# Izrada akcijske 3D računalne igre

**Zadravec, Lino**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Lino Zadravec**

# IZRADA AKCIJSKE 3D RAČUNALNE IGRE

**ZAVRŠNI RAD**

**Varaždin, 2021.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Lino Zadravec**

**Matični broj: 0016135613–R**

**Studij: Informacijski sustavi**

# IZRADA AKCIJSKE 3D RAČUNALNE IGRE

**ZAVRŠNI RAD**

**Mentor:**

Doc. dr. sc. Konecki Mario

**Varaždin, kolovoz 2021.**

*Lino Zadravec*

**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 je izrada 3D računalne igre. Žanr igre koja će u ovom radu biti opisana je akcijska igra obrane od zombija. U ovoj igri igrač će imati sposobnost, u prvom licu, kontrolirati karakter i rukovati oružjima koja će pomoći u zaustavljanju zombija. Kroz ovaj završni rad bit će opisan alat Unity, proces konceptualiziranja i razvoja igre te proces implementacije ovakve igre u alatu Unity, koji je jedan od najpoznatijih alata za kreiranje *indie* igara.



**Ključne riječi:** Akcijska igra; Unity; Razvoj igre; Programiranje; C#; Zombi

# Sadržaj

# 1. Uvod

Iako se u današnje vrijeme video igre nalaze u svim domovima diljem svijeta, početci su im bili u laboratorijima znanstvenika. 1952. godine, profesor računalnih znanosti Alexander Douglas napravio je prvu grafičku računalnu igru naziva „*OXO*“, nama bolje poznata kao „*križić-kružić*“. Ovu igru razvio je kao svoju doktorsku disertaciju na Univerzitetu Cambridge. [1]

Početak popularizacije video igara mogla bi se smatrati 1972. godina kada je Atari izmislio cijelu novu industriju oko arkadnih igara. Već 1973., Atari je počeo prodavati svoju prvu video igru Pong zbog koje je započelo masovna nabavka arkadnih strojeva u velikom broju barova, kuglana, trgovačkih centara diljem svijeta. [2]

U modernom vremenu, eksplozijom internetskih te kompjuterskih sposobnosti došlo je do velikog razvitka industrije video igara. Smatra se da u svijetu barem 1,5 milijarda ljudi s pristupom internetu igra video igre. [2] Zbog ovoliko velikog broja igrača može se zaključiti da je i tržišna vrijednost veoma velika. Prema [3] svjetska vrijednost tržišta video igara 2020. bila je 155 mlrd. američkih dolara što je približno 1 bln. hrvatskih kuna.

Pošto video igre donose velike količine novaca, divovi video igara kao što su Sony, Tencent, Nintendo, Activision te ostali, podižu standarde igara do točke gdje novi individualni programeri imaju probleme razvijanja igara do minimalnih očekivanja prosječnog igrača.

Upravo iz ovog razloga individualni programeri odlučuju koristiti alate zvane „*game engine*“ kao što je Unity. Ova vrsta alata omogućuje ubrzani razvoj video igara na način da pruža platformu s već puno implementiranih svojstava (eng. *feature*) koje su korištene u velikoj količini video igara te nam time štedi vrijeme koje bismo morali trošiti na njihov razvoj. [4] Neke od ovakvih svojstava su primjerice: 3D rasvjeta, sjene i fizika koja je krovni pojam za veoma veliku količinu svojstava.

# 2. Unity

Unity je višeplatformski softver za razvoj igara razvijen od strane Unity Technologies. Riječju višeplatformski označava se da je za ovaj softver većina Unity API i projektne strukture jednaka kroz sve podržane platforme: iOS, Android, Windows desktop, Mac OS desktop, Linux desktop i WebGL, što znači da ponekad projekt može biti ponovno izgrađen kako bi radio i na drugom uređaju, no pri tom se mora prilagoditi nekakvim osnovnim razlikama u hardveru te se moraju implementirati drugačije metode ukoliko je to potrebno. Primjerice, prilikom mijenjanja igre s Windows desktop aplikacije u Android, obično je potrebno prilagoditi igračeve kontrole s tipkovnice na gumbove koji će biti prikazani na zaslonu i pritisnuti pomoću ekrana na dodir. [5]

Unity je prvi puta najavljen i izdan u lipnju 2015. godine na Apple-ovoj svjetskoj konferenciji *developera*. Bio je prikazan kao Mac OS ekskluzivan softver za razvoj igara, što ga je u početku proslavilo pošto je bio jedan od prvih ovakvih softvera za Mac OS. Kroz vrijeme, Unity je proširio podršku i na ostale platforme koje su navedene u prošlom odlomku. [6]

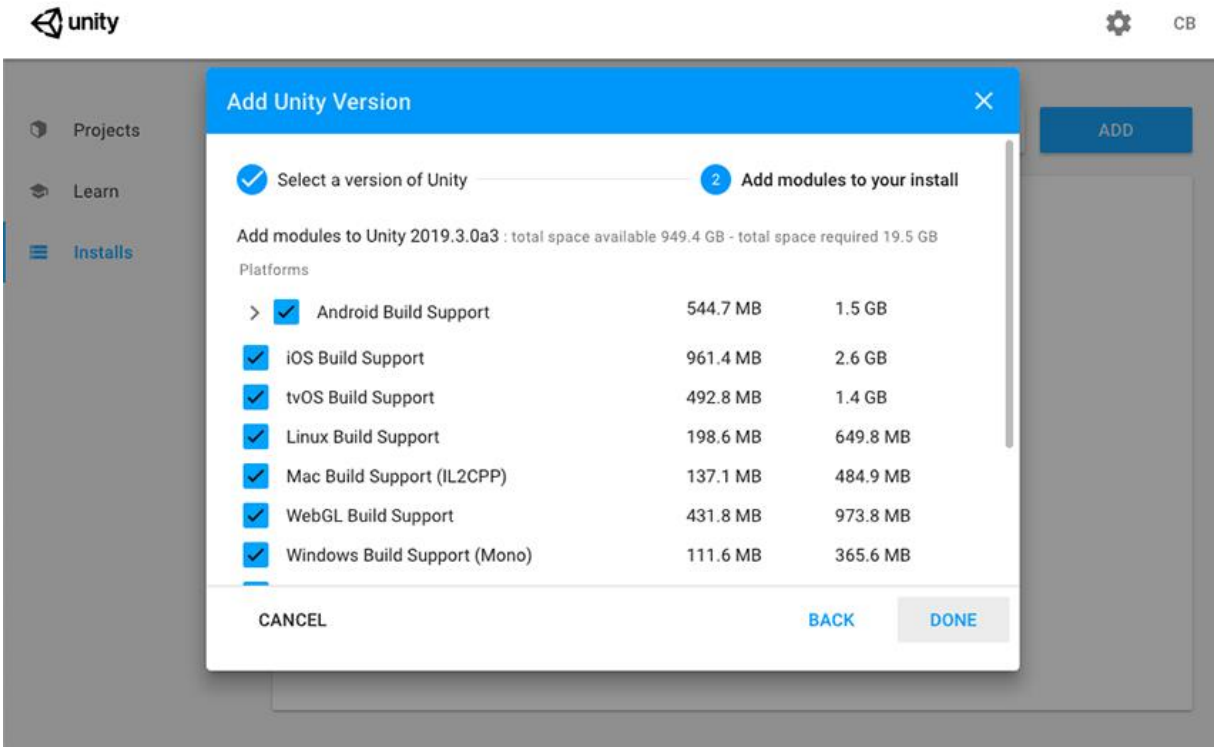
Unity većinski koriste mali programeri koji žele razvijati igre individualno, no među njima našao se i studio Blizzard Entertainment koji je razvio igru Hearthstone. Unity su koristili upravo zbog njegove sposobnosti *crossplatforminga*. [7] Izuzev studija Blizzard Entertainment, mnogo malih studija se također uspjelo proslaviti razvojem igara u ovom alatu. Neki od najpoznatijih su: Pokemon GO, Beat Saber, Hollow Knight, Rust i Temple Run. Kod ovih navedenih igara može se zamijetiti koliko se razlikuju prema svojim žanrovima i platformama. Navedene igrice su redom: *augmented reality*, *virtual reality*, 2D desktop igra, 3D desktop igra, i 3D mobilna igra. Upravo ova svestranost je jedna od najjačih strana Unityja.

