        float fillB = backHealthBar.fillAmount;
        float hFraction = health / maxHealth;
```

```
        if(fillB > hFraction)
        {
            frontHealtBar.fillAmount = hFraction;
            backHealthBar.color = Color.red;
            lerpTimer += Time.deltaTime;
            float percentComplete = lerpTimer / chipSpeed;
            percentComplete = percentComplete * percentComplete;
     backHealthBar.fillAmount = Mathf.Lerp(fillB, hFraction,
percentComplete);
        }
        if(fillF < hFraction)
        {
            backHealthBar.color = Color.green;
            backHealthBar.fillAmount = hFraction;
            lerpTimer += Time.deltaTime;
            float percentComplete = lerpTimer / chipSpeed;
            percentComplete = percentComplete * percentComplete;
      frontHealtBar.fillAmount = Mathf.Lerp(fillF,
backHealthBar.fillAmount, percentComplete);
        }
    }

    public void TakeDamage(float damage)
    {
        health -= damage;
        playerHitSound.Play();
        lerpTimer = 0f;
        durationTimer = 0;
        overlay.color = new Color(overlay.color.r, overlay.color.g,
overlay.color.b, 1);
        if(health < 0f)
        {
            SceneManager.LoadScene(5);
        }
    }

    public void RestoreHealth(float recovery)
    {
        health += recovery;
        lerpTimer = 0f;
    }
```

```
}
```

Ova skripta je preuzeta od “#3 Player Health & Damage Effects: Let’s make a First Person Game in Unity!, Natty GameDev, 28.04.2022.” i “How to make a Better Health Bar in Unity : Chip Away Tutorial, Natty GameDev, 15.09.2020”. PlayerHealth skripta je zadužena za praćenje trenutnih životnih bodova objekta igrača. Ovom skriptom su povezani također vizualni elementi koji vizualno pokazuju koliko je je ostalo igraču životnih bodova i povezana je vizualna obavijest o primljenom oštećenju. Varijable su poprilično objašnjene po samom imenu, ali varijable koje u svojem imenu imaju “Lerp” koriste interpolaciju pri animaciji. U update metodi se provjerava zdravlje igrača i ažurira vizualne elemente. Kada se primi oštećenje, provodi se interpolacija koja smanjuje vidljivost elementa za oštećenje. Ostale metode rade što im ime kaže.

1.1.23. PlayerLook

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

public class PlayerLook : MonoBehaviour
{
    public Camera cam;
    private float xRotation = 0f;

    public float xSensitivity = 30f;
    public float ySensitivity = 30f;

    private bool isCursorLocked = true;

    private void Start()
    {
        LockCursor();
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            isCursorLocked = !isCursorLocked;
            if (isCursorLocked)
```

```
            {
                LockCursor();
            }
            else
            {
                UnlockCursor();
            }
        }
        float mouseX = Input.GetAxis("Mouse X") * xSensitivity *
Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * ySensitivity *
Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -80f, 80f);

        cam.transform.localRotation = Quaternion.Euler(xRotation, 0, 0);
        transform.Rotate(Vector3.up * mouseX);
    }

    private void LockCursor()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    private void UnlockCursor()
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
```

Ova skripta je preuzeta od “#1 FPS Movement: Let’s make a First Person Game in Unity!, Natty GameDev, 08.12.2021.”. Skripta upravlja pogledom igrača, omogućujući okretanje pogleda preko pomaka miša. Računanje pomaka miša se gleda preko računanja mouseX i mouseY pomnoženo s osjetljivošću i vremenom. Da se miš otključa pritisne se tipka “escape”.

5.1.5. PlayerUI

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

```
using TMPro;

public class PlayerUI : MonoBehaviour
{
    [SerializeField]
    private TextMeshProUGUI promptText;
    // Start is called before the first frame update
    void Start()
    {
    }
    // Update is called once per frame
    public void UpdateText(string promptMessage)
    {
        promptText.text = promptMessage;
    }
}
```

Skripta preuzeta od “How to make a Better Health Bar in Unity : Chip Away Tutorial, Natty GameDev, 15.09.2020”. Ova skripta upravlja korisničkim sučeljem igrača. Skripta se koristi za prikazivanje obavijesti igraču na ekranu. Na poziv UpdateText() funkcije na igračevom ekranu će se prikazati odgovarajuća poruka.

11.1. Skripte za interakciju

1.1.24. Interactable

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public abstract class Interactable : MonoBehaviour
{
    public bool useEvents;
    public string promptMessage;

    public void BaseInteract()
    {
       if(useEvents) GetComponent<InteractionEvent>().OnInteract.Invoke();
        Interact();
    }
    protected virtual void Interact()
    {
```

```
    }
}
```

Skripta preuzeta od “#2 FPS Raycast Interactions: Let’s Make a First Person Game in Unity!, Natty GameDev, 22.02.2022.”. Ova skripta služi kao apstrakna klasa koja služi kao osnovna klasa za sve elemente ili objekte s kojima igrač može imati interakciju. Interact() se nasljeđuje, i zatim se stvaraju različite vrste interakcije. UseEvents() omogućuje kontrolu korištenja događaja za specifične objekte.

1.1.25. PlayerInteract

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

public class PlayerInteract : MonoBehaviour
{
    private Camera cam;
    [SerializeField]
    private float distance = 3f;
    [SerializeField]
    private LayerMask mask;
    private PlayerUI playerUI;
    private InputManager inputManager;

    void Start()
    {
        cam = GetComponent<PlayerLook>().cam;
        playerUI = GetComponent<PlayerUI>();
        inputManager = GetComponent<InputManager>();
    }

    void Update()
    {
        playerUI.UpdateText(string.Empty);
        Ray ray = new Ray(cam.transform.position, cam.transform.forward);
        Debug.DrawRay(ray.origin, ray.direction * distance);
        RaycastHit hitInfo;
        if (Physics.Raycast(ray, out hitInfo, distance, mask))
        {
            if(hitInfo.collider.GetComponent<Interactable>() != null)
```

```
            {
      Interactable interactable =
hitInfo.collider.GetComponent<Interactable>();
                playerUI.UpdateText(interactable.promptMessage);
                if (inputManager.onFoot.Interact.triggered)
                {
                    interactable.BaseInteract();
                }
            }
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Weapon"))
        {
            Debug.Log("Dotaknut weapon");
            Destroy(other.gameObject);
        }
    }
}
```

