Izrada videoigre pucanja iz prvog lica u programskom alatu Unity

Kučan, Ivan

Undergraduate thesis / Završni rad

2022

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:211:004740

Rights / Prava: Attribution-NonCommercial-NoDerivs 3.0 Unported/Imenovanje-Nekomercijalno-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-04-25**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Kučan

IZRADA VIDEOIGRE PUCANJA IZ PRVOG LICA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Kučan

JMBAG: 0016142529

Studij: Informacijski sustavi

IZRADA VIDEOIGRE PUCANJA IZ PRVOG LICA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, rujan 2022.

Ivan Kučan

Izjava o izvornosti

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

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

Sažetak

Tema ovog završnog rada izrada je 3D računalne videoigre pucanja iz prvog lica. U radu se na početku opisuje alat Unity u kojem se igra izrađuje, nakon toga slijedi definicija i opis videoigara, žanrova videoigara te opsežniji opis žanra igre koja se izrađuje. U nastavku je opis elemenata koje igra sadržava poput pucanja, kretanja, razina i korisničkog sučelja. Do kraja rada se detaljno opisuju koraci izrade igre. Taj opis sastoji se od objašnjenja kako su objekti kreirani i veza između objekata, opisa programske logike svakog od elementa igre uz prikaz programskog koda, postupaka za rješavanje određenih problema i prikaza krajnjeg rezultata elemenata pomoću slika.

Ključne riječi: algoritmi, programiranje, računalne igre, unity, videoigra pucanja iz prvog lica

Sadržaj

1. Uvod 1

2. Unity 2

2.1. Sučelje 3

2.2. Unity Asset Store 4

3. Videoigre 6

3.1. Žanrovi 6

3.2. Pucačine (eng. *Shooters*) 7

3.3. Pucanje iz prvog lica (eng. *First person shooter - FPS*) 8

3.3.1. Dizajn igre 10

3.3.2. Borba 10

3.3.3. Dizajn razine igre (eng. *Level design*) 10

3.3.4. Igra za više igrača (eng. *Multiplayer*) 11

4. Opis naše videoigre 12

4.1. Pucanje 12

4.2. Razine 12

4.3. Kretanje 12

4.4. Korisničko sučelje (eng. *User interface – UI*) 13

4.4.1. Glavni izbornik 13

4.4.2. Izbornik za pauzu 13

4.4.3. Izbornik za završetak razine 13

4.4.4. Zaslon s informacijama (eng. *Head-up display – HUD*) 14

5. Izrada igre 15

5.1. Kreiranje projekta 15

5.2. Kreiranje lika 15

5.3. Kretanje igrača 16

5.4. Pogled mišem 18

5.5. Puške i pucanje 20

5.5.1. Promjena puške 20
5.5.2. Problem - „clipping“ puške 21
5.5.3. Efekt pucanja i pogotka 22
5.5.4. Zvuk pucanja i punjenja spremnika 22
5.5.5. Funkcionalnosti puške 23
5.5.6. Animacija trzaja puške 30
5.5.7. Sway i bob 31
5.6. Mete 33
5.7. Izrada razina 34
5.7.1. Materijali 35
5.7.2. Okidači (eng. *trigger*) 36
5.7.3. Osvjetljenje 37
5.7.4. GameManager 38
5.8. Korisničko sučelje (eng. *User interface - UI*) 38
5.8.1. Glavni izbornik (eng. *Main menu*) 38
5.8.2. Izbornik za pauzu (eng. *Pause menu*) 40
5.8.3. Izbornik za završetak razine (eng. *End menu*) 40
5.8.4. Zaslon s informacijama (eng. *Head-up display – HUD*) 41
6. Zaključak 42
Popis literature 43
Popis slika 45

1. Uvod

Razvojem tehnologije raste i industrija videoigara te je u proteklih nekoliko godina iz fokusiranih tržišta izrasla u mainstream. Ta industrija sada je veća od filmske i glazbene industrije zajedno.[1] Rastu industrije pomogla i pandemija zbog koje su ljudi morali biti u svojim kućama pa su zabavu pronašli u igranju videoigara. To potvrđuje i činjenica da se tijekom pandemije tržište povećalo za 26%, na rekordnih 191 milijardu dolara. [2]

Postoje različite podjele videoigara. To može biti podjela prema platformi na kojoj se videoigra koristi poput osobnog računala ili konzole, prema načinu igranja: za jednog igrača ili za više igrača, prema žanru kao što su akcijske igre ili avanture ili prema stilu igranja koji može biti ležeran ili natjecateljski. Prema ovim podjelama, svatko može pronaći neku igru koja mu odgovara.

Da bi se sve te igre napravile, koriste se softveri koji se nazivaju „game engine“. To su softveri za razvoj i pokretanje igara koji nude skup alata i značajki za razvoj igara te se pomoću njih jednostavnije i brže izrađuju igre što je potrebno zbog rasta industrije. Jedan od takvih alata je i Unity, koji smo koristili za praktični dio ovog rada. Detaljima o Unity-u bavit ćemo se u nastavku.

2. Unity

Unity je jedan od najpopularnijih pokretača ili jezgara igre (eng. *game engine*) za 2D i 3D videoigre čiji je razvoj pokrenut još 2005. godine. To potvrđuje i činjenica da je mnogo poznatih igara napravljeno korištenjem Unity-a. Tu spadaju videoigre kao što su Heartstone, Rust, Fall Guys, The Forest, Escape from Tarkov i Raft. Iako se najčešće koristi za izradu videoigara, svoje mjesto našao je i u drugim industrijama zbog snažnih alata za simulacije i animacije, no također je prikladan i za početnike zbog svoje jednostavnosti. Za skriptiranje koristi se programski jezik C# uz Microsoft Visual Studio koji je integriran u Unity. [3]

Dakle, kao pokretač igre, ima ugrađena određena svojstva koja bi mogla biti potrebna osobi koja razvija videoigru kao što su fizika, rasvjeta, detekcija sudara objekata i prikazivanje trodimenzionalnog svijeta. Unity je i višeplatformsko integrirano razvojno okruženje (eng. *Integrated development environment – IDE*) što znači da ima ugrađene sve potrebne alate za izradu videoigre i podržava razne platforme za računala, mobilne uređaje, konzole i virtualnu stvarnost.

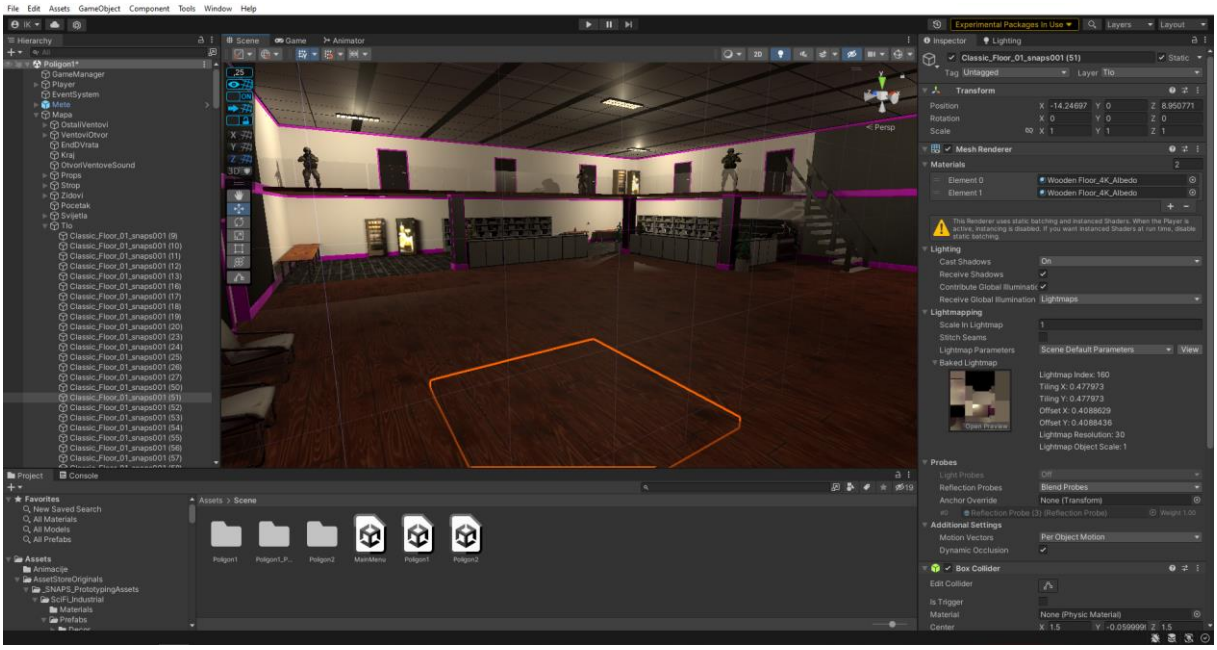
Unity dolazi u tri glavne verzije:

- Personal – besplatna verzija za pojedince i male organizacije s manje od sto tisuća dolara prihoda u zadnjih 12 mjeseci
- Plus – za pojedince koji se ozbiljnije bave izradom videoigara i za manje organizacije s prihodom manjim od dvjesto tisuća dolara u zadnjih 12 mjeseci. Pruža više funkcionalnosti i resurse za vježbu i učenje.
- Pro – za profesionalce s prihodom većim od dvjesto tisuća dolara godišnje. Uključuju dodatne funkcionalnosti i tehničku podršku.

Za potrebe ovog rada koristili smo besplatnu Personal verziju Unity-a.

Slika 1. Unity logo [4]

2.1. Sučelje



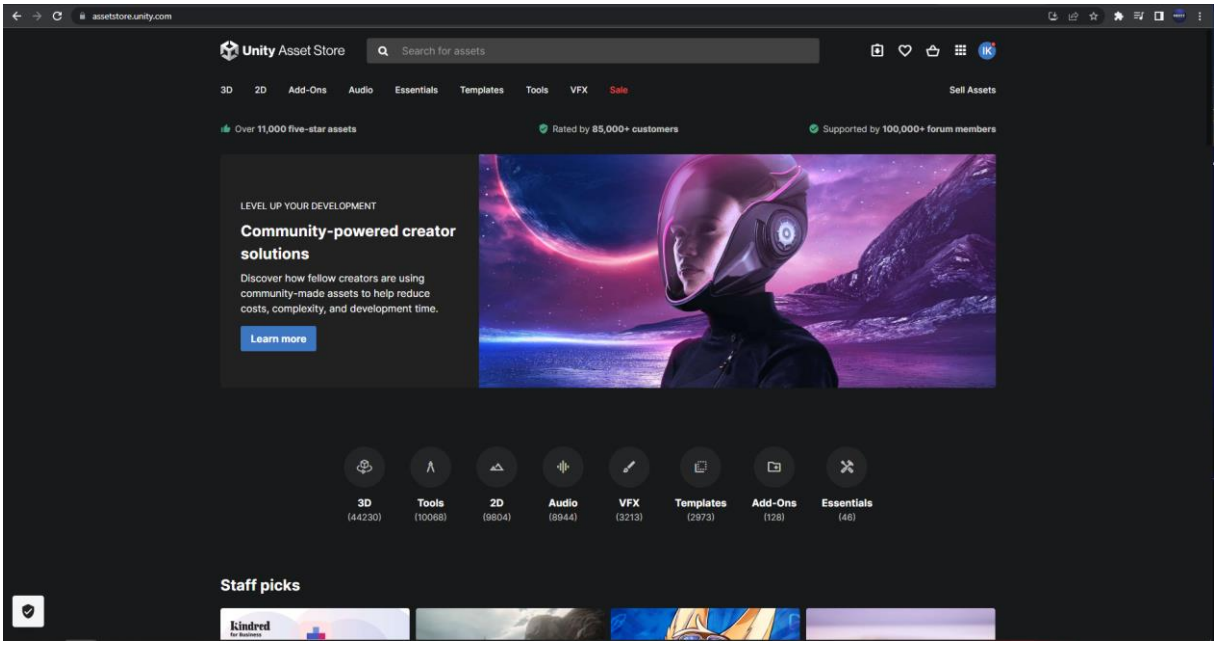
Slika 2. Unity - Sučelje za uređivanje

Na slici 2. vidimo izgled sučelja za uređivanje (eng. *editor interface*) . U prvom je planu interaktivni vizualni alat „Scene view“ u koji možemo dovući objekte i manipulirati njihovim svojstvima te u stvarnom vremenu (eng. *real time*) vidjeti kako će igra izgledati. Možemo se i prebaciti na pregled igre (eng. *Game view*) te će se na tom mjestu pojaviti prozor s igrom koju možemo odmah i testirati. S desne strane možemo manipulirati svojstvima odabranog objekta i dodavati mu nova svojstva poput animacija, događaja, efekata i fizike. Na lijevoj strani je hijerarhija (eng. *Hierarchy*) odnosno popis objekata u kojem je omogućen pregled i odabir svih objekata koji su dodani u scenu. Također, možemo i dodavati različite objekte kao što su 3D objekti različitih oblika, svijetla, izvor zvuka (eng. *Audio source*) ili elemente korisničkog sučelja (*User interface – UI*). S donje strane nalazi se prozor pod nazivom „Project“ u kojem se nalaze sve datoteke koje se mogu koristiti u projektu, bile one korištene ili ne. To je, dakle, preglednik datoteka (eng. *File explorer*) u kojem su datoteke organizirane u mape i iz njega se stvari mogu direktno dovući u „Scene view“ kako bi ih dodali u scenu.