## 2.1. Pripremanje Unityja

Kako bi se započeo razvoj u Unityju, potrebno je prvo instalirati Unity korištenjem Unity Huba. Ovaj alat se koristi za menadžment svih Unity projekata i instalacija koje su na korisnikovom računalu. Kako bi se instalirao Unity, potrebno je posjetiti njihovu web-stranicu i kliknuti na gumb „*Get started*“. Ovaj gumb zatim otvara stranicu s mogućim opcijama preuzimanja Unityja.

Za individualne osobe Unity je besplatan za korištenje, a plaćati se mora tek ukoliko se na igrama zaradi više od 100 tisuća američkih dolara unutar 12 mjeseci. Postoje i opcije za timove ili industrije, no u njih nećemo dublje ulaziti.

Nakon instalacije Unityja dobro je odmah instalirati i potrebne module koji će se koristiti. Na sljedećoj slici možemo vidjeti kako ovaj zaslon instalacije izgleda:

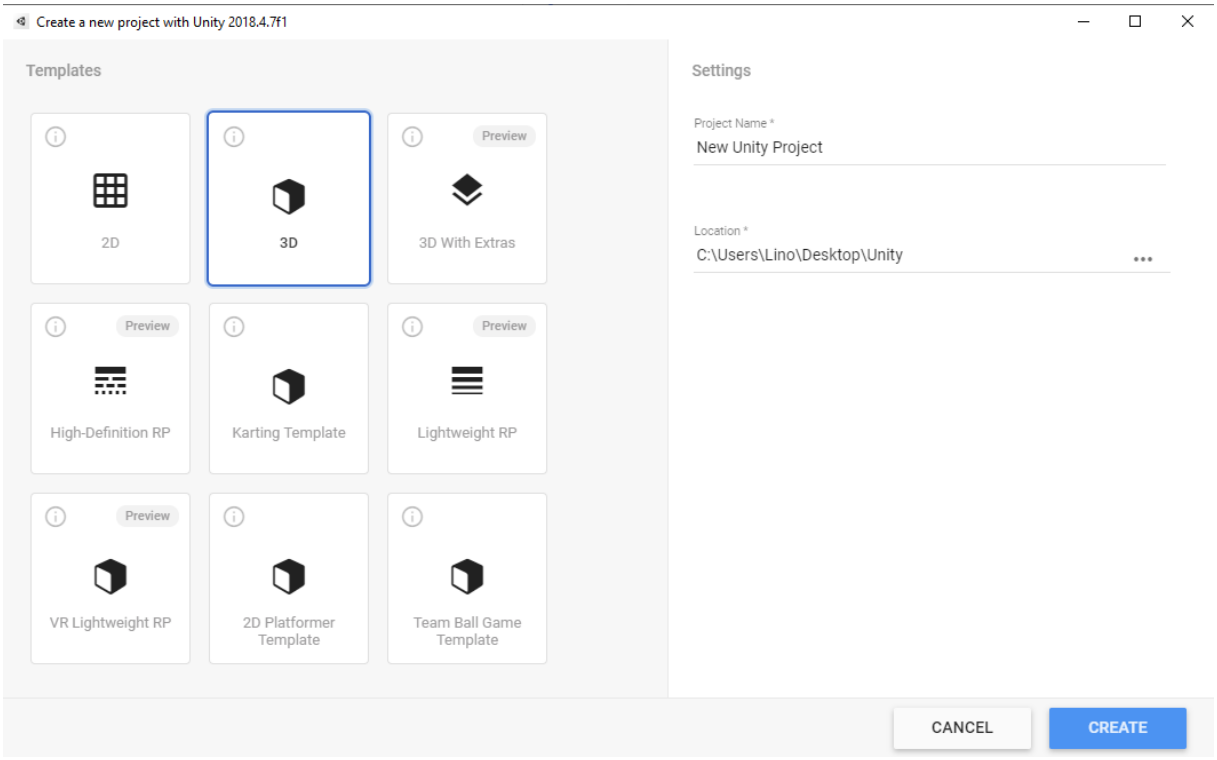



Slika 1: Instaliranje modula (Izvor: Unity Documentation, 2021)

Na Slici 1. možemo vidjeti da je ponuđeno više modula. Svaki od ovih modula zaslužan je za jednu platformu za koju će se igre graditi (eng. *build*), što je pojam koji je sličan kompajliranju, no koji je ipak malo širi. Na ovom zaslonu mogu se odabrati samo one platforme za koje korisnik planira razvijati igru. Ukoliko se korisnikov izbor promijeni moguće je dodatno instalirati i nove module kasnije. [8]

## 2.2. Stvaranje projekta

Kako bismo stvorili novi projekt potrebno je otvoriti Unity Hub koji je instaliran zajedno sa Unityjom i pritisnuti na gumb „*New*“. Nakon pritiska na „*New*“ dolazimo do izbornika koji nam prikazuje predloške za postavke projekata:




Slika 2: Izbornik kreiranja projekta (Izvor: Vlastita slika zaslona, 2021)

Na ovom izborniku prikazani su predlošci za projekte. Oni sadrže standardne postavke i prate pravila opće prakse za taj tip projekta.

U tekstualno polje „*Project Name*“ potrebno je upisati ime projekta što će stvoriti novu mapu na računalu s tim imenom. U ovoj mapi nalazit će se sve datoteke povezane s tim projektom.