Skripta preuzeta od “#2 FPS Raycast Interactions: Let’s Make a First Person Game in Unity!, Natty GameDev, 22.02.2022.”. Skripta upravlja interakcijom igrača s objektima u igri i omogućuje interakciju igrača s objektima. Također omogućuje uništavanje objekta sa posebnom oznakom. Varijabla “mask” definira koje sve “slojeve” treba uzeti u obzir kada Raycast provjerava objekte za interakciju. RaycastHit služi za provjeru sudara koristeći Physics.Raycast(). OnTriggerEnter() funkcija služi za otkrivanje sudara dva kolidera objekta.

1.1.26. Keypad

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Keypad : Interactable
{
    [SerializeField]
    private GameObject door;
    private bool doorOpen;
    protected override void Interact()
    {
        doorOpen = !doorOpen;
        door.GetComponent<Animator>().SetBool("IsOpen", doorOpen);
```

```
    }
}
```

Skripta preuzeta od “#2 FPS Raycast Interactions: Let’s Make a First Person Game in Unity!, Natty GameDev, 22.02.2022.”. Skripta služi za otvaranje vrata unutar igre. Kada igrač odradi interakciju s gumbom, upalit će se se “Animator” koji će odraditi svoju za otvaranje vrata.

12.1. Neprijatelji

1.1.27. Enemy

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class Enemy : MonoBehaviour
{
    private StateMachine stateMachine;
    private NavMeshAgent agent;
    private GameObject player;
    private Vector3 lastKnownPos;

    public NavMeshAgent Agent { get => agent; }
    public GameObject Player { get => player; }
    public Vector3 LastKnownPos { get => lastKnownPos; set => lastKnownPos
= value; }

    public Path path;
    [Header("Vrijednosti za gledanje")]
    public float sightDistance = 200f;
    public float fieldOfView = 360f;
    public float eyeHeight;
    [Header("Vrijednosti za pucanje")]
    public Transform izUsta;
    [Range(0.1f, 10f)]
    public float fireRate;

    public event System.Action OnDeath;
    private WaveSpawner waveSpawner;

    [SerializeField]
```

```
    private string currentState;
    void Start()
    {
        waveSpawner = GetComponent<WaveSpawner>();
        stateMachine = GetComponent<StateMachine>();
        agent = GetComponent<NavMeshAgent>();
        stateMachine.Initialise();
        player = GameObject.FindWithTag("Player"); //costly funkcija
    }

    void Update()
    {
        CanSeePlayer();
        currentState = stateMachine.activeState.ToString();
    }
    public bool CanSeePlayer()
    {
        if (player != null)
        {
            if (Vector3.Distance(transform.position,
player.transform.position) < sightDistance)
            {
                Vector3 targetDirection = player.transform.position -
transform.position - (Vector3.up * eyeHeight); //kut igraća
                float angleToPlayer = Vector3.Angle(targetDirection,
transform.forward);
                if (angleToPlayer >= -fieldOfView && angleToPlayer <=
fieldOfView)
                {
                    Ray ray = new Ray(transform.position + (Vector3.up *
eyeHeight), targetDirection);
                    RaycastHit hitInfo = new RaycastHit();
                    if (Physics.Raycast(ray, out hitInfo, sightDistance))
                    {
                        if (hitInfo.transform.gameObject == player)
                        {
                            Debug.DrawRay(ray.origin, ray.direction *
sightDistance);
                            return true;
                        }
                    }
                }
```

```
            }
        }
        return false;
    }
}
```

Skripta preuzeta od “#3 Game AI - Patroling: Let’s Make a First Person Game in Unity!, Natty GameDev, 09.09.2022.”. Ova skripta predstavlja komponentu nepijatelja. Skripta omogućuje da otkrije igrača u blizini i reagira na njegovu prisutnost. Također se prati trenutno stanje neprijatelja i omogućava različita ponašanja neprijatelja tijekom igre. “stateMachine” je objekt za upravljanjem stanja neprijatelja i koristi skriptu “StateMachine” koja mijenja stanja neprijatelja. U Update() se provjerava da li se igrač može vidjeti unutar “CanSeePlayer()” metode i ako vidi “currentState” se mijenja u drugo stanje.

1.1.28. EnemyAI

1.1.1.3. StateMachine

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class StateMachine : MonoBehaviour
{
    public BaseState activeState;
    public void Initialise()
    {
        ChangeState(new PatrolState());
    }
    void Update()
    {
        if(activeState != null)
        {
            activeState.Perform();
        }
    }
    public void ChangeState(BaseState newState)
    {
        if(activeState != null)
        {
            activeState.Exit();
        }
        activeState = newState;
```