2.2. Unity Asset Store

Prije nego što objasnimo što je „Asset store“ moramo definirati Unity asset. Unity asset je predmet koji možemo koristiti u projektu. [5] To može biti datoteka napravljena izvan samog Unity-a (npr. 3D model, audio zapis, slika ili bilo koji tip datoteke koji je podržan) ili datoteka napravljena u Unity-u kao što je upravljač animatora (eng. *Animator controller*).

Unity Asset Store je rastuća biblioteka datoteka koje kreiraju i objavljuju članovi zajednice kao i sama tvrtka Unity Technologies. Tamo je moguće pronaći razne tipove datoteka od animacija, modela i tekstura pa sve do gotovih cijelih projekata, vodiča za učenje ili dodataka za sučelje za uređivanje. Neke datoteke su besplatne, dok je za druge potrebno platiti određenu cijenu.



Slika 3. Unity Asset Store (assetstore.unity.com)

Kao što vidimo na slici 3., datoteke na Unity Asset Store-u raspoređene su u kategorije:

- 3D – animacije, likovi, okoliš, rekviziti, vozila
- Alati (eng. *Tools*) – alati koji korisniku olakšavaju i unaprjeđuju izradu i upravljanje gotovo svega što se može raditi u Unity-u (npr. verzioniranje, generator animacija, sprječavanje varanja ili automatska zamjena već postavljenih objekata nekim drugim objektom)

- 2D – likovi, okoliš, teksture, materijali, fontovi, UI elementi
- Audio – audio zapisi ambijenta, glazba, audio efekti
- Vizualni efekti (*Visual effects* – VFX) – čestice (eng. *particles*), sjenčari (eng. *shaders*)
- Predlošci (eng. *Templates*) – stvari korisne za početnike kao što su vodiči za učenje i započeti projekti
- Dodaci (eng. *Add-Ons*) – umjetna inteligencija i validatori
- Osnovno (eng. *Essentials*) – unaprijed napravljeni osnovni dijelovi igara poput upravljanja likom ili objekti kao kutija i stup

Kada pronađemo ono što nam je potrebno, kliknemo na gumb za preuzimanje te nam se u Unity-u otvori prozor u kojem se datoteke preuzimaju i uvoze u naš projekt.

3. Videoigre

Prije nego što opišemo žanrove videoigara i smjestimo našu igru u određeni žanr potrebno je definirati što je videoigra. Videoigra ili računalna igra je elektronička igra koja uključuje interakciju korisnika s korisničkim sučeljem ili uređajem za prepoznavanje unosa kao što je joystick, kontroler ili tipkovnica kako bi generirala vizualne povratne informacije koje se prikazuju na TV prijemniku, monitoru ili zaslonu osjetljivom na dodir. Tu su često i zvučne povratne informacije koje se isporučuju putem zvučnika ili slušalica. Nedavno se industrija proširila i na igre za mobilne telefone i tablet računala, sustave virtualne i proširene stvarnosti te igre u oblaku (eng. *remote cloud gaming*). [6]

Videoigre zahtijevaju platformu, specifičnu kombinaciju elektroničkih komponenti ili računalnog hardvera i softvera, kako bi mogle raditi. Igre su obično dizajnirane za igranje na jednoj ili ograničenom broju platformi. Glavna podjela videoigara je prema platformi na kojoj se koriste:

- Igre za osobno računalo (eng. *personal computer games*) – većina videoigara su igre za osobno računalo, no glavna namjena osobnih računala nije samo pokretanje igara, tako da postoje razlike u pokretanju iste igre na različitim računalima. Postoje i računala za igranje (eng. *gaming computer*). Takva računala namijenjena su posebno za igranje pa koriste komponente visokih performansi i visoke cijene.
- Igre za konzole (eng. *console games*) – vrsta videoigre koja se sastoji od slike i zvuka generiranih na konzoli za videoigre, obično prikazani na televizoru ili sličnom audio-video sustavu, kojima igrač može manipulirati. Konzole za videoigre posebno su dizajnirane za igranje igara te koriste određen hardver što razvojnim programerima daje konkretan hardverski cilj što pojednostavljuje razvoj u usporedbi s razvojem igara za osobna računala.
- Arkadne videoigre (eng. *arcade video games*) – većina radi na principu da se umetne novčić ili žeton za koje se dobiva određeno vrijeme za igru ili broj života u igri. Obično se nalaze u zabavnim parkovima.

3.1. Žanrovi

Videoigre su, kao i većina drugih oblika medija, kategorizirane u žanrove. Za razliku od filma ili televizije koji se kategoriziraju prema vizualnim ili narativnim elementima, žanrovi videoigara obično se temelje na načinu interakcije u igri. [6] Nazivi žanrova obično i sami

opisuju način igranja, no nazivi nekih žanrova potječu iz utjecajnih djela koja su definirala taj žanr. Najčešće definirani žanrovi videoigara su:

- Akcijska (eng. *Action*) – naglašeni fizički izazovi koji za svladavanje zahtijevaju koordinaciju očiju i ruku i motoričke vještine. Fokus je na igraču koji kontrolira većinu akcije. Akcijske igre podijeljene su u mnogo podžanrova poput platformskih igara, borbenih igara i pucačina.
- Avantura (eng. *Adventure*) - bez refleksnih izazova i akcije, obično rješavanje raznih zagonetki u interakciji s ljudima ili okolinom bez sukoba.
- Akcijska avantura (eng. *Action-adventure*) – kombiniraju elemente akcijskih i avanturističkih igara. Obično su usmjerene na istraživanje svijeta i skupljanje predmeta pomoću kojih se rješavaju zagonetke za čije prevladavanje su potrebni elementi akcijskih igara.
- Zagonetke (eng. *Puzzle*) – usmjerena na logičke i konceptualne izazove. Rješavanje zagonetki primarni je način igranja.
- Igranje uloga (eng. *Role-playing games - RPG*) – način igranja razvijen iz tradicionalnih igara uloga poput igre Dungeons & Dragons. Igrač je u ulozi lika koji se kroz igranje unaprjeđuje prevladavanjem izazova.
- Simulacija (eng. *Simulation*) – igre dizajnirane da simuliraju aspekte stvarnog ili izmišljenog svijeta. Tu spadaju simulacije vozila, simulacije izgradnje i upravljanja.
- Strategija (eng. *Strategy*) –zahtijeva pažljivo i vješto razmišljanje i planiranje kako bi se postigla pobjeda.
- Sportovi (eng. *Sports*) – videoigre koje simuliraju sportove. Protivnički tim mogu biti drugi ljudi iz stvarnog života ili umjetna inteligencija.
- Mrežna igra za mnogo igrača (eng. *Massively Multiplayer Online – MMO*) – mrežna videoigra koja podržava veliki broj igrača istovremeno. Takve igre mogu se pronaći za većinu mrežnih platformi.

3.2. Pucačine (eng. *Shooters*)

Naša igra spadat će pod pucačine. Već smo spomenuli da su pucačine podžanr akcijskih igara te je u njima fokus na porazu neprijateljevog lika korištenjem oružja. To su obično vatrena oružja koja se mogu koristiti u kombinaciji s drugim alatima kao što su granate. Uobičajeni resurs koji se nalazi u gotovo svim pucačkim igrama je streljivo, oklop ili zdravlje (eng. *health*) i nadogradnje kojima se unaprjeđuje oružje igrača.

Pucačine testiraju igračevu prostornu svijest, reflekse i brzinu kako u izoliranom okruženju za jednog igrača (eng. *singleplayer*) tako i u umreženom za više igrača (eng. *multiplayer*).

Pucačine su također podijeljene u više podžanrova:

- „Shoot 'em up“ – igrač se može kretati gore, dolje, lijevo ili desno po ekranu obično pucajući ravno ispred. Ovu kategoriju definirala je igra Space Invaders.
- Trči i pucaj pucačina (eng. *Run-and-gun shooter*) – 2D akcijske igre s bočnim ili okomitim pomicanjem gledišta.
- Streljana (eng. *Shooting gallery*) – igre u kojima igrač cilja u pokretne mete na nepomičnom zaslonu.
- „Light gun shooter“ – dizajnirane za igranje uz korištenje kontrolera u obliku pištolja.
- Pucanje iz prvog lica (eng. *First person shooter – FPS*) – prikazuju perspektivu igračevog lika na ekranu unutar trodimenzionalnog prostora, pri čemu igrač ima kontrolu i djelovanje nad kretanjem i radnjom lika unutar tog prostora.
- Pucanje iz trećeg lica (eng. *Third person shooter – TPS*) – usko povezana s pucačinom iz prvog lica uz razliku da su lik igrača i njegova okolina vidljivi na ekranu. Primarni fokus pogleda kamere je igračev lik.
- Kraljevska bitka (eng. *Battle royale*) – mrežna videoigra za više igrača u kojima počinju s minimalnom opremom, a cilj je eliminirati sve druge protivnike izbjegavajući ostati zarobljeni izvan sve manjeg sigurnog područja. [7]

3.3. Pucanje iz prvog lica (eng. *First person shooter - FPS*)

Našu igru detaljnije možemo smjestiti kao videoigru pucanja iz prvog lica. To je podžanr pucačkih videoigara usredotočenih na borbu temeljenu na vatrenom ili nekom drugom oružju u perspektivi iz prvog lica. Dakle, igrač akciju doživljava očima lika i kontrolira ga u trodimenzionalnom prostoru. Od nastanka žanra, napredna 3D grafika predstavlja izazov za razvoj hardvera, a s vremenom je način igranja za više igrača (eng. *multiplayer*) postao sastavni dio žanra.

Slika 4. Doom (1993.) [8]

Prvom igrom ovog žanra smatra se Wolfenstein 3D (1992.) koji je zaslužan za stvaranje osnovnog tipa žanra na kojem su se temeljili kasniji naslovi. Jedan takav naslov, zbog kojeg je žanr postao popularan i šire prihvaćen, bio je Doom (1993.). Zbog njegovog utjecaja nekoliko godina se za označavanje ovog žanra koristio izraz „Doom klon“. Jedno od čestih imena za žanr bilo je i „Pucačina u hodniku“ (eng. *Corridor shooter*) jer su ograničenja hardvera tog doba značila da se većina radnje odvijala u zatvorenim prostorima poput hodnika i tunela. [9]

Slika 5. Half-Life [10]

Igre Half-Life (1998.) i njegov nastavak Half-Life 2 (2004.) poboljšale su elemente naracije i zagonetke, a 1999. godine izašla je i modifikacija za Half-Life pod imenom Counter-

Strike koja je zajedno s Doom-om jedna od najutjecajnijih pucačina iz prvog lica. Ta igra, i kasnija verzija Counter-Strike: Source (2004.), postala je najpopularnija modifikacija igre za više igrača ikada, s više od 90000 igrača koji su se natjecali u bilo kojem trenutku tijekom svog vrhunca. [9]

3.3.1. Dizajn igre

Kao i većina pucačina, pucačine iz prvog lica uključuju igračev lik, jedno ili više oružja i različit broj neprijatelja. Odvijaju se u 3D okruženju pa su realističnije od 2D pucačina i točnije prikazuju gravitaciju, osvjetljenje, zvukove i sudare. Na zaslonu su obično prikazane ruke i oružje lika te zaslon s informacijama (eng. *Head-up display – HUD*) koji prikazuje zdravlje, količinu streljiva i detalje o lokaciji u obliku karte.