U izborniku „*Location*“ potrebno je odabrati mjesto gdje će se kreirati projektna mapa. Klikom na „*Create*“ kreira se projektna mapa sa svim potrebnim datotekama i otvara se Unity urednik za razvoj igara. [9]

## 2.3. Unity korisničko sučelje

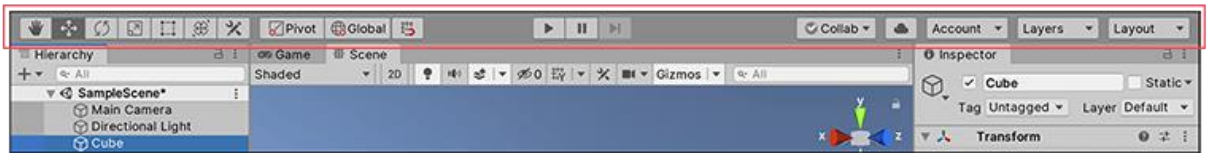
Kako bismo znali rukovati  Unity urednikom potrebno je u potpunosti poznavati Unity sučelje. Boljim poznavanjem korisničkog sučelja možemo si uvelike olakšati i ubrzati rad u bilo kojem softveru. U ovom odlomku bit će pomno objašnjeni svi dijelovi korisničkog sučelja. Na sljedećoj slici možemo vidjeti Unity urednik koji je segmentiran na dijelove korisničkog sučelja:


Slika 3: Dijelovi korisničkog sučelja (Izvor: Unity Documentation, 2021)

Na prethodnoj slici možemo vidjeti Unity urednik podijeljen na više segmenata prema slovima. Svako slovo predstavlja drugi segment koji će zasebno biti opisan u sljedećim podnaslovima. [10]

### 2.3.1. Alatna traka

Alatna traka nalazi se na vrhu Unity urednika (segment „*A*“ na slici 3.). Ona nije prozor i jedini je dio Unityjovog sučelja koji nije moguće premještati na zaslonu:




Slika 4: Alatna traka Unity urednika (Izvor: Unity Documentation, 2021)

Na alatnoj traci možemo primijetiti da se nalazi više grupa odvojenih kontrola. Svaka od njih zadužena je za određeni dio uređivanja.

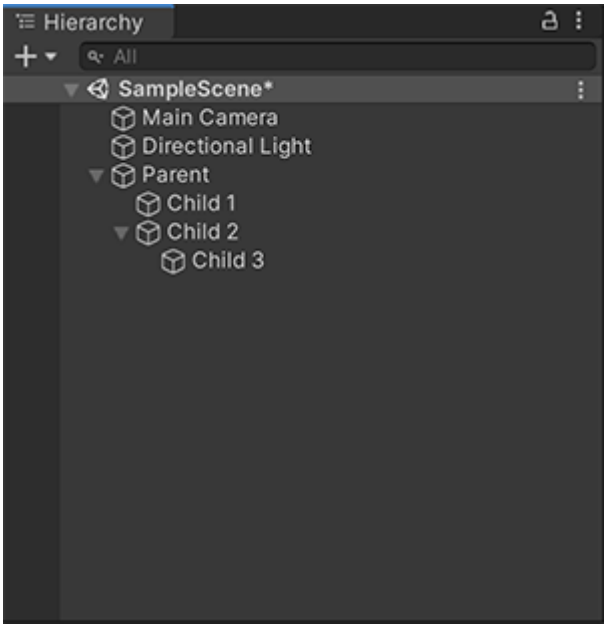
Najbitnija grupa je najljevija na slici 4. Naime, ona je zadužena za mijenjanje *Transform* komponente svakog objekta. Ova komponenta bit će bolje opisana u jednom od sljedećih poglavlja, no zasada bitno je znati da se pomoću ovih alata na alatnoj traci može mijenjati: lokacija, rotacija i veličina označenog objekta.

Srednja grupa kontrola, označena gumbovima *Play* i *Pause* zadužena je za pokretanje i pauziranje igre. Pritiskom na gumb *Play*, gumb se mijenja u gumb *Stop* kojim se igra završava.

Padajući izbornik s naslovom *Layers* zaslužan je za prikazivanje određenih slojeva u sceni. Zadan početni izbor je prikaz svih slojeva, no moguće je prikazati objekte samo određenog sloja ili više odabranih slojeva.

Pritiskom na padajući izbornik s naslovom *Layout* možemo odabrati određene predefinirane aranžmane svih dijelova korisničkog sučelja, pošto gotovo svaki dio korisničkog sučelja (osim alatne trake) može biti premješten na druge lokacije na zaslonu. [11]

### 2.3.2. Hijerarhijski prozor

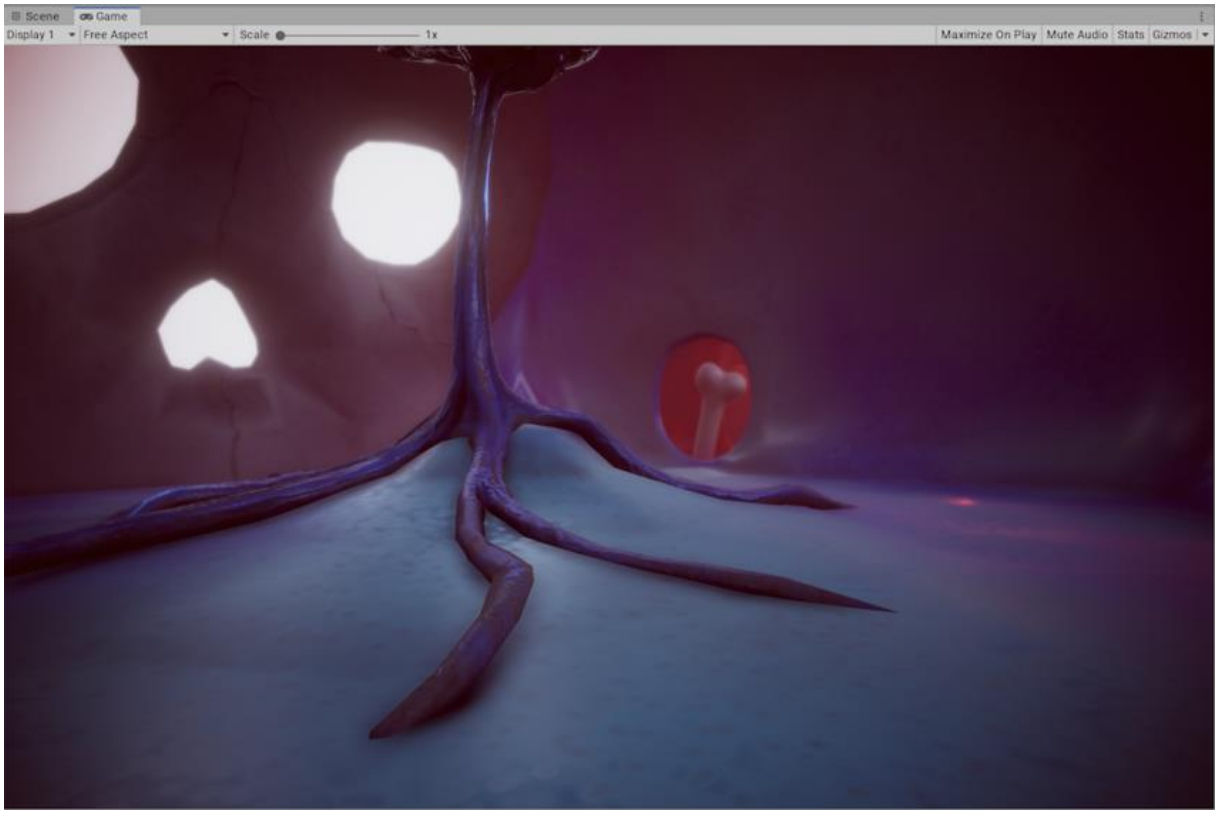



Slika 5: Hijerarhijski prozor Unity urednika (Izvor: Unity Documentation, 2021)