```
        if(activeState != null )
        {
            activeState.stateMachine = this;
            activeState.enemy = GetComponent<Enemy>();
            activeState.Enter();
        }
    }
}
```

Skripta preuzeta od “#3 Game AI - Patroling: Let’s Make a First Person Game in Unity!, Natty GameDev, 09.09.2022.” Skripta predstavlja jednostavan sustav stanja koji služi za upravljanje neprijatelja u igri. Skripta omogućuje promjenu ponašanja neprijatelja zavisno o kontekstu igre. Ova skripta pripomaže u upravljanju složenih ponašanjima neprijatelja u igri.

1.1.1.4. SearchState

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.ProBuilder.Shapes;
public class SearchState : BaseState
{
    private float searchTimer;
    private float moveTimer;
    public override void Enter()
    {
        enemy.Agent.SetDestination(enemy.LastKnownPos);
    }
    public override void Perform()
    {
        if (enemy.CanSeePlayer())
        {
            stateMachine.ChangeState(new AttackState());
        }
        if(enemy.Agent.remainingDistance < enemy.Agent.stoppingDistance)
        {
            searchTimer += Time.deltaTime;
            moveTimer += Time.deltaTime;
            if (moveTimer > Random.Range(3, 5))
            {
                enemy.Agent.SetDestination(enemy.transform.position +
(Random.insideUnitSphere * 10));
```

```
                moveTimer = 0;
            }
            if (searchTimer > 10)
            {
                stateMachine.ChangeState(new PatrolState());
            }
        }
    }
    public override void Exit()
    {
     }
}
```

Skripta preuzeta od “#6 Game AI – Search State: Let’s Make a First Person Game in Unity!, Natty GameDev, 21.07.2023.”. Ovo stanje se aktivira kada neprijatelj nije u mogućnosti pronaći neprijatelja, pa se pali stanje “traženja”. Ovo je implementirano da se simulira slučajno kretanje neprijatelja. Ako neprijatelj uspije pronaći igrača neprijatelj se prebacuje u “AttackState” u kojemu napada igrača, a ako ne nađe vraća se u “PatrolState”.

1.1.1.5. PatrolState

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PatrolState : BaseState
{
    public int waypointIndex;
    public float waitTimer = 0f;
    public override void Enter()
    {
    }
    public override void Perform()
    {
        PatrolCycle();
        if (enemy.CanSeePlayer())
        {
            stateMachine.ChangeState(new AttackState());
        }
    }
    public override void Exit()
    {
    }
```

```
    public void PatrolCycle()
    {
        if(enemy.Agent.remainingDistance < 0.2)
        {
            waitTimer += Time.deltaTime;
            if(waitTimer > 1)
            {
                if(waypointIndex < enemy.path.waypoints.Count - 1) {
                    waypointIndex++;
                }
                else
                {
                    waypointIndex = 0;
                }

enemy.Agent.SetDestination(enemy.path.waypoints[waypointIndex].position);
                waitTimer = 0;
            }
        }
    }
}
```

Skripta preuzeta od “#4 Game AI – Patrolling: Let’s Make a First Person Game in Unity!, Natty GameDev, 09.09.2022.”. U ovoj skripti neprijatelj se postavlja u stanje patroliranja. Ovo stanje omogućuje neprijatelju da slijedi točke putanje koje mu se zadaju unutar “Path” skripte. Na svakoj točci pričeka malo vremena i kreće na sljedeću točku. Ako usred patroliranja ugleda na neprijatelja prebacuje se na “AttackState”.

1.1.1.6. BaseState

```
public abstract class BaseState
{
    public Enemy enemy;
    public StateMachine stateMachine;

    public abstract void Enter();
    public abstract void Perform();
    public abstract void Exit();
}
```

Skripta preuzeta od “#4 Game AI – Patrolling: Let’s Make a First Person Game in Unity!, Natty GameDev, 09.09.2022.”. Ovo je apstraktna klasa za različita stanja koja če se koristiti u “StateMachine” skripti. Ovo je zajednički okvir za različite tipove stanja koja će se

implementirati unutar igre. Svako stanje od neprijatelja nasljeđuje ovu bazu i implementira vlastite implementacije za sve metode unutar ove skripte.

1.1.1.7. AttackState

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class AttackState : BaseState
{
    private float moveTimer;
    private float losePlayerTimer;
    private float shootTimer;
    public float bulletSpeed = 30f;
    public override void Enter()
    {
    }
    public override void Exit()
    {
    }
    public override void Perform()
    {
        if (enemy.CanSeePlayer()) //ugledan je player
        {
            losePlayerTimer = 0;
            moveTimer += Time.deltaTime;
            shootTimer += Time.deltaTime;
            enemy.transform.LookAt(enemy.Player.transform);
            if(shootTimer > enemy.fireRate)
            {
                Shoot();
            }
            if(moveTimer > Random.Range(1, 3))
            {
                enemy.Agent.SetDestination(enemy.transform.position +
(Random.insideUnitSphere * 5));
                moveTimer = 0;
            }
            enemy.LastKnownPos = enemy.Player.transform.position;
        }
        else
        {
```

```
            losePlayerTimer += Time.deltaTime;
            if(losePlayerTimer > 2)
            {
                //Prebaci nazad u search state
                stateMachine.ChangeState(new SearchState());
            }
        }
    }

    public void Shoot()
    {
        Transform izUsta = enemy.izUsta;
GameObject bullet = GameObject.Instantiate(Resources.Load("Prefabs/Bullet")
as GameObject, izUsta.position, enemy.transform.rotation);
        UnityEngine.Object.Destroy(bullet, 2.5f);
        Vector3 shootDirection = (enemy.Player.transform.position -
izUsta.transform.position).normalized;
        bullet.GetComponent<Rigidbody>().velocity =
Quaternion.AngleAxis(Random.Range(-3f,3f), Vector3.up) * shootDirection *
bulletSpeed;
        shootTimer = 0;
    }
}
```

Skripta preuzeta od “#5 Game AI – Combat/Attack: Let’s Make a First Person Game in Unity!, Natty GameDev, 08.05.2023.”. Ova skripta daje stanje napada za neprijatelja. Sastoji se od tri ključna elementa. Enter() i Exit () koje su prazne. Perform() koja se poziva kada je neprijatelj u stanju napada, u kojoj se provjerava da li je igrač vidljiv, ako je neprijatelj se okreće prema njemu, zatim ako je na određenoj udaljenosti može li početi pucati u igrača. Također da se neprijatelj malo napravi nepredvidljivim daje mu se opcija da se malo kreće u neku slučajnu točku. Zadnja bitnija metoda je Shoot() koja daje neprijatelju svojstvo pucanja. Uzima se objekt metka iz određene mape.