3.3.2. Borba

Fokus pucačina iz prvog lica općenito je akcijska igra, s brzom borbom i dinamičnim vatrenim okršajima kao glavno iskustvo, no određeni naslovi stavljaju veći naglasak na priču, rješavanje problema i logičke zagonetke.

Pucačine iz prvog lica obično igraču daju na izbor velik arsenal oružja od kojih svako oružje ima specifičnu namjenu, pa tako imaju velik utjecaj na to kako će igrač pristupiti igri. Neke igre nude realistične reprodukcije stvarnog oružja simulirajući njihovu brzinu paljbe, veličinu spremnika i preciznost dok druge igre mogu uključivati futurističke prototipove ili vanzemaljsku tehnologiju.

Na počecima pucačina u prvom licu dizajneri su često dopuštali likovima da odjednom nose velik broj različitih oružja, no modernije igre počele su usvajati realističniji pristup gdje igrač može opremiti samo pištolj, zajedno s puškom ili čak ograničavajući igrače na samo jedno oružje po izboru, prisiljavajući ih da prilagode odabir oružja prema situaciji u igri ili načinu na koji žele igrati. [11]

3.3.3. Dizajn razine igre (eng. *Level design*)

Pucačine iz prvog lica mogu biti strukturno sastavljene od razina ili koristiti tehniku kontinuirane naracije pa tako igra nikad ne napušta perspektivu prvog lica, no neke igre također imaju veliku okolinu koja nije podijeljena na razine i ona se može slobodno istraživati. U igrama ovog tipa igrač je u interakciji s okolinom. To može biti nešto jednostavno kao otvaranje vrata, no može biti i rješavanje problema temeljeno na raznim interaktivnim objektima. U nekim igrama igrač može i oštetiti okolinu do određene razine. Ponavljajući motiv u velikom broju igara je bačva koja sadrži eksplozivni materijal.

U pucačinama iz prvog lica za jednog igrača (eng. *singleplayer*) igre imaju različite postavke težine. Tako će na lakšoj postavci igrač uspjeti završiti razinu kroz samu reakciju, dok će na težim postavkama neprijatelji biti jači i agresivniji, nanositi više štete, a pojačanja igračevog lika ograničena. Zbog toga je na težim postavkama često potrebno zapamtiti raspored (eng. *layout*) razine i isplanirati rutu putem pokušaja i pogrešaka.

3.3.4. Igra za više igrača (eng. *Multiplayer*)

Pucačine iz prvog lica mogu sadržavati i način igranja za više igrača te su neke razine napravljene specijalno za takav način igranja. Neke igre dizajnirane su posebno za igranje s više igrača te imaju vrlo ograničene načine igranja za jednog igrača u kojima se igrač natječe protiv likova kojima upravlja sama igra.

Igre za više igrača imaju različite vrste igranja (eng. *Game mode*). Klasične vrste su Deathmatch (i njena timska varijanta Team Deathmatch) u kojoj igrači osvajaju bodove ubijajući likove drugih igrača (ili igrača drugih timova) i osvajanje zastave (eng. *Capture the flag*) gdje je cilj probijanje u neprijateljsku bazu, uzimanje njihove zastave i vraćanje iste u svoju bazu dok sprječavaju drugu ekipu da učini isto. U tim vrstama igranja igrači obično imaju mogućnost biranja između različitih klasa, od kojih svaka ima svoje prednosti i mane, opremu i ulogu unutar tima.

4. Opis naše videoigre

Kao praktični dio ovog rada izradit ćemo videoigru koristeći alat Unity. Već smo opisali žanr u koji će naša igra spadati, no moramo detaljnije utvrditi sve elemente koje će imati. To će biti igra pucanja iz prvog lica za jednog igrača koja će se igrati na osobnom računalu.

4.1. Pucanje

Glavni element koji mora biti obuhvaćen je pucanje. Imat ćemo dva oružja; pušku i pištolj te će puška biti automatska, dakle, kada korisnik drži lijevi klik miša puška će pucati sve dok ima metaka, dok će pištolj biti polu-automatski što znači da će ispaliti metak svakim klikom lijeve tipke miša, no ako bi igrač držao tipku, nakon ispaljenog metka neće se desiti ništa. Oružja su temeljena na stvarnim oružjima: AK-47 i Desert Eagle koje igrač može mijenjati pritiskom na Q ili pomoću kotačića miša. Puška će imati 30 metaka po spremniku, dok će pištolj imati 7. Ako spremnik nije pun, igrač pritiskom na tipku R može pokrenuti punjenje (eng. *Reload*) spremnika, no ako se spremnik potpuno isprazni, punjenje će se pokrenuti automatski.

4.2. Razine

Tema igre je poligon koji bi vojnik trebao proći kao vježbu. Igrač je dakle u ulozi vojnika. Cilj igre proći je poligon u što kraćem vremenu uz prelaženje prepreka. Igra će imati dvije razine. Prva razina bit će smještena je u zgradi s uredima koju su neprijatelji okupirali. Razina je podijeljena u sobe, te će se vrijeme početi brojati kada se igrač približi vratima za ulazak u prvu sobu i vrata će se automatski otvoriti. Vrata druge sobe otvaraju se tek kada igrač pogodi sve neprijatelje u prvoj sobi, a vrijeme se završava kada prođe cijeli poligon. Druga razina smještena je unutar hangara kao vojna vježba u improviziranoj drvenoj zgradi. Vrijeme započinje ulaskom u zgradu u kojoj su također postavljeni neprijatelji i završava se kada igrač pogodi sve neprijatelje. Na ovoj razini postoje i prepreke koje igrač mora proći u čučnju ili preskočiti.

4.3. Kretanje

Igrač upravlja kretanjem lika pomoću tipkovnice. Smjer kretanja određen je tipkom koju korisnik drži: W ili strelica gore za kretanje prema naprijed, S ili strelica dolje za kretanje unazad, A ili strelica lijevo za kretanje ulijevo, a D ili strelica desno za kretanje udesno. Igraču je također omogućeno i trčanje držanjem tipke Shift ili X prilikom kretanja prema naprijed. Igrač

može i čučnuti držanjem tipke Ctrl ili C i skočiti pritiskom na razmaknicu (eng. *Space*). Također, puška se miče lijevo-desno ovisno o brzini kretanja kako bismo dobili realističniji efekt kretanja.

4.4. Korisničko sučelje (eng. *User interface – UI*)

Korisničko sučelje možemo podijeliti na četiri dijela: glavni izbornik, izbornik koji se pojavljuje prilikom pauziranja igre, zaslon s informacijama koji se prikazuje prilikom završetka razine i informacije koje su prikazane na ekranu tijekom igranja.

4.4.1. Glavni izbornik

Ulaskom u igru igraču je prikazan glavni izbornik (eng. *main menu*) na kojem su tri gumba: Odaberi nivo, Postavke i Zatvori.

Klikom na gumb Odaberi nivo otvara se novi izbornik na kojem igrač može odabrati između dvije razine ili kliknuti na gumb za povratak na glavni izbornik. Klikom na neku od dvije razine pojavljuju se podaci o najboljem rezultatu koji je igrač ostvario na toj razini i gumb za pokretanje razine. Klikom na gumb za pokretanje razine učitava se odabrana razina.

Klikom na gumb Postavke otvara se novi izbornik pomoću kojeg igrač odabire postavke igre. Ovdje je moguće postaviti rezoluciju igre, odabrati hoće li igra biti prikazana preko cijelog zaslona i kvalitetu grafike. Također ima i popis kontrola, odnosno koji gumb na tipkovnici odgovara kojoj radnji u igri.

Pritiskom na gumb Zatvori, igra će se ugasiti.

4.4.2. Izbornik za pauzu

Prilikom igranja pritiskom na tipku Escape igra će se pauzirati te će se prikazati izbornik. Ovdje igrač ima izbor između nastavljanja igre, ponovnog pokretanja razine, izlaska na glavni izbornik ili odlaska na izbornik za promjenu postavki vezanih s igranjem. Na izborniku za postavke igrač može mijenjati osjetljivost miša, veličinu, debljinu i razmak nišana te odabrati jednu od ponuđenih boja za nišan: crvena, plava, zelena, žuta ili bijela.

4.4.3. Izbornik za završetak razine

Kod završetka razine prikazuje se izbornik na kojem igrač može ponovno pokrenuti istu razinu, vratiti se na glavni izbornik i prikazuju mu se informacije o rezultatu koji je ostvario:

- Vrijeme – vrijeme koje je igraču bilo potrebno da završi razinu
- Preciznost – broj metaka koji je pogodio metu podijeljen s ukupnim brojem ispaljenih metaka iskazan u postocima

- Broj ispaljenih metaka – broj metaka koje je korisnik ispalio prilikom prelaska razine
- HS% (eng. *Headshot percentage*) – broj metaka koji je pogodio glavu mete podijeljen s brojem meta iskazan u postocima.

Ako je to najbolji rezultat koji je igrač ostvario ispisuje mu se da je to njegov najbolji rezultat, a u suprotnom se ispisuju i podaci o njegovom najboljem rezultatu kako bi mogao usporediti.

4.4.4. Zaslon s informacijama (eng. *Head-up display – HUD*)

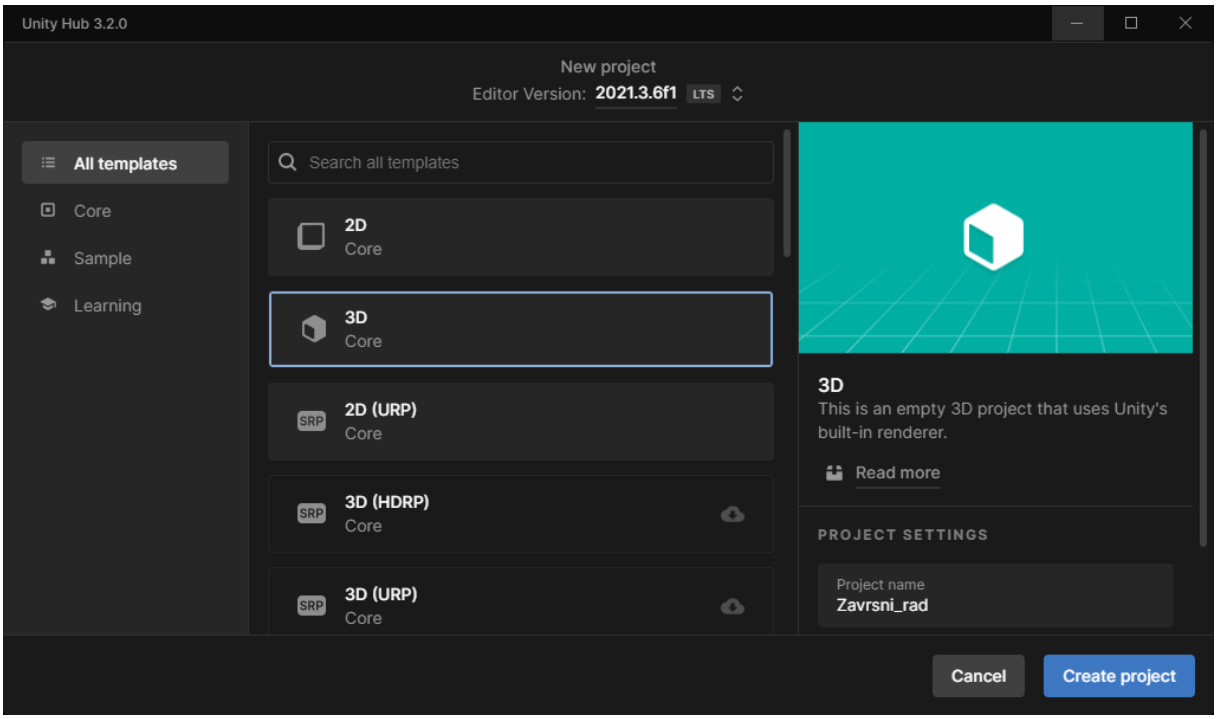
Tijekom igre na zaslonu su prikazane informacije o vremenu, trenutnom broju metaka u spremniku, broju pogođenih meta i ukupnom broju meta.

5. Izrada igre

U nastavku ovog rada opisat ćemo korake koje smo poduzeli za izradu praktičnog dijela ovog rada: videoigre pucanja iz prvog lica. Prikazat ćemo dijelove koda kojima smo ostvarili prethodno opisane funkcionalnosti i objasniti programsku logiku.