Ovaj prozor (slika 5.) zadužen je za prikaz svih objekata igre (eng. *game object*) u trenutnoj sceni, na hijerarhijski način. Ovaj prozor može se koristiti za sortiranje i grupiranje objekata igre kako bi se postiglo lakše upravljanje

Za stvaranje hijerarhije, Unity koristi koncept roditelj-dijete za grupiranje objekata igre. Svaki objekt igre može sadržavati ostale objekte igre kao djecu, koja će tada nasljeđivati njegova svojstva. Također, pri pomicanju, rotiranju ili povećavanju roditelja sva njegova djeca također nasljeđuju ove radnje. [12]

## 2.3.3. Prikaz igre



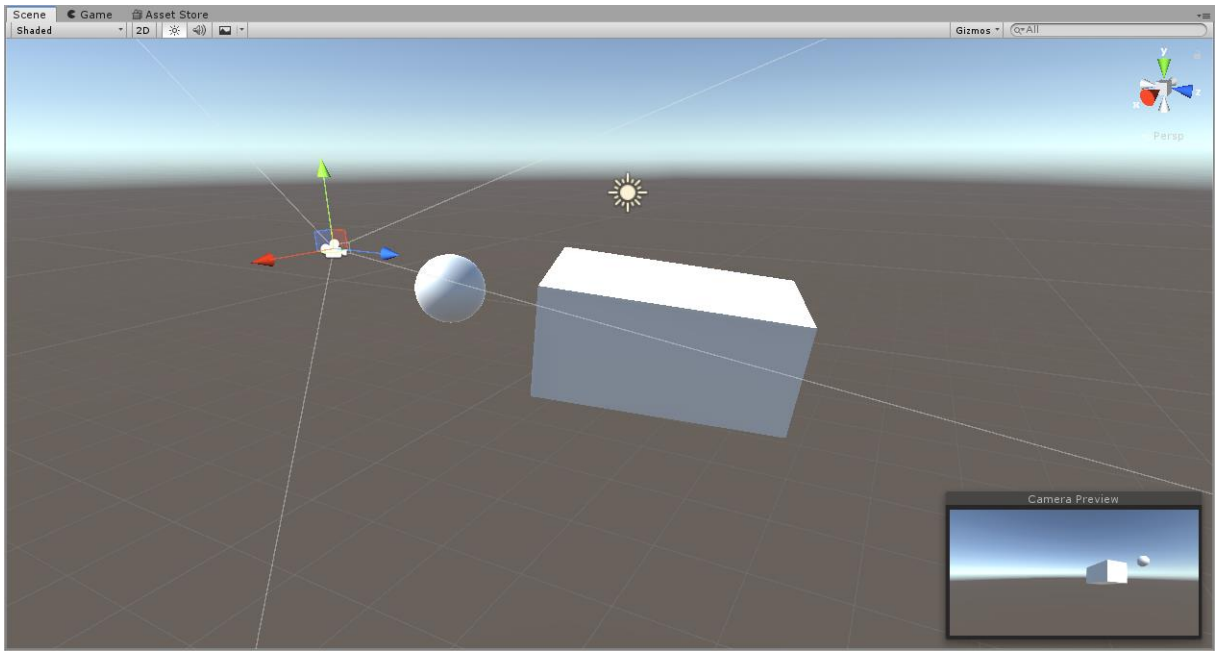

Slika 6: Prikaz igre Unity urednika (Izvor: Unity Documentation, 2021)

Prikaz igre je zapravo prikaz pogleda kamere u igri. Ovaj prikaz prikazuje što će igrač vidjeti kada igra igru kreiranu u Unityju. Kako bi igrač mogao išta vidjeti, obavezno mora biti implementirana barem jedna kamera.

Možemo primijetiti da se iznad prikaza same igre nalazi kontrolna traka. Pomoću ove trake možemo birati: koja se kamera koristi, omjer širine i visine slike te zum. U gornjem desnom kutu možemo vidjeti još opcija koje se odnose na još dodatnih postavki prikaza igre, a to su: maksimiziranje slike pri pokretanju, utišavanje zvuka te statistika. [13]

## 2.3.4. Prikaz scene

Pomoću ovog prikaza možemo se vizualno navigirati po našoj sceni. Ovo je zapravo glavni prikaz koji se koristi pri kreiranju scena pošto pomoću njega možemo pomicati, rotirati i povećavati objekte u sceni. [14] Prikaz scene ima dodatne postavke, tj. kontrolnu traku na istom mjestu kao i prikaz igre. Ovu traku možemo vidjeti iznad prikaza scene na sljedećoj slici:




Slika 7: Prikaz scene Unity urednika (Izvor: Vlastita slika zaslona, 2021)

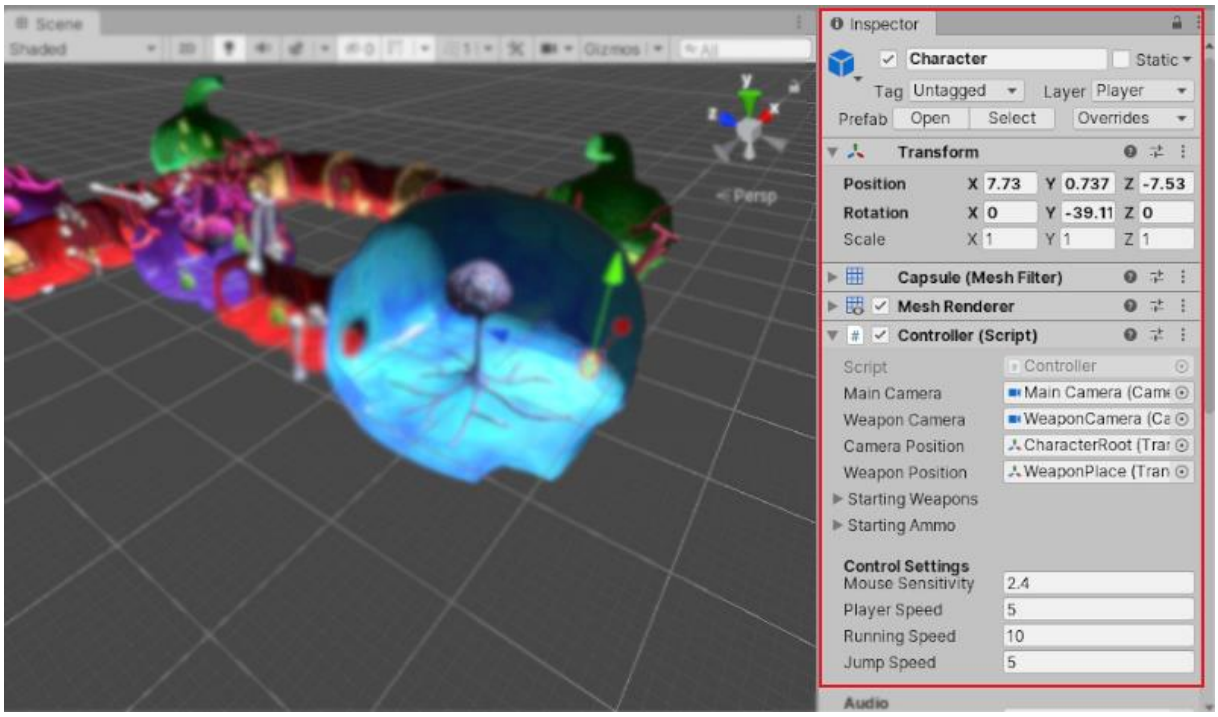
Na traci prvi navedeni padajući izbornik je *Shading mode.* On je zaslužan za odabir prikaza objekata igre u sceni. Postoji veoma mnogo različitih načina prikaza u koje nećemo ulaziti, no bitno je znati da oni postoje jer su ponekad potrebni i drugačiji načini prikaza objekata igre kako bismo točnije vidjeli potrebne informacije.