1.1.29. Target

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem.Android;
public class Target : MonoBehaviour, IDamageable
{
    public float health = 100f;
```

```
    public AudioSource enemyDeath;
    public void Damage(float damage)
    {
        Debug.Log("TARGET HP => " + damage);
        health -= damage;
        if (health <= 0f)
        {
            GetComponent<Enemy>().Die(); // Call the Die method to handle
enemy death
            enemyDeath.Play();
            Destroy(gameObject);
        }
    }
}
```

Skripta preuzeta od “Unity Basic Weapon System Tutorial, Plai, 23.01.2022.”. Skripta omogućuje dodavanje životnih bodova neprijateljima, njihovo uništavanje kada koja se izvodi preko Damage() metode.

1.1.30. EnemyForRunning

```
using TMPro;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.UIElements;
public class EnemyForRunning : MonoBehaviour
{
    public Transform player;
    private NavMeshAgent navMeshAgent;
    public float detectionRange = 10f;
    public float speed = 3f;
    private void Start()
    {
        navMeshAgent = GetComponent<NavMeshAgent>();
        navMeshAgent.speed = speed;
    }
    private void Update()
    {
       float distanceToPlayer = Vector3.Distance(transform.position,
player.position);
        if (distanceToPlayer <= detectionRange)
        {
```

```
            if (navMeshAgent != null && gameObject.activeInHierarchy)
            {
                    navMeshAgent.SetDestination(player.position);
            }
        }
        else
        {
            navMeshAgent.ResetPath();
        }
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            other.GetComponent<PlayerHealth>().TakeDamage(1);
        }
    }
}
```

Skripta preuzeta od “Can I make my enemy run towards the player faster in this code? (Unity), ChronicleEdge, 06.05.2020”. U ovoj skripti se postavlja zadatak neprijatelju koji govori da treba trčati na poziciju igrača. NavMesh omogućuje neprijatelju da može po NavMesh elementima se pomicati. Kada je dotaknuo igrača, preko OnTriggerEnter se gleda prvo da li je dotaknuo objekt koji ima oznaku “Player” i zatim zove metodu od Igrača imenom TakeDamage i igrač gubi jedan životni bod.

1.1.31. Path

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
public class Path : MonoBehaviour
{
    public List<Transform> waypoints;
    [SerializeField]
    private bool alwaysDrawPath;
    [SerializeField]
    private bool drawAsLoop;
    [SerializeField]
```

```
    private bool drawNumbers;
    public Color debugColour = Color.white;
     public void OnDrawGizmos()
    {
        if (alwaysDrawPath)
        {
            DrawPath();
        }
    }
    public void DrawPath()
    {
        for (int i = 0; i < waypoints.Count; i++)
        {
            GUIStyle labelStyle = new GUIStyle();
            labelStyle.fontSize = 30;
            labelStyle.normal.textColor = debugColour;
         if (drawNumbers) Handles.Label(waypoints[i].position,
i.ToString(), labelStyle);
            if (i >= 1)
            {
                Gizmos.color = debugColour;
      Gizmos.DrawLine(waypoints[i - 1].position, waypoints[i].position);
                if (drawAsLoop) Gizmos.DrawLine(waypoints[waypoints.Count -
1].position, waypoints[0].position);
            }
        }
    }
    public void OnDrawGizmosSelected()
    {
        if (alwaysDrawPath)
            return;
        else
            DrawPath();
    }
}
```

Skripta preuzeta od “#4 Game AI – Patrolling: Let’s Make a First Person Game in Unity!, Natty GameDev, 09.09.2022.”. Skripta omogućuje stvaranja puteva za neprijatelje koji su unutar “PatrolState”. Unutar skripte se stvara lista u koju se zadaju lokacije koje zatim prati neprijatelj. Sve varijable koje u imenu imaju “draw” služe samo da se unutar scene mogu lakše vidjeti linije i točke za patrolne linije.

13.1. Oružje

1.1.32. GunData

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[CreateAssetMenu(fileName ="Gun", menuName ="Weapon/Gun")]
public class GunData : ScriptableObject
{
    [Header("Gun Info")]
    public new string name;
    public float damage;
    public float maxDistance;
    public float fireRate;
    public float fireSpeed;
}
```

Skripta preuzeta od “Unity Basic Weapon System Tutorial, Plai, 23.01.2022.”. Skripta predstavlja ScriptableObject (hrv. Objekt koji se može skriptirati) koja sadrži konfiguracijske podatke za oružje. “CreateAssetMenu” atribut koji će postati stvaratlje objekta u Unity editoru. Skripta se koristi kao predložak za različite vrste oružja unutar igre. Ovdje se može namjestiti ime, snaga, udaljenost, brzina pucanja, i brzina metka.

1.1.33. Gun

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Gun : MonoBehaviour
{
    [Header("Reference")]
    [SerializeField] private GunData gunData;
    [SerializeField] private Transform muzzle;
    [SerializeField] private bool AddBulletSpread = true;
    [SerializeField] private Vector3 BulletSpreadVariance = new
Vector3(0.1f, 0.1f, 0.1f);
    [SerializeField] private ParticleSystem ImpactParticleSystem;
    [SerializeField] private TrailRenderer BulletTrail;
    [SerializeField] private LayerMask Mask;
```

```
    private float timeSinceLastShot = 0f;
    private float timeBetweenShots;

    private void Start()
    {
        PlayerShoot.shootInput += Shoot;
        CalculateTimeBetweenShots();
    }

    private void CalculateTimeBetweenShots()
    {
        timeBetweenShots = 60f / gunData.fireRate;
    }

    public void Shoot()
    {
        if (Time.time >= timeSinceLastShot + timeBetweenShots)
        {
            if (Physics.Raycast(muzzle.position, transform.forward, out
RaycastHit hitInfo, gunData.maxDistance, Mask))
            {
            TrailRenderer trail = Instantiate(BulletTrail, muzzle.position,
Quaternion.identity);
                StartCoroutine(SpawnTrail(trail, hitInfo));
                Destroy(trail.gameObject, 2.5f);
  IDamageable damageable = hitInfo.transform.GetComponent<IDamageable>();
                damageable?.Damage(gunData.damage);
            }
            timeSinceLastShot = Time.time;
            OnGunShot();
        }
    }
    private Vector3 GetDirection()
    {
        Vector3 direction = transform.forward;
        if (AddBulletSpread)
        {
            direction += new Vector3(
                Random.Range(-BulletSpreadVariance.x,
BulletSpreadVariance.x),
                Random.Range(-BulletSpreadVariance.y,
BulletSpreadVariance.y),
```

```
                Random.Range(-BulletSpreadVariance.z,
BulletSpreadVariance.z)
                );
            direction.Normalize();
        }
        return direction;
    }

    private IEnumerator SpawnTrail(TrailRenderer Trail, RaycastHit Hit)
    {
        float time = 0;
        Vector3 startPosition = Trail.transform.position;
        while (time < 1)
        {
            Trail.transform.position = Vector3.Lerp(startPosition,
Hit.point, time);
            time += Time.deltaTime / Trail.time;
            yield return null;
        }
        Trail.transform.position = Hit.point;
        Instantiate(ImpactParticleSystem, Hit.point,
Quaternion.LookRotation(Hit.normal));
        Destroy(Trail.gameObject, Trail.time);
    }
    private void OnGunShot()
    {
        GetDirection();
    }
}
```