5.1. Kreiranje projekta

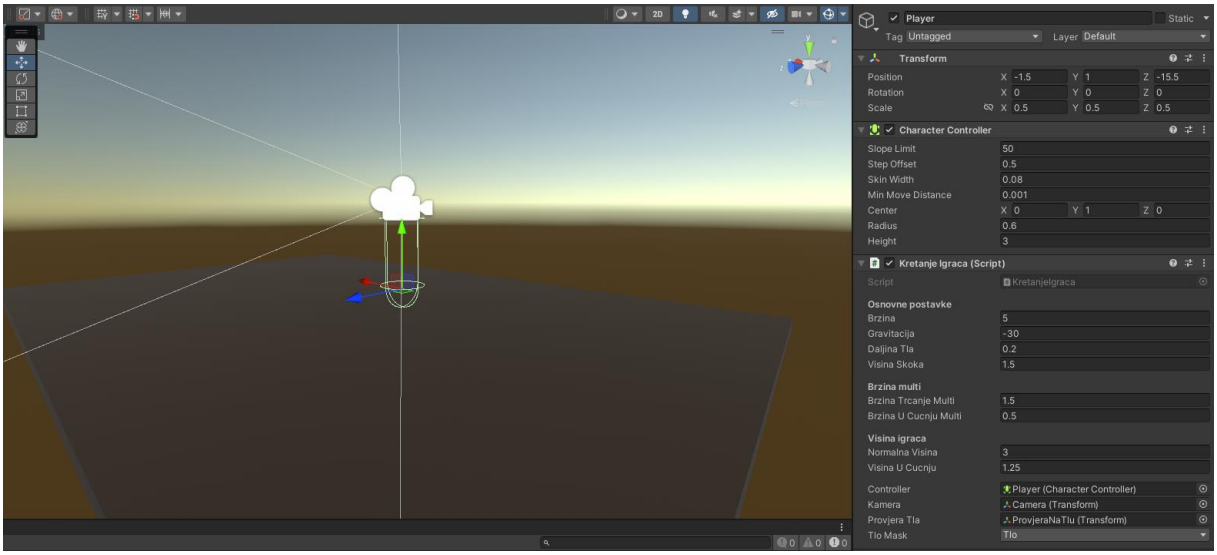
Na početku, bilo je potrebno preuzeti Unity. Nakon što smo ga instalirali, kreirali smo novi projekt i odabrali predložak (eng. *template*) za trodimenzionalnu igru.



Slika 6. Unity - Kreiranje projekta

5.2. Kreiranje lika

Na početku bilo je potrebno kreirati lika. Napravili smo prazan objekt, dodali mu komponentu Character controller i podesili veličinu lika. Kao dijete tog objekta stavili smo i kameru koja je bila kreirana kod stvaranja projekta i postavili je pri vrhu našeg lika. Također, kreirali smo još jedan prazan objekt ProvjeraNaTlu, dijete objekta igrača, koji smo stavili na dno našeg lika. Pomoću njega provjeravat ćemo je li lik na tlu. Nakon toga, objektu smo dodali C# skriptu pod nazivom KretanjeIgraca.



Slika 7. Unity - Kreiranje lika

5.3. Kretanje igrača

Update metoda poziva se prilikom kreiranja svake sličice igre. Na početku provjeravamo je li lik igrača na tlu da bismo kasnije mogli ograničiti pokrete koje igrač može napraviti.

To radimo pomoću objekta ProvjeraNaTlu i metode Checksphere Unity-eve klase Physics. Da bismo to koristili na taj način, tlo moramo postaviti kao poseban layer koji smo nazvali Tlo. Tada u metodu uvrstimo poziciju našeg objekta za provjeru tla: ProvjeraNaTlu, varijablu daljinaTla i layer za tlo kako bi ignorirali sve osim tla. Metoda će stvoriti imaginarnu kuglu s radijusom veličine daljinaTla i ako je ona u dodiru s tlom vratit će true koji spremamo u varijablu naTlu, u suprotnom će vratiti false. Nakon toga, postavljamo trenutnu brzinu padanja na broj manji od nule kako bi se igrač u potpunosti spustio na tlo. To moramo napraviti jer će CheckSphere vratiti true i prije nego smo u potpunosti na tlu zbog radijusa kugle.

Slijedi provjera igračevih unosa, pa ako igrač drži C ili lijevi Ctrl stisnutim poziva se metoda Cucni, a u suprotnom metoda Ustani. Isto tako provjeravamo ako igrač drži X ili lijevi Shift pa postavljamo varijablu uTrcanju na true. Ovdje provjeravamo i kreće li se igrač prema naprijed kako bismo ga ograničili da ne može trčati prema nazad ili u stranu.

Nakon toga u varijablu smjer tipa Vector3 zabilježimo smjer kretanja pomoću igračevih unosa i tak vektor normaliziramo jer bi u suprotnom igrač imao veću briznu prilikom kretanja dijagonalno.

Dalje pomičemo igrača koristeći controller.Move, gdje je controller naš objekt s komponentom Character controller, u prethodno određenom smjeru, brzinom spremljenom u varijabli brzina i multiplierom brzine kretanja u određenom stanju. Zbog toga što ovo radimo u Update metodi, to sve moramo pomnožiti i sa Time.deltaTime kako bi spriječili da broj generiranih sličica po sekundi utječe na brzinu kretanja.

Kasnije će nam kod računanja preciznosti puške biti potrebna brzina kretanja pa je zapisujemo u instancu GameManagera.

Sljedeće provjeravamo je li korisnik stisnuo tipku za skakanje, je li na tlu i da nije u čučnju. Tada dodajemo brzinu na okomitu os u pozitivnom smjeru (prema „gore“) i pomičemo lika. Također, na tu brzinu dodajemo i gravitaciju koja je u suprotnom smjeru, pa dobijemo efekt okomitog hitca.

Metode Cucni i Ustani su jednostavne. Kod njih samo smanjujemo, odnosno povećavamo visinu našeg lika, no u metodi ustani provjeravamo i postoji li neki objekt iznad našeg lika dok je on u čučnju da bismo spriječili da ustane ako npr. prolazi kroz tunel.

```
void Update()
{
        naTlu = Physics.CheckSphere(provjeraTla.position, daljinaTla,
tloMask);

        if(naTlu && padanjeBrzina.y < 0)
        {
            padanjeBrzina.y = -2f;
        }
        if (Input.GetKey(KeyCode.C) || Input.GetKey(KeyCode.LeftControl))
        {
            Cucni();
        }
        else
        {
            Ustani();
        }
        if ((Input.GetKey(KeyCode.X) || Input.GetKey(KeyCode.LeftShift)) &&
Input.GetAxis("Vertical") > 0)
        {
            uTrcanju = true;
        }
        else
        {
            uTrcanju = false;
        }

        Vector3 smjer = Vector3.zero;
        smjer.x = Input.GetAxis("Horizontal");
        smjer.z = Input.GetAxis("Vertical");

        smjer = Vector3.ClampMagnitude(smjer, 1f);
        smjer = transform.right * smjer.x + transform.forward * smjer.z;

        if (uCucnju)
```

```
        {
            controller.Move(smjer * brzina * brzinaUCucnjuMulti *
Time.deltaTime);
        }
        else if(uTrcanju && Puska.reloadUTijeku == false)
        {
            controller.Move(smjer * brzina * brzinaTrcanjeMulti *
Time.deltaTime);
        }
        else
        {
            controller.Move(smjer * brzina * Time.deltaTime);
        }
        GameManager.Instance.brzinaKretanja =
controller.velocity.magnitude;

        if (Input.GetButtonDown("Jump") && naTlu && !uCucnju)
        {
            padanjeBrzina.y = Mathf.Sqrt(visinaSkoka * -2f * gravitacija);

        }
        padanjeBrzina.y += gravitacija * Time.deltaTime;
        controller.Move(padanjeBrzina * Time.deltaTime);
}

void Cucni()
{
        controller.height = visinaUCucnju;
        uCucnju = true;
}

void Ustani()
{
        if(uCucnju && !Physics.Raycast(kamera.transform.position,
Vector3.up, 1f))
        {
            controller.height = normalnaVisina;
            uCucnju = false;
            padanjeBrzina.y = 0;
        }
}
```

5.4. Pogled mišem

Sljedeća skripta dodana je kao komponenta na prethodno spomenutu kameru.

Ovdje prvi puta koristimo klasu PlayerPrefs pa ćemo je objasniti. PlayerPrefs je klasa koja sprema postavke igrača između sesija igre. Pomoću nje moguće je spremiti niz znakova, decimalne brojeve i cijele brojeve u registar (eng. *registry*) platforme.

Start metoda poziva se prije generiranje prve sličice. U njoj na početku resetiramo kursor miša i postavljamo ga kao zaključanog na centar ekrana te on postaje nevidljiv. Ako je to prvi puta da igrač pokreće igru osjetljivost miša postavlja se na 1, no ako je igrač prethodno podesio osjetljivost kroz meni za postavke, učitat će se ta vrijednost.

```
[SerializeField] private float osjetljivostMisa;
public Transform tijelo;
[SerializeField] Slider sensitivitySlider;
[SerializeField] TextMeshProUGUI sensitivityValue;
float rotacijaX = 0f;

void Start()
{
        Cursor.lockState = CursorLockMode.None;
        Cursor.lockState = CursorLockMode.Locked;

        if (osjetljivostMisa == 0f) osjetljivostMisa = 1f;
        if (PlayerPrefs.HasKey("Sensitivity"))
        {
                osjetljivostMisa = PlayerPrefs.GetFloat("Sensitivity");
        }
        else
        {
            osjetljivostMisa = 1f;
            PlayerPrefs.SetFloat("Sensitivity", 1f);
        }
        sensitivitySlider.value = osjetljivostMisa;
        sensitivityValue.text = osjetljivostMisa.ToString("0.00");
}
```

U update metodi započinjemo s provjerom je li igra pauzirana ili ako je razina završena. Tada odmah završavamo izvođenje metode jer ne želimo da se igrač može okretati dok je igra u nekom od tih stanja. Ako to nije slučaj, spremamo igračev pomak miša i množimo ga s osjetljivosti. Od trenutne rotacije kamere po x-osi (gore-dolje) dodajemo pomak miša, ali u suprotnom smjeru kako bi pomak miša prema gore uskladili s pomakom kamere prema gore i suprotno. Nakon toga koristimo Mathf.Clamp metodu kojom ograničavamo rotaciju na 180 stupnjeva i tu rotaciju primjenjujemo na objekt kamere. Kod rotacije lijevo-desno pomičemo cijeli objekt igrača pa odmah možemo primijeniti tu rotaciju bez prethodne manipulacije rotacijom.

Metodu PromjeniSensitivity pozivamo kada igrač u postavkama promjeni osjetljivost miša. Ona postavlja osjetljivost miša na onu koju je korisnik odabrao i sprema postavku korištenjem PlayerPrefs.

```
void Update()
{
        if (Pauziranje.igraPauzirana == true || EndLevel.endUIPrikazan ==
true) return;
        float misX = Input.GetAxis("Mouse X") * osjetljivostMisa;
        float misY = Input.GetAxis("Mouse Y") * osjetljivostMisa;

        rotacijaX -= misY;
        rotacijaX = Mathf.Clamp(rotacijaX, -90f, 90f);
        transform.localRotation = Quaternion.Euler(rotacijaX, 0f, 0f);

        tijelo.Rotate(Vector3.up * misX);
}
```

```
public void PromjeniSensitivity(float sens)
{
        osjetljivostMisa = sens;
        sensitivityValue.text = osjetljivostMisa.ToString("0.00");
        PlayerPrefs.SetFloat("Sensitivity", sens);
}
```

5.5. Puške i pucanje

Da bismo imali realni prikaz oružja na Asset Store-u pronašli smo modele oružja za AK-47 i Desert Eagle.

5.5.1. Promjena puške

Kao dijete igrača dodali smo prazan objekt koji će nam služiti kao držač za puške i kao njegovu djecu stavili preuzete modele puški. Tom objektu dodali smo skriptu pomoću koje će igrač mijenjati između ta dva oružja pritiskom na Q ili korištenjem kotačića na mišu.