Sljedeći gumb na kontrolnoj traci prikaza scene je izmjena pogleda scene iz trodimenzionalnog u prividno dvodimenzionalan. Prilikom klika na izmjenu u dvodimenzionalni pogled, kamera se orijentira pogledom prema pozitivnoj Z-osi u Unity uredniku, tako da je X-os uperena prema desno, a Y-os prema gore.

Pokraj *2D* gumba nalaze se tri uzastopna gumba zadužena za uključivanje i isključivanje pojedinih svojstava. Prvi gumb zadužen je za uključivanje i isključivanje scenske svjetlosti, drugi gumb za uključivanje i isključivanje zvučnih zapisa, dok je treći gumb zapravo padajući izbornik kojim se mogu isključiti određena svojstva scene kao što je *skybo*x, magla, *flare*, sistemi čestica te *post-processing*. [15]

## 2.3.5. Inspektorski prozor

Ovo je jedan od najbitnijih prozora u Unityju jer se na njemu prikazuju i ažuriraju svojstva koja će nositi svaki objekt igre. Ovo uključuje sve, od boje koju će objekt poprimati, lokacije u svijetu koju će zauzimati, pa sve do fizike tog objekta, njegovih zvukova i prikaza na zaslonu.




Slika 8: Inspektorski prozor Unity urednika (Izvor: Unity Documentation, 2021)

Na slici 8. inspektorski prozor nalazi se unutar crvene crte. Prema zadanim postavkama on se nalazi s desne strane zaslona, no ovo nije pravilo.

Na desnom vrhu slike možemo vidjeti da se ovdje može imenovati objekt igre (u ovom slučaju *Character*) te da se uz to ovom objektu mogu dati dodatna svojstva kao što je potvrdni okvir *Static*, oznaka (eng. *Tag*) te sloj (eng. *Layer*).

Ispod ovih svojstava nalaze se takozvane komponente (eng. *Components*). Pri kreiranju novog objekta uključene su osnovne komponente, a to su obično sljedeće: prva komponenta je *Transform* - ovo je jedina obavezna komponenta koju mora imati svaki objekt igre u Unityju i ona je zadužena za pohranjivanje lokacije, rotacije i veličine objekta igre, druga komponenta je *Mesh filter* - ona nije obavezna na svim objektima igre, no dolazi uz veliku većinu njih pošto ona definira koji oblik u prostoru taj objekt zauzima, treća komponenta je *Mesh Renderer* - koja je zadužena za prikazivanje tog objekta na zaslonu; primjerice ukoliko kreiramo kocku kao objekt igre, onda očekujemo da se na zaslonu prikazuje kocka, a u slučaju da se komponenta *Mesh Renderer* ugasi, kocka i dalje postoji, no ona se ne prikazuje na zaslonu već je nevidljiva. [16]

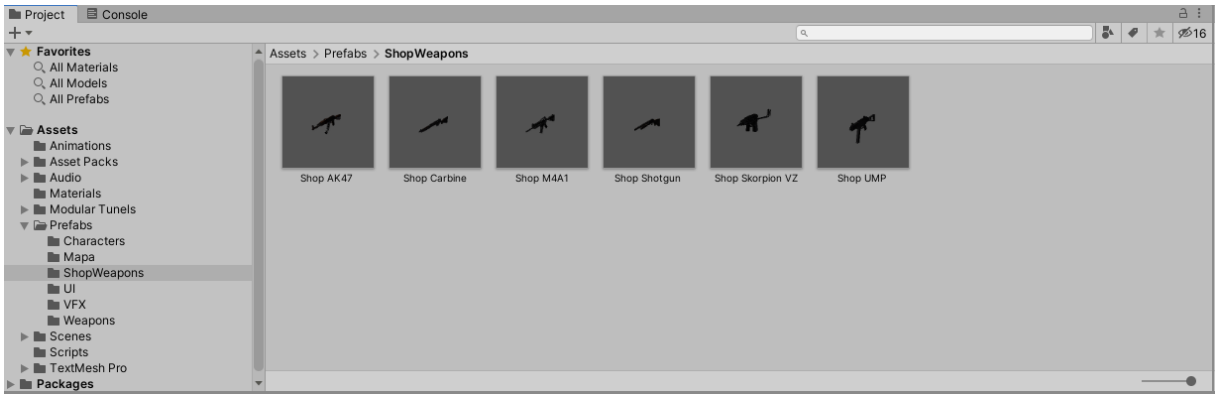
Osim osnovnih komponenata korisnik može svakom objektu igre dodati komponenata koliko god je potrebno. Jedan od najviše značajnih razloga zašto je Unity toliko popularan jest obilje komponenti koje su unaprijed isprogramirane za individualnog programera. Ovime se

štedi na vremenu jer programer ne mora utrošiti vrijeme na razvijanje određene komponente koja je potrebna u gotovo svakoj igri kao što je na primjer komponenta *Audio Source* koja se koristi za postavljanje izvora zvuka. Programer u ovom slučaju mora samo staviti komponentu na objekt igre i odabrati zvuk koji želi da taj objekt proizvodi. Naravno, na programeru je da i isprogramira događaj kada će se taj zvuk izvoditi, no programer unatoč tome i dalje štedi veoma puno vremena.

Uz unaprijed postavljene komponente, programer ima mogućnost stvoriti svoju komponentu u obliku skripte. Ovo se postiže klikom na „*Add Component*“ i zatim „*New Script*“ što u prijevodu znači, „*Dodaj Komponentu*“ i “*Nova Skripta*“.