Skripta preuzeta od “Unity Basic Weapon System Tutorial, Plai, 23.01.2022.”. Skripta predstavlja sve karakteristike oružja i dodaju se skripte preko kojih je moguće raditi radnje za uništavanje neprijatelja. Također je postavljena skripta za vizualno predstavljanje ispaljivanja metaka koja je preuzeta od “Hitscan Guns with Bullet Tracers | Raycast Shooting Unity Tutorial, LlamAcademy, 30.12.2021.” pomoću kojih se stavlja vizualni dio na puškama. To su čestice preko kojih izgleda kao da metci izlaze iz puške. Skripta također izračunava koga se pogodilo, direkcija u kojoj se puca i šta će se desiti kad metak pogodi nešto.

1.1.34. PickupWeapon

```
using UnityEngine;
public class PickupWeapon : MonoBehaviour
```

```
{
    public GameObject realGun;
    public GameObject fakeGun;
    public AudioSource gunPickupSound;
    private void OnTriggerEnter(Collider other)
    {
        WeaponState.Instance.SwitchWeapon(realGun);
        gunPickupSound.Play();
    }
}
```

Skripta preuzeta od “How To Make An FPS WOLFENSTEIN 3D Game Unity Tutorial 006 – PICKING UP A WEAPON, Jimmy Vegas, 05.10.2018.”. Skripta služi za interakciju s oružjem kada igrač određenom objektu koje je postavljeno kao oružje. Skripta služi da na igraču se upali oružje koje se nalazi kao dječji objekt, a isključuje lažno oružje.

1.1.35. WeaponState

```
using UnityEngine;

public class WeaponState : MonoBehaviour
{
    public static WeaponState Instance;
    public GameObject currentWeapon;
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }
    public void SwitchWeapon(GameObject newWeapon)
    {
        if (currentWeapon == null)
        {
            currentWeapon = newWeapon;
            newWeapon.SetActive(true);
```

```
        }
    }
}
```

Skripta preuzeta od “How To Make An FPS WOLFENSTEIN 3D Game Unity Tutorial 006 – PICKING UP A WEAPON, Jimmy Vegas, 05.10.2018.”. Skripta upravlja stanjem oružja u igri i omogućuje mijenjanje oružja koje je trenutno u ruci.

1.1.36. PlayerShoot

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

public class PlayerShoot : MonoBehaviour
{
    public static Action shootInput;
    private bool isShooting = false;
    public GameObject hitMarker;
    public float distance;
    public AudioSource hitMarkerHit;
    private bool isHitMarkerActive = false;
    void Start()
    {
        hitMarker.SetActive(false);
    }
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            isShooting = true;
        }

        if (Input.GetMouseButtonUp(0))
        {
            isShooting = false;
        }

        if (isShooting)
        {
            shootInput?.Invoke();
```

```
            Shoot();
        }
    }
    private void Shoot()
    {
        RaycastHit hit;

        if (Physics.Raycast(Camera.main.transform.position,
Camera.main.transform.forward, out hit, distance))
        {
            if (hit.collider.tag == "Enemy")
            {
                if (!isHitMarkerActive)
                {
                    StartCoroutine(ShowHitMarker());
                }
            }
        }
    }
    private IEnumerator ShowHitMarker()
    {
        isHitMarkerActive = true;
        hitMarker.SetActive(true);
        hitMarkerHit.Play();

        yield return new WaitForSeconds(0.2f);

        hitMarker.SetActive(false);
        isHitMarkerActive = false;
    }
}
```

Skripta preuzeta od “Unity Basic Weapon System Tutorial, Plai, 23.01.2022.”.

Skripta je odgovorna za ispuštanje metaka kada igrač stisne lijevi klik miša i da prestane pucati kada je pustio lijevi klik miša, također se pali skripta za prikaz sličice koja govori da se pogodio neprijatelj. Unijeto je također da se izračunava koliko brzo da se metci ispaljuju ako se drži lijevi klik.

14.1. Spawner

1.1.37. EnemyForWaves

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyForWaves : MonoBehaviour
{
    private WaveSpawner waveSpawner;
    private bool hasDecremented = false;

    private void Start()
    {
        waveSpawner = GetComponentInParent<WaveSpawner>();
        hasDecremented = false;
    }
    private void Update()
    {
        if (waveSpawner == null || hasDecremented)
        {
            return;
        }

        if (waveSpawner.currentWaveIndex >= waveSpawner.waves.Length)
        {
            return;
        }

        if (waveSpawner.currentWaveIndex >= 0 &&
waveSpawner.currentWaveIndex < waveSpawner.waves.Length)
        {
            waveSpawner.waves[waveSpawner.currentWaveIndex].enemiesLeft--;
            hasDecremented = true;
        }
    }
}
```