Slika 8. Unity - Hijerarhija igrača

Na početku pozivamo metodu OdaberiPusku koja zbog, na početku postavljene varijable indeksa na 0, prikazuje pušku AK-47. Ta metoda radi na način da prolazi kroz svu djecu objekta u kojem držimo puške i aktivira ili deaktivira objekt puške ovisno o indeksu. Taj indeks mijenja se u Update metodi kada korisnik pomakne kotačić miša prema gore ili prema dolje, ili stisne Q.

```
public int indeksOdabranePuske = 0;

void Start()
{
    OdaberiPusku();
}

void Update()
{
    int indeksZadnjeOdabrane = indeksOdabranePuske;

    if(Input.GetAxis("Mouse ScrollWheel") > 0f || Input.GetAxis("Mouse
ScrollWheel") < 0f || Input.GetKeyDown(KeyCode.Q))
    {
```

```
            if(indeksOdabranePuske == 0)
            {
                indeksOdabranePuske = 1;
            }
            else
            {
                indeksOdabranePuske = 0;
            }
        }
        if(indeksZadnjeOdabrane != indeksOdabranePuske) OdaberiPusku();
    }

    void OdaberiPusku()
    {
        int i = 0;
        foreach (Transform puska in transform)
        {
            if(i == indeksOdabranePuske)
            {
                puska.gameObject.SetActive(true);
            }
            else
            {
                puska.gameObject.SetActive(false);
            }
            i++;
        }
    }
```

5.5.2. Problem - „clipping“ puške

Kod prikaza puški pojavio nam se problem da, kada dođemo blizu zida, puška ulazi u zid. To smo riješili na način da smo napravili novi layer i nazvali ga Puska te postavili obje puške da pripadaju tom layeru. Nakon toga napravili smo duplikat glavne kamere, stavili Clear Flags na Depth only i pod Culling Mask postavili da prikazuje samo taj layer, dok smo kod glavne kamere isti maknuli.

Slika 9. Unity - Clipping problem

Slika 10. Unity - Riješen clipping problem

5.5.3. Efekt pucanja i pogotka

Kao dijete svakog od oružja kreirali smo objekt Particle System i postavili ga na vrh cijevi oružja. Također, kao dijete tog Particle Systema kreirali smo i svijetlo tipa Point Light i postavili ga ispod puške. Boju Particle Systema postavili smo na narančastu, pod Emission dodali Burst i dodijelili mu svijetlo. Taj efekt kasnije aktiviramo svaki puta kada puška pukne. Dobiven efekt prikazan je na sljedećoj slici.

Također, kreirali smo još dva Particle Systema koje instanciramo kada metak pogodi neki objekt. Ako je to meta, tada je taj efekt crvene boje, a inače bijelo-narančast.

Pronašli smo i sliku rupe koju metak napravi prilikom pogotka, na primjer, zida. Nju ćemo koristiti tako da je pri ispaljivanju metka postavimo na mjesto koje je metak pogodio.

Dobiven efekt prikazan je na sljedećoj slici.

Slika 11. Unity - Efekt pucanja i pogotka

5.5.4. Zvuk pucanja i punjenja spremnika

Na internetu smo pronašli zvukove pucanja i punjenja spremnika za puške i preuzeli ih. Uvezli smo ih u Unity, svakom od oružja dodali po dvije komponente Audio Source i dodijelili im zvukove. Njih ćemo kasnije pokretati prilikom pucanja i punjenja spremnika.

5.5.5. Funkcionalnosti puške

U Start metodi postavljamo početne vrijednosti i dohvaćamo komponente koje će nam biti potrebne. Dakle, dohvaćamo skriptu WeaponRecoil koju ovdje koristimo za animaciju pucanja pištolja, zapisujemo početnu poziciju puške da bismo je mogli vratiti na tu poziciju nakon animacije za punjenje spremnika. Također, postavljamo broj metaka i prikazujemo ga na ekranu te dohvaćamo zvukove za pucanje i punjenje spremnika.

```
void Start()
{
    weaponRecoil = GetComponent<WeaponRecoil>();
    normPos = transform.localPosition;
    trenutnoMetaka = maxMetaka;
    prikazBrojaTrenutnihMetaka.text = trenutnoMetaka.ToString();
    komponente = GetComponents(typeof(AudioSource));
    foreach(AudioSource zvuk in komponente)
    {
        if(zvuk.clip.name == "ak47-pucanje")
        {
            zvukPucanja = zvuk;
        }
        if(zvuk.clip.name == "ak47-reload")
        {
            zvukReloada = zvuk;
        }

        if (zvuk.clip.name == "deagle-pucanje")
        {
            zvukPucanja = zvuk;
        }
        if (zvuk.clip.name == "deagle-reload")
        {
            zvukReloada = zvuk;
        }
    }
}
```

Metoda OnEnable poziva se prilikom postavljanja objekta kao aktivnog. Dakle, izvršavat će se kod promjene puške, gdje jednu pušku aktiviramo, dok drugu deaktiviramo. Ovdje samo postavljamo zastavicu za punjenje puške na false i prikazujemo broj metaka za tu pušku.

```
void OnEnable()
{
    reloadUTijeku = false;
    prikazBrojaTrenutnihMetaka.text = trenutnoMetaka.ToString();
}
```

5.5.5.1. Update

Na početku metode Update odmah provjeravamo je li punjenje spremnika u tijeku. Ako je to istina, odmah možemo završiti izvođenje metode jer igraču u tom slučaju pucanje nije

omogućeno. Nakon toga provjerimo ima li puška metaka u spremniku te ako nema pokrećemo punjenje spremnika koje ćemo opisati kasnije. Provjeravamo i je li igrač stisnuo R i da spremnik puške nije pun te u tom slučaju isto započinjemo punjenje spremnika.

Nakon toga moramo provjeriti je li puška koja se trenutno koristi automatska ili nije. Automatske i poluautomatske puške moramo odvojiti na taj način jer kod automatskih puški koristimo Input.GetButton koji vraća true sve dok je gumb stisnut, dok kod poluautomatskih koristimo Input.GetButton koji vraća true samo jednom, i to onaj frame kada je prvi put stisnut gumb. Dakle, kada provjerimo je li stisnut gumb i je li prošlo dovoljno vremena od zadnjeg hitca, zabilježimo kada će biti moguće opaliti sljedeći hitac i pozivamo funkciju Pucaj. Kod poluautomatske puške odmah pozivamo metode iz klase weaponRecoil pomoću kojih animiramo trzaj. Kod automatske puške, kada se lijevi klik miša pusti resetiramo uzorak trzaja (eng. *recoil pattern*) puške, što ćemo objasniti kasnije.

```
void Update()
{
    if (reloadUTijeku) return;
    if(trenutnoMetaka <= 0)
    {
        StartCoroutine(Reload());
        return;
    }
    if (Input.GetKeyDown(KeyCode.R) && trenutnoMetaka < maxMetaka)
    {
        StartCoroutine(Reload());
        return;
    }
    if (automatska == true)
    {
        if (Input.GetButton("Fire1") && Time.time >= sljedeciHitac)
        {
            sljedeciHitac = Time.time + 1f / brzinaPucanja;
            Pucaj();
        }
        else if (Input.GetMouseButtonUp(0))
        {
            ResetirajRecoil();
        }
    }
    else
    {
        if (Input.GetButtonDown("Fire1") && Time.time >= sljedeciHitac)
        {
            sljedeciHitac = Time.time + 1f / brzinaPucanja;
            weaponRecoil.DodajRecoil();
            weaponRecoil.DodajRecoilRotacija();
            weaponRecoil.StartCoroutine("MakniRecoilDeagle");
            Pucaj();
        }
    }
}
```

5.5.5.2. Reload

Kao tip metode kojom punimo spremnik puške koristimo IEnumerator da bismo je mogli pozvati kao korutinu. To nam je potrebno jer se izvršene korutine može pauzirati u bilo kojem trenutku pomoću naredbe yield. Tada korutina pauzira izvršene i automatski nastavlja u sljedećem frameu. U korutinama također ćemo koristiti i yield return new WaitForSeconds koja prima broj sekundi u decimalnom obliku te se korutina nastavlja izvoditi od te linije nakon zadanog vremena. Kao što smo vidjeli u prethodnom bloku koda, korutina se započinje naredbom StartCoroutine().

Slika 12. Unity - Početna pozicija puške

Slika 13. Unity - Pozicija puške prilikom punjenja spremnika

Na početku metode provjeravamo je li u pitanju Desert Eagle i pauziramo daljnje izvođenje za 0.1 sekundu kako bi završila animacija trzaja. Prvo postavljamo varijablu reloadUTijeku na true da bismo mogli spriječiti pucanje prilikom punjenja spremnika. Nakon toga pušku postavljamo u poziciju za reload i pokrećemo zvuk punjenja spremnika te čekamo da prođe određen broj sekundi. Nakon toga vraćamo pušku na početnu poziciju, resetiramo trzaj puške, postavimo broj metaka u spremniku na maksimalan broj metaka i prikažemo novi broj metaka na ekranu te postavimo reloadUTijeku na false.

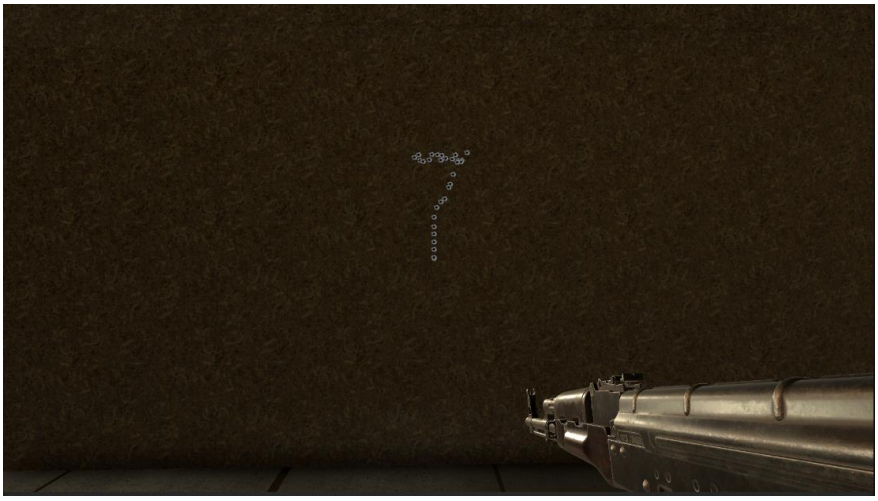
```
IEnumerator Reload()
{
    if (transform.name == "Desert Eagle") yield return new
WaitForSeconds(0.1f);
    reloadUTijeku = true;

    //Animacija reloada i zvuk
    transform.localPosition = reloadPos;
    var rotationVector = transform.rotation.eulerAngles;
    rotationVector.x = 20;
    transform.rotation = Quaternion.Euler(rotationVector);
    zvukReloada.Play();
```

```
        //Trajanje reloada
        yield return new WaitForSeconds(trajanjeReloada);

        //Vracanje puske na početnu poziciju
        rotationVector.x = 0;
        transform.rotation = Quaternion.Euler(rotationVector);
        transform.localPosition = normPos;
        transform.localRotation = Quaternion.identity;

        ResetirajRecoil();

        //Postavljanje metaka
        trenutnoMetaka = maxMetaka;
        prikazBrojaTrenutnihMetaka.text = trenutnoMetaka.ToString();

        reloadUTijeku = false;
    }
```

5.5.5.3. Recoil

Metoda ResetirajRecoil vraća sve varijable korištene za izračunavanje trzaja na 0.

Metoda IzracunajRecoil prima poziciju nišana prilikom poziva metode i vraća smjer u kojem će metak biti ispaljen prema izračunu, ako igrač kontinuirano puca. Izračun radi na sljedeći način: Prvi metak u automatskoj paljbi je uvijek potpuno precizan, osim ako je igrač u pokretu. Nakon toga prvih nekoliko metaka ide prema gore, za određenu daljinu u odnosnu na prethodno ispaljeni metak. Nakon nekoliko metaka, i dalje se sljedeći pomiče za istu daljinu prema gore, no sada se za određenu daljinu pomiče i udesno. Kada ta točka dođe do određene gornje granice, ostaje na toj visini no počinje se pomicati ulijevo. Tako dobijemo uzorak nalik na broj 7 što možemo vidjeti na sljedećoj slici.