## 2.3.6. Projektni prozor

Projektni prozor zaslužan je za prikaz svih datoteka vezanih uz otvoren projekt. On je također glavni način za navigaciju i traženje imovine projekta (eng. *asset*) koja zapravo obuhvaća sve: modele, skripte, materijale, zvukove, slike i bilo koje datoteke povezane uz projekt. [17] Na sljedećoj slici možemo vidjeti kako ovaj prozor izgleda:



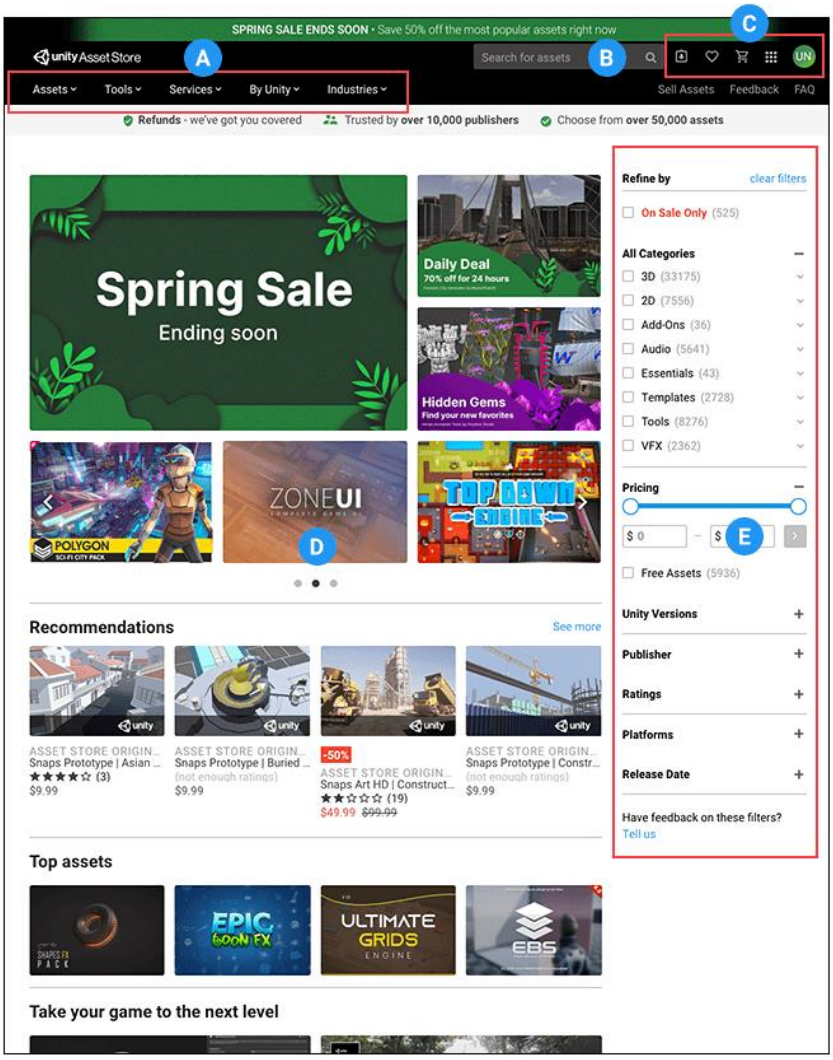

Slika 9: Projektni prozor Unity urednika (Izvor: Vlastita slika zaslona, 2021)

Možemo primijetiti da slijeva imamo izbornik koji je hijerarhijski raspoređen tako da mape postaju padajući izbornici svih njihovih sadržaja. Pritiskom na bilo koju od ovih mapa, ona se otvara i prikazuje svoje sadržaje. Na slici 9. otvorena je mapa „*ShopWeapons*“ iz osobnog projekta i možemo vidjeti da ona sadrži više modela oružja.

Pri izradi projekta bitno je biti sistematičan od početka i stavljati svaku novu datoteku u mapu s ostalim datotekama iste svrhe. Ukoliko je projektni prozor neuredan, previše vremena bit će utrošeno na pronalazak datoteka, što uvelike doprinosi neefikasnosti vremena uloženog u projekt.

## 2.4. Skladište imovine (Asset store)

Skladište imovine sadrži skladište besplatnih i komercijalnih imovina koje nude sam Unity Technologies, ali i njihovi članovi zajednice. Ovo skladište veoma je raznoliko, jer sadrži teksture, modele, animacije, zvučne zapise, skripte pa čak i potpune projektne primjere. Od Unity verzije 2020. on više nije uključen kao prozor unutar korisničkog sučelja, već je samo prisutan kao web-stranica:




Slika 10: Web-stranica skladište imovine (Izvor: Unity Documentation, 2021)

Na slici 10. segment „*A*“ zadužen je za grupiranje imovine u kolekcije. Klikom na bilo koju od ovih kolekcija otvara se padajući izbornik s više grupa koje su dio te kolekcije, čime lakše pronalazimo imovinu koja nam je potrebna.

Pomoću okvira za pretraživanje „*B*“ (eng. *searchbox*) lakše pronalazimo imovinu koja nam je potrebna na način da upišemo što tražimo. Ove pojmove treba upisivati na engleskom jeziku pošto Unity Asset Store pretražuje prema ključnim riječima koje su na engleskom jeziku. Pretraživati možemo na način da upišemo riječ ili frazu koja je opisu te imovine ili upisivanjem imena imovine.

Odjeljak „*C*“ prikazuje nam ikone koje služe kao linkovi do sljedećih područja. Redom to su: lista preuzete imovine, favoriti, vaša košarica, aplikacije skladišta imovine, vaš Unity profil.

U odjeljku „*D*“ pri početnom otvaranju stranice prikazuje se predložena imovina. To uključuje rasprodaje te prijedloge bazirane na prethodnim kupnjama. Ovaj odjeljak također služi kao mjesto gdje se prikazuje imovina nakon što se koriste odjeljci „*B*“ ili „*E*“, tj. kada se filtriraju ili sortiraju te označuju kategorije imovine.

Odjeljak „*E*“ jedan je od najkorisnijih odjeljaka jer dopušta da korisnik pregledava imovinu na način da ih filtrira prema svojim kriterijima. To može raditi određivanjem cijene koju je voljan platiti, ali i odabirom kategorija, ocjena, izdavača ili prema datumima izdavanja. [18]

U ovom završnom radu skladište imovine uvelike je doprinijelo ubrzavanju razvoja akcijske 3D računalne igre kako je i predviđeno. Pomoću skladišta imovine pronađeni su besplatni modeli oružja, zombija te *sandbox* standardni modeli za testiranje. Osim modela, Unity Technologies u svojem paketu „*Standard Assets*“ izdao je i besplatnu komponentu naziva *Rigidbody First Person Controller*, koji je zaslužan za kontroliranje pogleda kamere i kretanje igrača u igri.

# 3. Proces razvoja igre

U ovom odjeljku opisivat ćemo proces razvoja igre u alatu Unity. Vrsta igre koja će biti razvijena u ovom radu jest 3D akcijska igra obrane od zombija. Točnije, razvijat će se igra u kojoj glavni lik (igrač) oružjima zaustavlja zombije koji pokušavaju doći do igrača i ozlijediti ga udarcima. Igrač će imati sposobnosti: pomicanja, rotiranja gledišta, trčanja, pucanja te kupovanja novih oružja. Zombi, s druge strane imat će sposobnosti praćenja glavnog lika te udaranja.

Kroz sljedeće podnaslove opisat ćemo kako smo došli od novokreiranog projekta do funkcionalne igre.

## 3.1. Opći dizajn igre

Prije nego što započnemo s razvojem igre, moramo imati ideju igre koju ćemo kreirati. Bez dizajniranja i planiranja u kasnijem trenutku razvoja moglo bi doći do kontradikcija ili funkcionalnosti koje u kombinaciji nemaju smisla.