Skripta preuzeta od “Unity Tutorial: How to Create a Wave Spawner, Codecodile, 11.01.2023.”. Skripta koje je odgovorna za praćenje koliko je neprijatelja preostalo u trenutnom valu generiranja neprijatelja. Skripta je zaslužna za odgovarajuće praćenje preostalih neprijatelja u valu.

1.1.38. WaveSpawner

```
using System.Collections;
```

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

public class WaveSpawner : MonoBehaviour
{
    [SerializeField] private float countdown;
    [SerializeField] private GameObject spawnPoint;
    [SerializeField] private float minXSpawn = -10f;
    [SerializeField] private float maxXSpawn = 10f;
    [SerializeField] private float minZSpawn = 0f;
    [SerializeField] private float maxZSpawn = 0f;
    public Wave[] waves;
    public int currentWaveIndex = 0;
    private bool readyToCountDown;
    [SerializeField] private int remainingEnemiesInWave;

    private void Start()
    {
        readyToCountDown = true;
        currentWaveIndex = 0;
        for (int i = 0; i < waves.Length; i++)
        {
            waves[i].enemiesLeft = waves[i].enemies.Length;
        }
        SetRemainingEnemiesInWave();
    }

    private void Update()
    {
        if (countdown <= 0)
        {
            readyToCountDown = false;
            currentWaveIndex++;
            if (currentWaveIndex + 1 >= waves.Length)
            {
                Debug.Log("Finished Level");

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
            }
            else
```

```
            {
                countdown = waves[currentWaveIndex].timeToNextWave;
                StartCoroutine(SpawnWave());
            }
        }
        if (readyToCountDown)
        {
            countdown -= Time.deltaTime;
        }
    }
    private IEnumerator SpawnWave()
    {
        if (currentWaveIndex < waves.Length)
        {
            remainingEnemiesInWave =
waves[currentWaveIndex].enemies.Length;
            for (int i = 0; i < waves[currentWaveIndex].enemies.Length;
i++)
            {
                float randomX = Random.Range(minXSpawn, maxXSpawn);
                float randomZ = Random.Range(minZSpawn, maxZSpawn);
                Vector3 spawnPosition = spawnPoint.transform.position + new
Vector3(randomX, 0f, randomZ);
                Enemy enemy =
Instantiate(waves[currentWaveIndex].enemies[i], spawnPosition,
Quaternion.identity);
                enemy.transform.SetParent(spawnPoint.transform);
                enemy.OnDeath += HandleEnemyDeath;
                yield return new
WaitForSeconds(waves[currentWaveIndex].timeToNextEnemy);
            }
        }
    }
    private void SetRemainingEnemiesInWave()
    {
        if (currentWaveIndex >= 0 && currentWaveIndex < waves.Length)
        {
            remainingEnemiesInWave =
waves[currentWaveIndex].enemies.Length;
        }
    }


    private void HandleEnemyDeath()
```

```
    {
        remainingEnemiesInWave--;
        if (remainingEnemiesInWave <= 0 && currentWaveIndex < waves.Length
- 1)
        {
            readyToCountDown = true;
        }
    }
}
[System.Serializable]
public class Wave
{
    public Enemy[] enemies;
    public float timeToNextEnemy;
    public float timeToNextWave;
    public int enemiesLeft;
}
```

Skripta preuzeta od “Unity Tutorial: How to Create a Wave Spawner, Codecodile, 11.01.2023.”. Skripta omogućuje generiranja valova neprijatelja u igrici. U njoj se nalaze brojači za smanjivanje vremena preostalog za sljedeći val, brojač za preostale neprijatelje, brojač za razliku generiranja sljedećeg neprijatelja, poziciju gdje će se generirati neprijatelji, itd. Također je logika postavljena da kada se unište svi neprijatelji da se može otići na sljedeći level.

1.1.39. EnemySpawnerEasier

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemySpawnerEasier : MonoBehaviour
{
    public GameObject objectToActivate;
    public float delayInSeconds = 2f;
    private void Start()
    {
        objectToActivate.SetActive(false);
        Invoke("ActivateObject", delayInSeconds);
```

```
    }
    private void ActivateObject()
    {
        objectToActivate.SetActive(true);
    }
}
```

Skripta napravljena pomoću online alata. Jednostavnija skripta za generiranje neprijatelja koja kada je na neprijatelju generira ga nakon određenog broja sekundi.

15.1. Neprijateljski šef

1.1.40. EnemyHeart

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyHeart : MonoBehaviour, IDamageable
{
    public float health = 10f;
    public GameObject e;
    public AudioSource heartDestroyed;

    public void Damage(float damage)
    {
        health -= damage;
        if (health <= 0f)
        {
            ParentOfTheEnemy.destroyedHearts += 1;
            heartDestroyed.Play();
            Destroy(e);
        }
    }
}
```

Skripta napravljena preko prijašnjih skripti i skupljenog znanja i uz pomoć interneta. Skripta predstavlja objekt koji se može uništiti i nakon što se uništi zove metodu “ParentofTheEnemy” i oduzima u dodaje toj skripti plus jedan na “destroyedHearts” varijabli, nakon što je dodala uništava taj objekt.

1.1.41. ParentOfTheEnemy

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class ParentOfTheEnemy : MonoBehaviour
{
    public static int destroyedHearts = 0;
    public void Update()
    {
        if (destroyedHearts == 3)
        {
            SceneManager.LoadScene(4);
        }
    }
}
```

Skripta napravljena preko “How to add a score counter into your Unity 2D game| Easy Unity 2D Tutorial, Alexander Zotov, 14.04.2017.”. Skripta je nastavak skripte “EnemyHeart”, ukojoj se uništavaju objekti. Kada su se uništili tri objekta skripta šalje na sljedeću scenu.