Slika 14. Unity – Recoil

```
private void ResetirajRecoil()
{
    prviMetakUSprayu = true;
    kolicinaRecoilaUp = 0;
    prosliRecoilUp = 0;
    kolicinaRecoilaR = 0;
    prosliRecoilR = 0;
}

private Vector3 IzracunajRecoil(Vector3 pozicijaCrosshaira)
    {
        if (!prviMetakUSprayu)
        {
            //Prvih nekoliko metaka recoil samo prema gore
            if (kolicinaRecoilaUp < 0.05f)
            {
                prosliRecoilUp = kolicinaRecoilaUp;
                kolicinaRecoilaUp += recoilMultiUp;
            }
            //Nakon nekoliko metaka dodajemo recoil i prema desno
            else if (kolicinaRecoilaUp < 0.14f)
            {
                prosliRecoilUp = kolicinaRecoilaUp;
                kolicinaRecoilaUp += recoilMultiUp;

                prosliRecoilR = kolicinaRecoilaR;
                kolicinaRecoilaR += recoilMultiRL;
                //Ovisno o smjeru u kojem gledamo mijenjaju se osi x i z pa
se mijenja i smjer "desno"
                if (pozicijaCrosshaira.z > 0.45f) pozicijaCrosshaira.x +=
UnityEngine.Random.Range(prosliRecoilR, kolicinaRecoilaR);
                else if (pozicijaCrosshaira.z < -0.45f)
pozicijaCrosshaira.x -= UnityEngine.Random.Range(prosliRecoilR,
kolicinaRecoilaR);
                else if (pozicijaCrosshaira.x > 0.45f) pozicijaCrosshaira.z
-= UnityEngine.Random.Range(prosliRecoilR, kolicinaRecoilaR);
                else if (pozicijaCrosshaira.x < -0.45f)
pozicijaCrosshaira.z += UnityEngine.Random.Range(prosliRecoilR,
kolicinaRecoilaR);


            }
            //Nakon sto recoil dođe do najviše točke, dodajemo recoil prema
lijevo
            else
            {
                kolicinaRecoilaUp = 0.14f;
                prosliRecoilUp = 0.13f;

                kolicinaRecoilaR = prosliRecoilR;
                prosliRecoilR -= recoilMultiRL;
                //Ovisno o smjeru u kojem gledamo mijenjaju se osi x i z pa
se mijenja i smjer "desno"
                if (pozicijaCrosshaira.z > 0.45f) pozicijaCrosshaira.x +=
UnityEngine.Random.Range(prosliRecoilR, kolicinaRecoilaR);
                else if (pozicijaCrosshaira.z < -0.45f)
pozicijaCrosshaira.x -= UnityEngine.Random.Range(prosliRecoilR,
kolicinaRecoilaR);
                else if (pozicijaCrosshaira.x > 0.45f) pozicijaCrosshaira.z
-= UnityEngine.Random.Range(prosliRecoilR, kolicinaRecoilaR);
```

```
                else if (pozicijaCrosshaira.x < -0.45f)
pozicijaCrosshaira.z += UnityEngine.Random.Range(prosliRecoilR,
kolicinaRecoilaR);
            }
            pozicijaCrosshaira.y +=
UnityEngine.Random.Range(prosliRecoilUp, kolicinaRecoilaUp);
        }
        else
        {
            prviMetakUSprayu = false;
        }
        return pozicijaCrosshaira;
    }
```

5.5.5.4. Pucanje

Na početku metode Pucaj provjerava se da igra nije pauzirana i da nije prikazan meni koji se prikazuje prilikom završetka razine te ako je to istina, odmah se prekida izvođenje metode. Nakon toga smanjujemo broj trenutnih metaka za 1, prikazujemo tu promjenu na ekranu. Kreiramo varijablu RaycastHit u koju se spremaju informacije o Raycastu. Raycast je metoda koja prima početnu poziciju i smjer. Ona ispaljuje zraku (eng. *ray*), odnosno zamišljenu liniju iz početne pozicije prema zadanom smjeru te vraća podatke o pogođenom objektu, ako objekt ima komponentu Collider. Dakle, tu varijablu koristit ćemo kako bi prepoznali što je metak pogodio. Nakon toga prikazujemo efekt vatre koja izlazi iz cijevi i pokrećemo zvuk pucanja.

Sada je potrebno zabilježiti smjer kamere, odnosno sredinu ekrana koji igrač vidi da bismo imali smjer u kojem ćemo ispaliti metak. Zatim pozivamo prethodno opisanu metodu kojom izračunavamo trzaj puške i u instanci GameManagera upisujemo ispaljeni metak. Potrebno je i provjeriti kreće li se igrač jer ne želimo da u tom slučaju puška bude precizna. Dakle, na već izračunati smjer pucanja metka dodajemo slučajan broj između dvije vrijednosti. Sada možemo ispaliti metak korištenjem Raycast od pozicije kamere igrača, u prethodno izračunatom smjeru i podatke o pogotku spremamo u varijablu pogodak. Nakon toga, ovisno o pogođenom dijelu tijela mete, meti smanjujemo zdravlje i na mjestu pogotka prikazujemo efekt pogotka. Ako nije pogođena meta, nego neki drugi objekt, na mjestu pogotka prikazujemo drugačiji efekt pogotka i prikaz rupe od metka na pogođenom objektu.

```
private void Pucaj()
{
    if (Pauziranje.igraPauzirana == true || EndLevel.endUIPrikazan == true)
return;
    trenutnoMetaka--;
    prikazBrojaTrenutnihMetaka.text = trenutnoMetaka.ToString();
    RaycastHit pogodak;

    Vatra.Play();
    zvukPucanja.Play();
    Vector3 pozicijaCrosshaira = igracCam.transform.forward;
```

```
    pozicijaCrosshaira = IzracunajRecoil(pozicijaCrosshaira);

    GameManager.Instance.DodajIspaljeni();

    if(GameManager.Instance.brzinaKretanja > 2.52 || !KretanjeIgraca.naTlu)
    {
        if (KretanjeIgraca.naTlu) jumpInaccuracyMulti = 1f;
        else jumpInaccuracyMulti = 2f;
        pozicijaCrosshaira.x += UnityEngine.Random.Range(-gunInaccuracy *
jumpInaccuracyMulti, gunInaccuracy * jumpInaccuracyMulti);
        pozicijaCrosshaira.y += UnityEngine.Random.Range(-gunInaccuracy *
jumpInaccuracyMulti, gunInaccuracy * jumpInaccuracyMulti);
        pozicijaCrosshaira.z += UnityEngine.Random.Range(-gunInaccuracy *
jumpInaccuracyMulti, gunInaccuracy * jumpInaccuracyMulti);
    }

    if (Physics.Raycast(igracCam.transform.position, pozicijaCrosshaira,
out pogodak))
    {
        if (pogodak.transform.name == "Glava" || pogodak.transform.name ==
"Tijelo" || pogodak.transform.name == "LRuka" || pogodak.transform.name ==
"DRuka")
        {
            Meta meta = pogodak.transform.parent.GetComponent<Meta>();

            if (meta != null)
            {
                if (pogodak.transform.name == "Glava")
                {
                    meta.SmanjiHP(damage * hsMulti);
                    GameManager.Instance.DodajHS();
                }
                else if (pogodak.transform.name == "Tijelo")
                {
                    meta.SmanjiHP(damage * bodyMulti);
                }
                else if (pogodak.transform.name == "LRuka" ||
pogodak.transform.name == "DRuka")
                {
                    meta.SmanjiHP(damage * armMulti);
                }
            }
            GameObject efektBlood = Instantiate(bloodSplatter,
pogodak.point, Quaternion.LookRotation(pogodak.normal));
            Destroy(efektBlood, 2f);
        }
        else
        {
            GameObject efektPogotka = Instantiate(pogodakEfekt,
pogodak.point, Quaternion.LookRotation(pogodak.normal));

            GameObject bulletDecal = Instantiate(bulletHoleDecal);
            bulletDecal.transform.position = pogodak.point;
            bulletDecal.transform.forward = pogodak.normal * -1f;

            Destroy(bulletDecal, 5f);
            Destroy(efektPogotka, 2f);
        }
    }
}
```

5.5.6. Animacija trzaja puške

Da bi pucanje bilo realističnije, bilo je potrebno implementirati animaciju trzaja puške. Iako Unity ima vrlo dobre alate za kreiranje animacija odlučili smo to napraviti pomoću skripte.

Kao i kod pucanja, i ovdje želimo spriječiti prikaz animacije prilikom punjenja spremnika. Već smo objasnili razliku automatskih i poluautomatskih puški kod pucanja, pa tako na različit način radimo i animaciju. Razlog tomu je da automatskoj puški kontinuirano dodajemo trzaj i vraćamo na početnu poziciju tek kad pucanje završi, a kod pištolja odmah nakon ispaljenog metka dodajemo i oduzimamo trzaj.

```
void Update()
{
    if (Puska.reloadUTijeku) return;

    if (transform.name == "AK-47")
    {
        if (Input.GetButton("Fire1"))
        {
            DodajRecoilRotacija();
            DodajRecoil();
        }
        else if (Input.GetButtonUp("Fire1"))
        {
            MakniRecoilRotacija();
            MakniRecoil();
        }
    }
}
```

Metoda DodajRecoilRotacija automatskoj puški kontinuirano dodaje rotaciju „prema gore“ sve dok traje pucanje, a metoda MakniRecoilRotacija vraća pušku na početnu vrijednost rotacije.

```
public void DodajRecoilRotacija()
{
    transform.localEulerAngles += recoil * Time.deltaTime;
}

public void MakniRecoilRotacija()
{
    transform.localEulerAngles = pocetnaPozicijaRotacijaAK;
}
```

Metoda DodajRecoil služi nam za dodavanje trzaja „naprijed-natrag“. To radimo tako da doslovno pomičemo lokalnu poziciju puške po z-osi. Dakle, odredili smo maksimalnu i minimalnu vrijednost hoda puške gdje je maksimalna vrijednost početna pozicija puške, a minimalna - najdalja pozicija do koje će puška trznuti. To koristimo tako da prilikom pucanja, velikom brzinom, oduzimamo određenu vrijednost z-osi sve do minimalne vrijednosti, a nakon toga pribrajamo vrijednost do maksimalne vrijednosti.

Za pištolj također koristimo ove metode, no pozivamo ih odmah prilikom pucanja i pokrećemo korutinu MakniRecoilDeagle koja će nakon određenog vremena odmah postaviti pištolj na početnu poziciju.

```
public void DodajRecoil()
{
    if (transform.localPosition.z >= recoilF)
    {
        maxF = true;
        maxB = false;
    }
    else if (transform.localPosition.z <= recoilB)
    {
        maxF = false;
        maxB = true;
    }

    if (maxF)
    {
        transform.localPosition -= Vector3.forward * recoilMulti *
Time.deltaTime;
    }
    else if (maxB)
    {
        transform.localPosition += Vector3.forward * recoilMulti *
Time.deltaTime;
    }
}

public void MakniRecoil()
{
    transform.localPosition = pocetnaPozicijaAK;
}

public IEnumerator MakniRecoilDeagle()
{
    yield return new WaitForSeconds(deagleTrajanjeAnimacije);
    transform.localPosition = pocetnaPozicijaDeagle;
    transform.localEulerAngles = pocetnaPozicijaRotacijaDeagle;
}
```

5.5.7. Sway i bob

Kako bismo izbjegli da puška bude potpuno mirna prilikom kretanja odlučili smo napraviti kretanje puške. Sway možemo objasniti kao „kašnjenje“ puške prilikom pogleda mišem dok je Bob pomicanje puške lijevo-desno, kao što bi se puška u stvarnosti pomicala kada bi čovjek hodao, usklađeno s koracima.

Ovdje koristimo kvaternione (eng. *Quaternion*), no nećemo ih objašnjavati, nego samo reći da je to klasa za spremanje trodimenzionalne orijentacije i opisivanje relativne rotacije iz jedne orijentacije u drugu. Koristit ćemo dvije metode te klase AngleAxis i Slerp. AngleAxis prima dvije vrijednosti; *kut* i *os*, i stvara rotaciju koja rotira *kut* stupnjeva oko *osi*. Slerp prima

dva kvaterniona *a,b* i decimalni broj *t* koji je ograničen na raspon [0, 1]. Ova metoda sferno interpolira između kvaterniona *a* i *b* omjerom *t*.