Prema videu [19] u kojemu *indie* programer objašnjava kako NE napraviti igricu, programer je započeo s velikom listom funkcionalnosti koje je želio implementirati u igru, no nije u potpunosti protumačio kako će to u igri izgledati i kako će ta igra imati smisla. Nakon par godina razvoja kada je uspio implementirati sve željene funkcionalnosti, programer je shvatio da igra uopće nije zabavna, jer ima previše stvari koje se događaju koje nisu kompatibilne. Prema istom videu glavna pouka jest: „Dobra igra započinje dobrim prototipom, koji je već zabavan za igrati, a tek tada se taj prototip proširuje dodatnim funkcionalnostima i dodacima“.

U sljedećim podnaslovima objasnit ćemo ukratko glavne dijelove dizajna igre.

### 3.1.1. Jezgra igre

Jezgru igre (eng. *Core gameplay mechanic*) bitno je uspostaviti odmah na početku.

Jezgrom igre smatra se radnja koju igrač kroz igru radi konstantno od samog početka. Primjeri ovoga su: trčanje, let, pucanje itd. Kako bismo uopće znali započeti određivanjem jezgre igre, moramo odrediti žanr igre koju ćemo dizajnirati te nakon što smo to učinili, sljedeći korak je izmisliti jednu ili dvije utjecajne funkcionalnosti. U ovome dijelu dizajna ne bi smjelo biti komplikacija jer kada se radi o najutjecajnijim funkcionalnostima one bi trebale ostati jednostavne. [20]

U našem slučaju žanr koji ćemo raditi je akcijska igra, dok je jezgra igre trčanje i pucanje. Ovime smo odabrali našu jezgru igre, koja čini kostur našeg finalnog programa. Oko nje bismo morali razvijati ostale funkcionalnosti i sve funkcionalnosti morale bi biti komplementarne upravo toj jezgri.

### 3.1.2. Iskustvo igrača

Slično pojmu korisničko iskustvo, iskustvo igrača (eng. *Player experience*) jest iskustvo kroz koje igrač prolazi kada igra igru, tj. na koji je način naša igra primljena od strane igrača. Iskustvo igrača može se podijeliti na sljedećih šest kategorija:

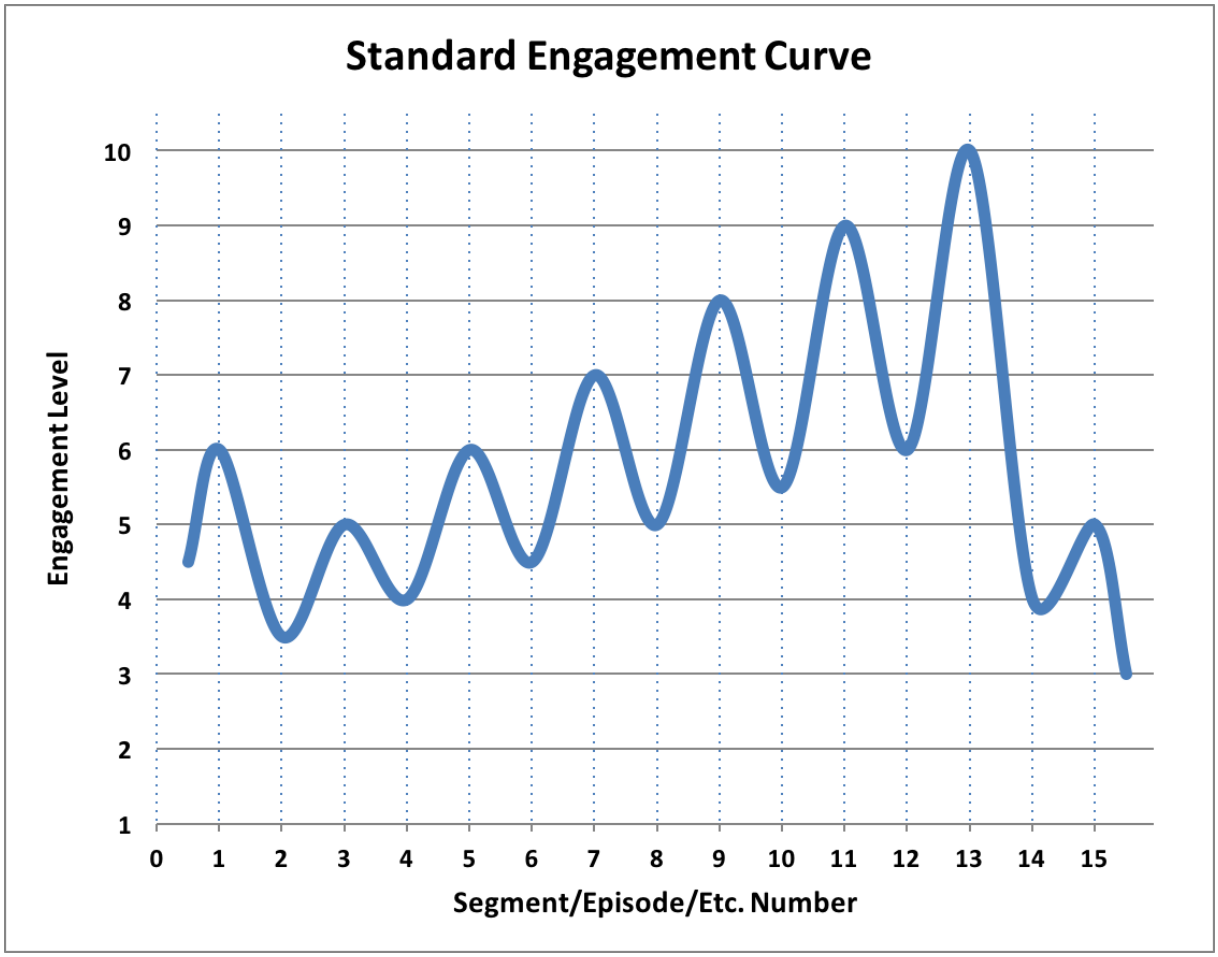
1. Motivacija
2. Smisleni izbor
3. Ravnoteža
4. Upotrebljivost
5. Estetika
6. Zabava

Svih šest kategorija bitni su koraci pri dizajnu igre, jer njime odlučujemo koji tip igrača želimo zaintrigirati našom igrom. Kroz sljedeće odlomke objasnit ćemo svaku pojedinu kategoriju.

Motivacija uvijek odgovara na pitanje: „Zašto korisnik igra našu igru?“. Ovdje postoji puno različitih razloga koji sežu od isprobavanja novog žanra igre, preko bijega od stvarnosti pa sve do učenja u obrazovne svrhe. Bez obzira na igru, motivacija se dijeli na najmanje sljedeće dvije kategorije: interes i demonstracija sposobnosti. Interes se većinom odnosi na igre za zabavu, gdje igrač najviše želi samo doživjeti što mu igra nudi. S druge strane, demonstracija sposobnosti je drugačija, ona proizlazi iz ljudske potrebe da se natječe s ostalim igračima. Ovdje motivaciju čini želja za boljim igranjem i napredovanjem.