16.1. 2D skripte

1.1.42. StartMenu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class StartMenu : MonoBehaviour
{
    public void StartGame()
    {
        SceneManager.LoadScene(1);
    }
```

```
    public void RestartGame()
    {
        SceneManager.LoadScene(1);
    }
    public void MenuScreen()
    {
        SceneManager.LoadScene(0);
    }
}
```

Skripta napravljena preko “Start & Game Over Screen | Build Your First 3D Game in Unity #10, Coding in Flow, 11.07.2021.”. U skripti se samo nalaze funkcije koje služe za prebacivanje scena.

1.1.43. EndMenu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EndMenu : MonoBehaviour
{
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

Skripta napravljena preko “Start & Game Over Screen | Build Your First 3D Game in Unity #10, Coding in Flow, 11.07.2021.”. U skripti se samo nalazi funkcija koja gasi igricu.

6. Zaključak

Unity je odličan početnički alat za svakog programera ili osobe koja želi početi raditi igrice. Na internetu postoji mnogo alata koji su dostupni besplatno da pomognu pri izgradnji prve igrice. Bitno je da je osoba predana tome da će završiti igricu i dobro promisliti da li je to put za tu osobu. Prije nego što se uopće krene raditi igrica pametno je naučiti programski jezik u kojemu će se skripte za igricu pisati. Bitno je isto imati neku viziju u glavi kakvu igrica će se htjeti izgraditi da postoji glavni cilj i da se može znati kada je osoba zadovoljna sa zadnjim produktom.

Za radnju igrice nisu potrebne knjige u današnje vrijeme jer svi alati su dostupni na internetu preko kojih se uči. U ovome radu sam naučio da iako skripte se mogu prepisati, uvijek će doći do problema koji možda nisu prisutni u lekcijama iz kojih se prepisuje ili uzima ideja.

U drugu ruku Blender je vrlo kompleksan program koji ne bi preporučio apsolutnim početnicima zbog visoke razine potrebnog znanja koje je potrebno da se shvati, što sve dostupne opcije rade i zašto. Najduže vremena sam proveo na ovome programu i samo shvaćanja kako funkcioniraju kosti, kako ih postaviti i nakon toga kako stvoriti animaciju. Postoje par “Youtube” videa koji imaju početnike na umu i preko kojih se objasni samo kako stvoriti neki oblik, kako mu staviti kosti i kako napraviti par jednostavnih animacija.

Igricu je bilo vrelo zabavno raditi iako kao i u bilo kojem programiranju se dođe do nekog “zida”, gdje osoba misli da ne će nikada riješiti, ali kada dođe do onog trenutka gdje kada se riješi kao da se dobilo neko novo znanje o programiranju.

Na kraju kada je napisan rad, predao sam ga prijatelju kojeg sam pitao da ga pročita i da mi kaže šta misli o njemu i šta misli da li bi on mogao započeti raditi svoju igricu iako nikada nije programirao. Dobio sam poruku da mu je rad bio zanimljiv, razumio je koja je poanta ovoga rada. Napisao je isto da sad ide krenuti raditi svoju igricu i da ima ovaj rad pored sebe da bi ga mogao pratiti i da je razumio osnovne koncepte. To je po meni cijeli uspijeh ovog rada i zadovoljan sam kako je ispao.

7. Popis literature

1. How to use Unity NavMesh Pathfinding! (Unity Tutorial), Code Monkey, 14.07.2021. https://www.youtube.com/watch?v=atCOd4o7tG4&ab_channel=CodeMonkey

2. How To Make An FPS WOLFENSTEIN 3D Game Unity Tutorial 006 – PICKING UP A WEAPON, Jimmy Vegas, 05.10.2018. https://www.youtube.com/watch?v=kmd_nY-teLk&list=PLZ1b66Z1KFKh51c-tMSxc3ozDsAOLWyhC&index=8&ab_channel=JimmyVegas

3. Unity Basic Weapon System Tutorial, Plai, 23.01.2022. https://www.youtube.com/watch?v=kXbQMhwj5Uc&ab_channel=Plai

4. Gunpickup – Sound Effect, BoostSound, 16.07.2023. https://www.youtube.com/watch?v=mpXg_GeABI8&ab_channel=BoostSound

5. Unity Tutorial: How to Create a Wave Spawner, Codecodile, 11.01.2023. https://www.youtube.com/watch?v=duo45NjwZ78&ab_channel=Codecodile

6. Hitscan Guns with Bullet Tracers | Raycast Shooting Unity Tutorial, LIamAcademy, 30.12.2021. https://www.youtube.com/watch?v=cI3E7_f74MA&ab_channel=LIamAcademy

7. Let's Make a First Person Game in Unity!, 21.07.2023. https://www.youtube.com/playlist?list=PLGUw8UNswJEOv8c5ZcoHarbON6mIEUFBC

8. See how easily you can make an hitmarker for an fps game in Unity with this video, FPS Builders, 28.09.2020. https://www.youtube.com/watch?v=mlHM5wEsBpM&ab_channel=FPSBuilders

9. Start & Game Over Screen | Build Your First 3D Game in Unity #10, Coding in Flow, 11.07.2021. https://www.youtube.com/watch?v=1Ye-mCuIIdw&ab_channel=CodinginFlow