Dakle, na početku, ako punjenje spremnika nije u tijeku, spremamo pomak miša i kreiramo ciljnu rotaciju te sferno interpoliramo između trenutne rotacije i ciljne rotacije.

```
void Update()
{
    if (!Puska.reloadUTijeku)
    {
        float mouseX = Input.GetAxisRaw("Mouse X") * swayMultiplier;
        float mouseY = Input.GetAxisRaw("Mouse Y") * swayMultiplier;

        Quaternion rotationX = Quaternion.AngleAxis(-mouseY,
Vector3.right);
        Quaternion rotationY = Quaternion.AngleAxis(mouseX, Vector3.up);

        Quaternion targetRotation = rotationX * rotationY;

        transform.localRotation = Quaternion.Slerp(transform.localRotation,
targetRotation, smooth * Time.deltaTime);

        Bob();
    }
}
```

Bob smo implementirali na isti način kao i trzaj puške „naprijed-natrag“, no ovdje je to kretanje „lijevo-desno“. Imamo maksimalnu vrijednost za lijevo i maksimalnu vrijednost za desno te, ako je brzina kretanja igrača veća od 1, pomičemo pušku od jedne vrijednosti do druge, a brzina kretanja puške ovisi o brzini kretanja igrača.

```
void Bob()
{
    if (GameManager.Instance.brzinaKretanja > 1)
    {
        if (transform.localPosition.x >= bobD)
        {
            maxD = true;
            maxL = false;
        }
        else if (transform.localPosition.x <= bobL)
        {
            maxD = false;
            maxL = true;
        }
        if (maxD)
        {
            transform.localPosition -= Vector3.right *
GameManager.Instance.brzinaKretanja / bobMulti * Time.deltaTime;
        }
        else if (maxL)
        {
            transform.localPosition += Vector3.right *
GameManager.Instance.brzinaKretanja / bobMulti * Time.deltaTime;
        }
    }
}
```

5.6. Mete

Na Asset Store-u pronašli smo model vojnika koji ćemo koristiti kao meta. Da bismo mogli raspoznati u koji dio tijela je meta pogođena kreirali smo jednostavne 3D objekte koji imaju samo Collider, postavili ih kao djecu modela vojnika i pozicionirali ih na pripadajuće mjesto. To nam je potrebno kako bi mogli povećati ili smanjiti štetu koju nanosimo meti ovisno o pogođenom dijelu tijela.

Slika 15. Unity - Meta

Skripta koja je dodijeljena meti kao komponenta je jednostavna. Ovdje imamo jednu varijablu kojom označavamo zdravlje i dvije metode. Jedna metoda služi za smanjivanje zdravlja i bilježenje pogotka, a druga za micanje objekta iz scene i bilježenje eliminacije mete.

```
public float hp = 100f;

public void SmanjiHP(float dmg)
{
    GameManager.Instance.DodajPogodak();
    hp -= dmg;
```

```
    if (hp <= 0)
    {
        Despawn();
    }
}

void Despawn()
{
    Destroy(gameObject);
    GameManager.Instance.DodajKill();

}
```

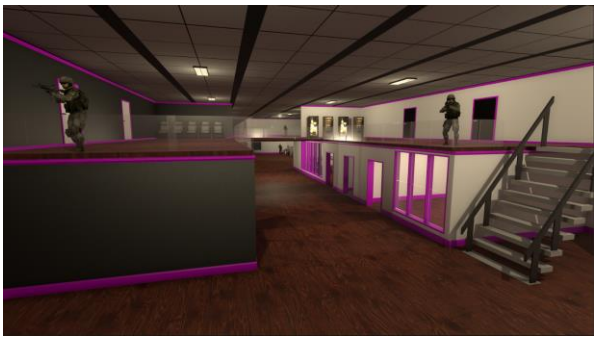
5.7. Izrada razina

Za izradu razina preuzeli smo dva Unity-eva Asseta: „Snaps Prototype | Office“ i „Snaps Prototype | Sci-Fi / Industrial“. Pomoću njih je jednostavnije izraditi razine jer u njima možemo pronaći već napravljene objekte zidova, podova, vrata i rekvizita kojima ćemo urediti našu razinu.

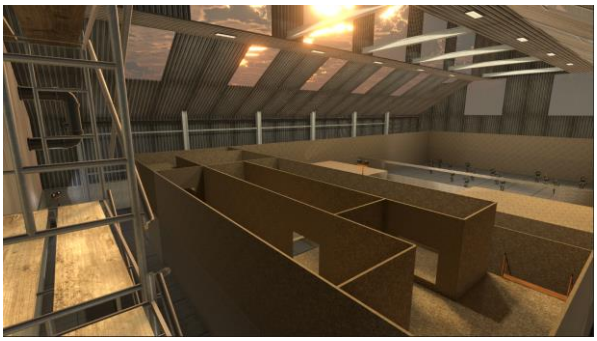
Slika 16. Unity - Asset za prototipiranje

Krajnji izgled naših razina prikazan je na sljedeće četiri slike, a u nastavku ćemo opisati neke elemente.

Slika 17. Unity - Prva razina - prva soba



Slika 18. Unity - Prva razina - druga soba



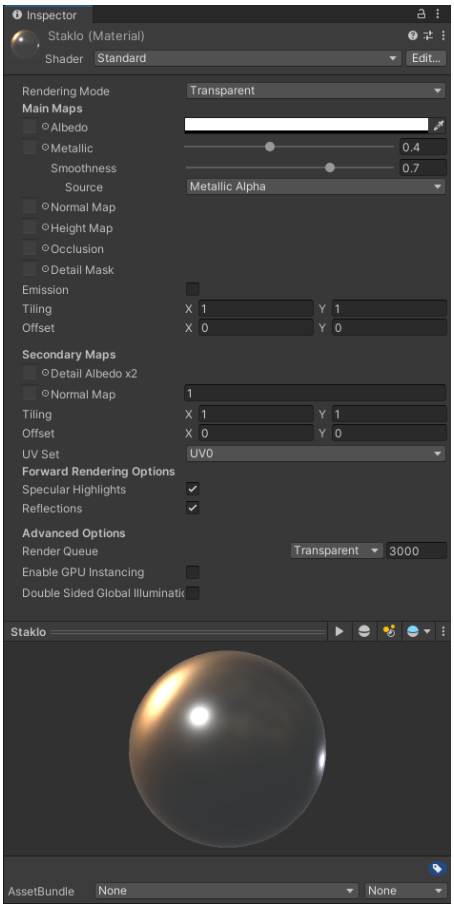
Slika 19. Unity - Druga razina

Slika 20. Unity - Druga razina - prva soba

5.7.1. Materijali

Materijale poput stakla, bijele i crne boje ili metala izradili smo sami, prikazano na slici 21, dok smo materijale kao drvo preuzeli na Asset Store-u.

Materijale smo izradili na način da smo kreirali novi materijal (eng. *Material*), odabrali jedan od željenih Rendering modova: Transparent za prozirne objekte, a Opaque za neprozirne, odabrali boju koja nam je potrebna te podešavali Metallic i Smoothness. Parametar Metallic određuje koliko je površina „slična“ metalu (eng. *metal-like*) Kada je površina više metalna, ona više reflektira okolinu i njena zadana boja postaje manje vidljiva. Parametrom Smoothness određujemo glatkoću površine. Glatka površina ima vrlo malo detalja mikropovršine ili ih uopće nema pa se svjetlost odbija ujednačeno, stvarajući jasne refleksije. Ako je površina hrapava, tada se svjetlost odbija u širokom rasponu kutova koji stvaraju difuznu boju bez jasnih refleksija.



Slika 21. Unity - Materijal Staklo

5.7.2. Okidači (eng. *trigger*)

U opisu prve razine spomenuli smo automatsko otvaranje vrata. To smo postigli pomoću okidača postavljenih ispred vrata koji, kada igrač uđe u njih, pokreću animaciju otvaranja vrata. Prikazat ćemo kod za otvaranje prvih vrata.

Metoda OnTriggerEnter poziva se prilikom sudara objekata što je u našem slučaju ulazak igrača u prostor okidača. Za prvi okidač provjeravamo je li brojač vremena započet kako igrač ne bi mogao ponovo ući u prostor okidača i resetirati brojač. Tada pozivamo metodu GameManagera, pokrećemo zvuk otvaranja nove sobe i postavljamo zastavicu da je zvuk pušten.

```
private void OnTriggerEnter(Collider other)
{
    if(!GameManager.aktivanTimer) GameManager.Instance.ZapocniVrijeme();
    if (!soundPusten)
    {
        pocetakVremena.Play();
        soundPusten = true;
    }
}
```

Sljedeći blok koda nalazi se u skripti GameManager.cs, no opisat ćemo ga ovdje zbog preglednosti. U metodi ZapocniVrijeme postavljamo zastavicu da je brojač aktivan, provjeravamo da smo na prvoj razini i postavljamo parametar animatora na true pa se pokreće animacija otvaranja vrata. Također, postavljamo brojač ispaljenih metaka na 0 kako ne bismo brojali metke koje je igrač ispalio prije nego što je započelo brojanje vremena.

```
public void ZapocniVrijeme()
{
    aktivanTimer = true;
    if (SceneManager.GetActiveScene().name == "Poligon1")
    {
        otvaranjeVrata.SetBool("OtvoriVrata", true);
    }
    brojIspaljenihMetaka = 0;
}
```

5.7.3. Osvjetljenje

U Unity-u postoje četiri tipa svijetla:

- Point Light – emitira svjetlost u svim smjerovima jednako
- Spot Light – emitira svjetlo u obliku stošca
- Directional Light – nalazi se beskrajno daleko i emitira svjetlost samo u jednom smjeru, obično korišteno kao sunce
- Area Light – svjetlo definirano pravokutnikom ili diskom, emitira svjetlost u svim smjerovima ravnomjerno preko,

no mi smo u našem projektu koristili samo Point light, Spot light i Directional light.

Također, postoje različiti načini rada svjetala:

- Baked – Unity unaprijed izračunava (eng. *pre-calculates*) osvjetljenje prije izvođenja igre i ne uključuje ih u izračune osvjetljenja prilikom izvođenja.
- Realtime – Unity izračunava i ažurira osvjetljenje prilikom generiranja svake sličice igre.
- Mixed – kombinira elemente Baked i Realtime načina, na primjer kombiniranje dinamičkih sjena s pečenim (eng. *baked*) osvjetljenjem iz istog izvora svjetla.

Zbog optimizacije performansi igre Realtime smo koristili samo za glavna svijetla u sobi, Mixed za svjetla blizu meta, a većinu sporednih svjetala postavili smo kao Baked.

5.7.4. GameManager

GameManager je skripta kojom upravljamo tijekom igre. Zbog opsežnosti skripte nećemo prikazivati kod, ali ćemo je opisati.

Klasa GameManager služi kao Singleton što znači da postoji samo jedna instanca ove klase, a koristimo je na taj način zbog jednostavnog pristupa podacima klase. U nju tijekom igre zapisujemo podatke o samom tijeku igre poput trenutnog stanja brojača vremena, broja ispaljenih metaka, broja pogodaka, broja preostalih meta i slično, o igraču kao što je brzina kretanja igrača te imamo reference na animatore preko kojih pokrećemo animacije.

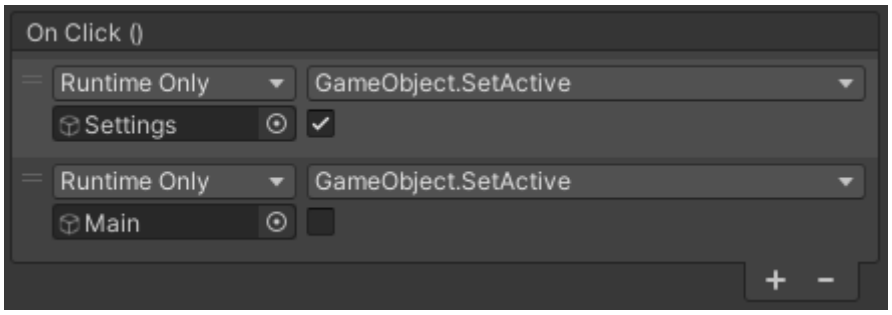
Dakle, u ovoj klasi započinjemo vrijeme, brojimo proteklo vrijeme, otvaramo vrata soba, zapisujemo podatke o pucanju i metama te zaustavljamo vrijeme.

5.8. Korisničko sučelje (eng. *User interface - UI*)

U ovom dijelu opisat ćemo izbornike i zaslon s informacijama, priložiti slike kreiranih izbornika i objasniti na koji način rade.

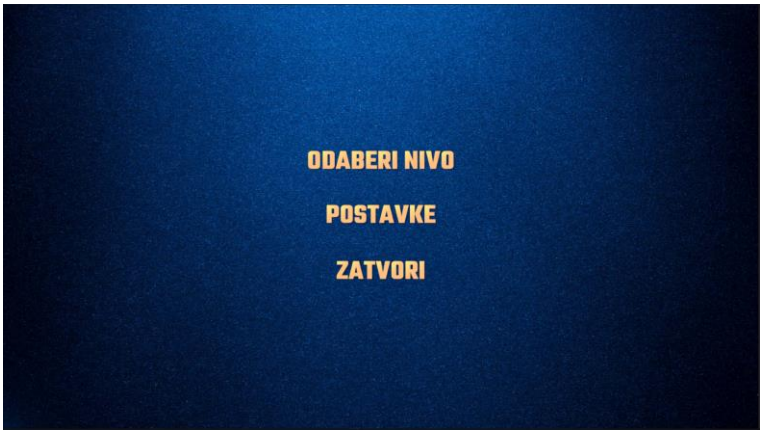
5.8.1. Glavni izbornik (eng. *Main menu*)

Da bismo kreirali glavni izbornik napravili smo novu scenu i u njoj dodali objekt tipa Canvas. Na njemu smo postavili sliku pozadine i gumbe. Budući da su sva tri zaslona izbornika na istom Canvasu, kroz zaslone navigiramo pomoću OnClick funkcije koju možemo dodati na gumb. Tom funkcijom, na način prikazan na slici 22., klikom na gumb deaktiviramo zaslon početni zaslon, a aktiviramo zaslon za postavke.



Slika 22. Unity - OnClick funkcija

Na početnom zaslonu (slika 23.) glavnog izbornika igrač ima 3 opcije: odabrati nivo (slika 24.), promijeniti postavke igre (slika 25.) ili zatvoriti igru.



Slika 23. Unity - Glavni izbornik - početni zaslon

Slika 24. Unity - Glavni izbornik - odaberi nivo

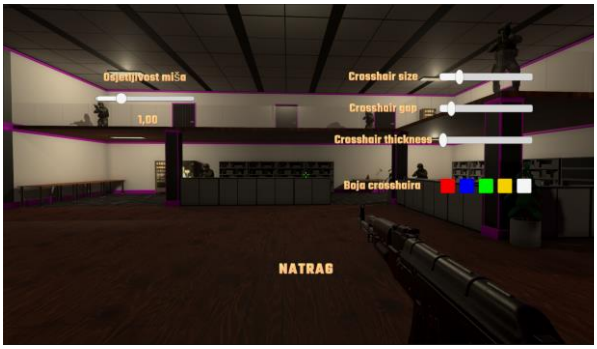
Slika 25. Unity - Glavni izbornik - postavke

5.8.2. Izbornik za pauzu (eng. *Pause menu*)

Izbornik za pauzu napravljen je na isti način kao i glavni izbornik, no on se aktivira kada igrač tijekom igre pritisne Escape tipku. Tada se zaustavlja vrijeme naredbom `Time.timeScale = 0f` i igrač ima opciju nastaviti igru, ispočetka pokrenuti razinu ili otići na glavni izbornik. Ako ode u meni za postavke, tamo su dodani klizači kojima može mijenjati osjetljivost miša, veličinu nišana, razmak nišana te debljinu nišana. Također, klikom na neku od boja može promijeniti boju nišana.

U ovom izborniku, sve postavke se primjenjuju odmah, ali se i spremaju pomoću klase PlayerPrefs.

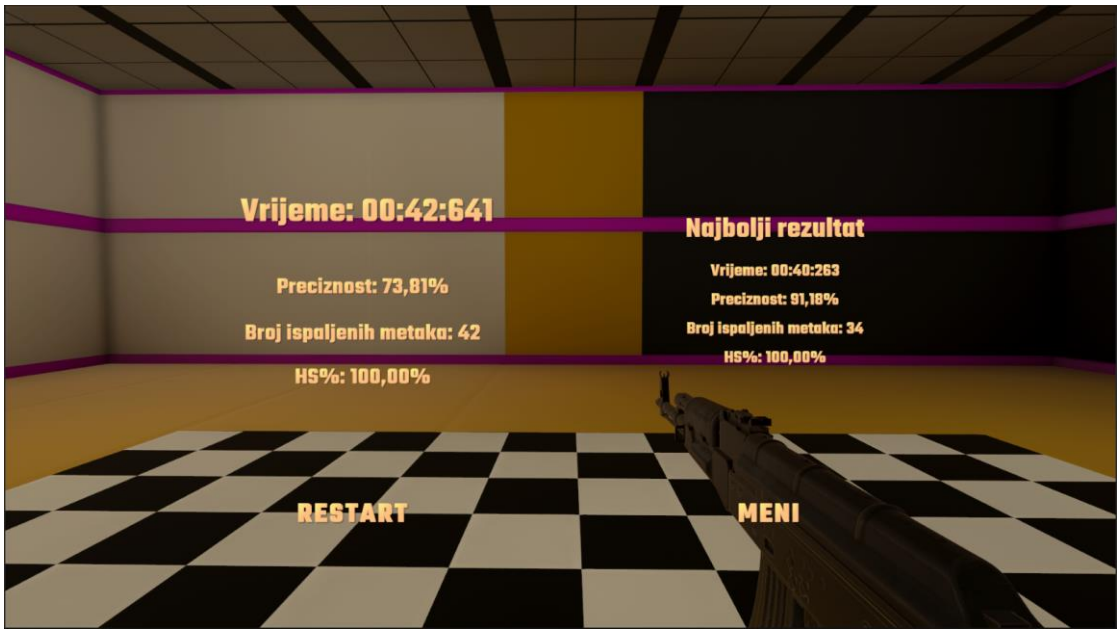
Slika 26. Unity - Pauza



Slika 27. Unity - Pauza - postavke

5.8.3. Izbornik za završetak razine (eng. *End menu*)

Završetkom razine prikazuje se izbornik s informacijama o igračevom rezultatu. Ako je to prvi puta da je završio razinu, rezultat se sprema. Ako već završio razinu, a rezultat je lošiji nego prije prikazuju mu se podaci o rezultatu trenutnog pokušaja i podaci najboljeg rezultata, a ako je rezultat bolji tada se taj rezultat sprema.



Slika 28. Unity - End menu

5.8.4. Zaslon s informacijama (eng. *Head-up display – HUD*)

Zaslon s informacijama igraču tijekom igranja pruža bitne informacije. Na gornjem dijelu ekrana u stvarnom vremenu prikazuje se proteklo vrijeme od početka razine. Na dnu ekrana stoji informacija o broju ubijenih meta od ukupnog broja meta, a u desnom donjem kutu je trenutni broj metaka u spremniku puške.

Slika 29. Unity - Zaslon s informacijama

6. Zaključak

U ovom radu objasnili smo da je Unity jedan od najpopularnijih pokretača igre za 2D i 3D igre, naveli da se koristi i u drugim industrijama i neka od svojstava koja pruža. Također, naveli smo i da dolazi u 3 verzije i opisali izgled sučelja. Nakon toga objasnili smo da je Asset Store rastuća biblioteka datoteka i dali primjere datoteka koje se mogu pronaći za sve kategorije datoteka.

U sljedećem odlomku opisali smo što su videoigre, naveli da glavna podjela videoigara prema platformama obuhvaća videoigre za osobno računalo, videoigre za konzole i arkadne videoigre. Naveli smo, uz kratak opis, i osnovne žanrove igara. Nakon toga kratko smo opisali i žanr pucačina, da bismo mogli doći do žanra igre koju smo izradili, videoigre pucanja iz prvog lica. Tada smo detaljnije opisali žanr pucanja iz prvog lica da bismo dobili bolju sliku o elementima koje igra mora sadržavati.

Sljedeće na redu bilo je detaljno opisivanje kako bi naša igra trebala izgledati. Ovdje smo opisali da će u igri biti dva oružja temeljena na stvarnim oružjima i da će jedno biti automatsko, a drugo poluautomatsko. Opisali smo i kako bi se puške trebale ponašati prilikom pucanja, punjenja spremnika i promjene puške. Tada smo spomenuli da je tema igre poligon koji vojnik mora proći kao vježbu i opisali izgled razina. Naveli smo i načine na koje bi se igrač može kretati te koje funkcionalnosti izbornika moraju biti zadovoljene.

Sve do kraja rada opisivali smo izradu te igre počevši s kreiranjem projekta i lika. Tada smo detaljno, uz priložen programski kod, opisali programsku logiku kretanja igrača, pogleda mišem, svega vezanog za puške i mete. Na kraju smo ukratko prikazali način na koji smo izradili razine i korisničko sučelje.

Popis literature

[1] M. Saltzman. (2021, 10. lip.). „E3 2021: Video games are bigger business than ever, topping movies and music combined,“ USA TODAY. Dostupno: https://eu.usatoday.com/videos/tech/2021/06/10/e-3-2021-video-games-big-business-topping-film-and-music-combined/7637695002/ [pristupano 07.09.2022.].

[2] R. Browne (2022, 7. srp.). „Video game sales set to fall for first time in years as industry braces for recession,“ CNBC. Dostupno: https://www.cnbc.com/2022/07/07/video-game-industry-not-recession-proof-sales-set-to-fall-in-2022.html [pristupano 07.09.2022.].

[3] A. Sinicki (2021, 20. ožu.). „What is Unity? Everything you need to know,“ AndroidAuthority. Dostupno: https://www.androidauthority.com/what-is-unity-1131558/ [pristupano 04.09.2022.].

[4] Unity logo [Slika] (bez dat.) Dostupno: https://unity3d.com/fr/legal/branding_trademarks [pristupano 05.09.2022.]

[5] Unity (bez dat.) Quick guide to the Unity Asset Store [Na internetu]. Dostupno: https://unity3d.com/quick-guide-to-unity-asset-store [pristupano 07.09.2022.].

[6] „Video game,“ (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Video_game [pristupano 03.09.2022.]

[7] D. Pavlovic, „Video Game Genres: Everything You Need to Know“, 2020. [Na internetu]. Dostupno: https://www.hp.com/us-en/shop/tech-takes/video-game-genres [pristupano 04.09.2022.].

[8] *Doom* [Slika] (bez dat.) Dostupno: https://upload.wikimedia.org/wikipedia/en/d/d1/Doom_ingame_2.png [pristupano 05.09.2022.]

[9] M. Konnikova (2013, 25. stu.). „Why gamers can't stop playing first-person shooters,“ The NewYorker. Dostupno: https://www.newyorker.com/tech/annals-of-technology/why-gamers-cant-stop-playing-first-person-shooters [pristupano 04.09.2022.]

[10] *Half-Life* [Slika] (bez dat.) Dostupno: https://upload.wikimedia.org/wikipedia/en/6/65/Halflife_ingame.jpg?20170112182111 [pristupano 05.09.2022.]

[11] „First-person shooter,“ (bez dat.). u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/First-person_shooter [pristupano 03.09.2022.]

[12] Unity (bez dat.) Manual [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/index.html [pristupano 06.09.2022.]

Popis slika

Slika 1. Unity logo [4] 2
Slika 2. Unity - Sučelje za uređivanje 3
Slika 3. Unity Asset Store (assetstore.unity.com) 4
Slika 4. Doom (1993.) [8] 9
Slika 5. Half-Life [10] 9
Slika 6. Unity - Kreiranje projekta 15
Slika 7. Unity - Kreiranje lika 16
Slika 8. Unity - Hijerarhija igrača 20
Slika 9. Unity - Clipping problem 21
Slika 10. Unity - Riješen clipping problem 21
Slika 11. Unity - Efekt pucanja i pogotka 22
Slika 12. Unity - Početna pozicija puške 25
Slika 13. Unity - Pozicija puške prilikom punjenja spremnika 25
Slika 14. Unity – Recoil 26
Slika 15. Unity - Meta 33
Slika 16. Unity - Asset za prototipiranje 34
Slika 17. Unity - Prva razina - prva soba 35
Slika 18. Unity - Prva razina - druga soba 35
Slika 19. Unity - Druga razina 35
Slika 20. Unity - Druga razina - prva soba 35
Slika 21. Unity - Materijal Staklo 36
Slika 22. Unity - OnClick funkcija 38
Slika 23. Unity - Glavni izbornik - početni zaslon 39
Slika 24. Unity - Glavni izbornik - odaberi nivo 39
Slika 25. Unity - Glavni izbornik - postavke 39
Slika 26. Unity - Pauza 40
Slika 27. Unity - Pauza - postavke 40
Slika 28. Unity - End menu 41
Slika 29. Unity - Zaslon s informacijama 41