10. Roblox Death Sound (Oof) - Sound Effect (HD), Gaming Sound FX, 25.12.2017., https://www.youtube.com/watch?v=3w-2gUSus34&ab_channel=GamingSoundFX

11. Can I make my enemy run towards the player faster in this code? (Unity), ChronicleEdge, 06.05.2020. https://stackoverflow.com/questions/61633360/can-i-make-my-enemy-run-towards-the-player-faster-in-this-code-unity

12. About, Bez podatka, https://www.blender.org/about/

13. Editors, 09.15.2023., https://docs.blender.org/manual/en/latest/editors/index.html

14. Cursor.lockState, 08.09.2023., https://docs.unity3d.com/ScriptReference/Cursor-lockState.html

8. Popis slika

Slika 1 Unity Hub (Izvor: Unity Hub, 2023.) 3
Slika 2 Kreiranje projekta (Izvor: Unity Hub, 2023.) 4
Slika 3 Izgled početne scene (Izvor: Unity scene editor, 2023.) 4
Slika 4 Tools (Alati) (Izvor: Unity “Scene” prozor, 2023.) 5
Slika 5 Hijerarhija (Izvor: Unity “Scene”, 2023.) 6
Slika 6 Projektna mapa (Izvor: Unity “Scene”, 2023.) 7
Slika 7 Inspektor (Izvor: Unity “Scene”, 2023.) 8
Slika 8 Game prozor (Izvor: Unity ”Scene”, 2023.) 9
Slika 9 Unity Asset store (Izvor: https://assetstore.unity.com, 2023.) 10
Slika 10 Package Manager (Izvor: Unity aplikacija, 2023.) 10
Slika 11 Import Unity Package (Izvor: Unity aplikacija, 2023.) 11
Slika 12 Probuilder prozor i kreator (Izvor: Unity aplikacija, 2023) 12
Slika 13 Input Action prozor (Izvor: Unity aplikacija, 2023.) 13
Slika 14 Namještanje kamere (Izvor: Unity aplikacija, 2023.) 14
Slika 15 Input actions za igrača (Izvor: Unity aplikacija, Input Actions, 2023.) 15
Slika 16 Izgled objekta sa ubačenim skriptama (Izvor: Unity aplikacija, Inspector, 2023.) 17
Slika 17 Gotova arena (Izvor: Unity aplikacija, 2023.) 18
Slika 18 Nivo s gornje strane (Izvor: Unity aplikacija, scena, 2023.) 19
Slika 19 Zatvorena vrata (Izvor: Unity aplikacija, scena, 2023.) 20
Slika 20 Otvorena vrata (Izvor: Unity aplikacija, scena, 2023.) 20
Slika 21 Tekst unutar scene (Izvor: Unity aplikacija, scena, 2023.) 21
Slika 22 Animator za vrata (Izvor: Unity aplikacija, animator, 2023.) 22
Slika 23 Objekt neprijatelja (Izvor: Unity aplikacija, 2023.) 23
Slika 24 Prostor za hodanje (Izvor: Unity aplikacija, navigacija, 2023.) 25
Slika 25 Putanja neprijatelja (Izvor: Unity aplikacija, scena, 2023.) 25
Slika 26 Karakteristike oružja (Izvor: Unity aplikacija, inspektor, 2023.) 26
Slika 27 Hijerarhija prvog i lažnog oružja (Izvor: Unity aplikacija, hijerarhija i scena, 2023.) 27
Slika 28 Izgled zdravstvene trake (Izvor: Unity aplikacija, hijerarhija i scena, 2023.) 28
Slika 29 Spawner za neprijatelje (Izvor: Unity aplikacija, hijerarhija i scena, 2023.) 29
Slika 30 Izgled zadnjeg neprijtelja (Izvor: Unity aplikacija, hijerarhija i scena, 2023.) 29
Slika 31 Audio Source komponenta (Izvor: Unity aplikacija, inspektor , 2023.) 30
Slika 32 Blender (Izvor: Blender aplikacija, 2023.) 32
Slika 33 Početni prozor (Izvor: Blender aplikacija, 2023.) 33
Slika 34 Alatna traka (Izvor: Blender aplikacija, 2023.) 33
Slika 35 Hijerarhijski prozor (Izvor: Blender aplikacija) 34
Slika 36 Inspektor za objekt (Izvor: Blender aplikacija, 2023) 35
Slika 37 Početni lik (Izvor: Blender aplikacija, 2023) 36
Slika 38 Kost u liku (Izvor: Blender aplikacija, 2023) 37
Slika 39 Cijeli postavljeni kostur (Izvor: Blender aplikacija, 2023) 38
Slika 40 Objekt sa izrezima (Izvor: Blender aplikacija, 2023) 38
Slika 41 Objekt s dobro postavljenim kosturom (Izvor: Blender aplikacija, 2023) 39
Slika 42 Objekt s dobro postavljenim kosturom (Izvor: Blender aplikacija, 2023) 40
Slika 43 Postavljanje da se animacija ponavlja (Izvor: Unity aplikacija, 2023) 41
Slika 44 Izvođenje animacije (Izvor: Unity aplikacija, 2023) 42

9. Prilozi

https://unity.com/features/probuilder - Preuzeto 01.09.2023.

https://assetstore.unity.com/packages/2d/textures-materials/floors/yughues-free-ground-materials-13001 - Preuzeto 01.09.2023.

https://assetstore.unity.com/packages/2d/textures-materials/world-materials-free-150182 - Preuzeto 01.09.2023.

https://assetstore.unity.com/packages/2d/textures-materials/nature/yughues-free-nature-materials-13237 - Preuzeto 01.09.2023.

https://assetstore.unity.com/packages/2d/textures-materials/nature/2d-gem-pack-by-gamertose-50487 Preuzeto 01.09.2023.

https://images.pngnice.com/download/2007/Crack-Glass-PNG-File.png - Preuzeto 01.09.2023.

Link do gotove igrice -> https://drive.google.com/file/d/1846Kb5t73-5GTcZ7z3LVEpusWM6XB5f9/view?usp=drive_link