Smisleni izbori su načini kojim igrač demonstrira svoju kompetenciju. Bez izbora, igrači ne mogu utjecati na ono što igraju čime se gubi na uživljenosti u igru. Za ove izbore ujedno je bitno i da su oni smisleni, to jest da imaju utjecaj na daljnji ishod igre. Ovo omogućava igraču da stvara svoje taktike i strategije te da planira svoje korake kroz igru, no dizajner bi i dalje trebao biti taj koji odlučuje koliko velik utjecaj različiti izbori imaju na igrača.

Ravnoteža je činjenica da dizajner treba kontrolirati uživljenost igrača u igru. Na sljedećem grafu vidimo crtu uživljenosti koja balansira trenutke visokih intenziteta i uživljenosti s onima s manjom uživljenošću. Na apscisi nalazi se prolazak vremena kroz igru, dok je na ordinati visina uživljenosti.




Slika 11: Standardni graf uživljenosti

Iz grafa možemo iščitati da kroz igru uživljenost od početka do kraja lagano raste, no da ujedno i krivuda. Ovo je veoma bitno jer ukoliko stvorimo igru u kojoj je igrač uvijek uživljen i konstantno napet, igrač će se polagano izmoriti i time izgubiti interes za daljnjom igrom. S druge strane, ukoliko stvorimo igru u kojoj uopće nema uživljenosti, igrač će također izgubiti interes. Upravo zbog ovog razloga veoma je bitno pronaći ravnotežu. Najbolje igre za balansiranje uživljenosti su one koje se smatraju: „lagane za naučiti, ali teške za usavršiti“. To su primjerice igre kao šah, go ili primjer video igre tetris. [22]

Upotrebljivost se odnosi na jezgru igre što obuhvaća tri pitanja: „Što se događa u igri?“, „Što se događa igračima?“ i „Koje sposobnosti igrač ima nad igrom?“. Ovo obuhvaća igračeve sposobnosti kroz igru te shvaćanje cilja igre i kako doći do njega. Upotrebljivost postižemo na različite načine, dok je u nekim igrama fokus samo na izvršavanju cilja bez interakcije između igrača i protivnika (npr. Bingo). Postoje i igre u kojima radnje koje protivnici poduzimaju

direktno izazivaju proturadnje od strane igrača što se odnosi na veliku većinu računalnih kompetitivnih internetskih igara (npr. Starcraft).

Estetika kao što i sama riječ označava, odnosi se na doživljaj igre kroz ljudska osjetila. Iako postoji pet osjetila, većina igara ne koristi ih sve. Najviše pažnje pridaje se vidu i sluhu. Treće osjetilo po redu bilo bi osjetilo dodira koje možemo primjerice osjetiti na Playstation 4 kontrolerima kada vibriraju kod određenih događaja. Tehnologija za osjet njuha i okusa u dizajnu igrica nažalost još nije dovoljno napredna zbog čega igrice ne pridaju pažnju ovim osjetilima. Estetika definitivno ima velik utjecaj na doživljaj igrača, no u razvitku *indie* igara ona obično ne dobiva dovoljno pažnje zato što sam programer nije dovoljno sposoban kreirati nešto estetski ugodno.

Zabava kao zadnja kategorija ima veoma velik utjecaj na doživljaj igre. Ona je subjektivna i proizlazi iz iskustva igrača. Na zabavu dizajner ne može direktno utjecati, no ona je kombinacija prije navedenih kategorija koje zajedno tvore osjećaj zabave u igraču. [21]

## 3.2. Dizajn igre ovog završnog rada

Pri početku razvoja igre, sve je započelo od predefiniranog žanra igre, a to je „akcijska 3D računalna igra“. Od tog trenutka došlo je do odluke da će u ovom radu jezgra igre biti pucanje neprijatelja. Iz ove jezgre postoji mnogo raznih podžanrova koji se mogu ostvariti što bi uključivalo: *Shoot 'em up*, *Run-and-gun shooter*, *Shooting gallery, Light gun shooter*, *First-person shooter* i *Third-person shooter* koji nisu prevedeni zbog gubitka u prijevodu. [23]

U našem slučaju igra će biti *First-person shooter*, tj. pucanje iz prvog lica ili ubuduće FPS. Točnije radit će se o FPS igri beskrajne obrane od zombija koja je pronašla inspiraciju u igri „Call of Duty: Black Ops – Zombies“.

Motivacija igre bazira će se na demonstraciji sposobnosti koja će igrača motivirati da pokušava napredovati i doći dalje nego što je uspio u svojem prošlom pokušaju.

Igrač će imati smislene izbore u obliku izbora lokacije na kojoj će se braniti od zombija te koja će oružja birati za obranu. Ova oružja razlikovat će se po svojim svojstvima gdje će svako oružje u jednom pogledu biti smislen izbor barem u jednom dijelu igre.

Kako bismo pratili ravnotežu uživljenosti igrača u igru, beskrajni zombiji dolazit će u valovima između kojih će biti mala vremenska pauza. Prilikom ove pauze igrač će moći odmoriti od intenziteta obrane te promijeniti svoju lokaciju i oružje ukoliko je to potrebno.

Upotrebljivost, koja obuhvaća cilj igre i kako doći do njega, u ovoj igri jest preživjeti što dulje prilikom napada zombija koji svakim valom postaju sve jači i višebrojniji. Ovo se radi na

način da se zombiji pucaju oružjem koje je igraču pri ruci. Prilikom pucanja zombija igrač dobiva novac koji može zamijeniti za novo i bolje oružje. Kako bi igrač došao do boljeg oružja, mora kupiti i novi prostor. Ukoliko igrač odluči ne kupiti novi prostor, zaglavio je s oružjem koje neće biti dovoljno dobro za ubijanje jačih zombija.

Estetika ove igre velikim dijelom bit će *outsourceana*, tj. fontovi, zvukovi, 3D modeli igre i mapa kupljeni su ili pronađeni kao besplatni za korištenje na internetu. Ovo je veoma česta praksa u indie igrama zato što programer obično nije dovoljno sposoban ili talentiran za kreirati nešto estetski lijepo u kratkom vremenu. Estetska tema koja će se provlačiti kroz igru jest tamna i strašna atmosfera, a proizlazi iz cilja igre - pucanje zombija.

S obzirom na to da je zabava stvar subjektivne prirode, nadamo se da će ona proizaći iz ostalih aspekata dizajna ove igre i da će doći prirodno.

## 3.3. Mehanike igre

Igrač ima mogućnost kretanja po mapi tipkama „W“, „A“, „S“, „D“ koje su standard FPS igara. Tipkom „Shift“ kretanje se ubrzava za određeni koeficijent pri čemu igrač prelazi u stanje trčanja. Razmaknicom igrač može skočiti kako bi preskočio određene zapreke ukoliko ih ima.

Lijevim klikom na mišu igrač može pucati oružjem koje je trenutno aktivno. Pri jednoj od pušaka igrač može desnim klikom zumirati kako bi lakše naciljao udaljene zombije. Zumiranje je implementirano samo kod puške *Enfield* jer je ona namijenjena za dalekometno streljaštvo. Pri pucanju igrač pomoću nišana na zaslonu cilja protivnike. Razlikuje se hitac u glavu i hitac u ostatak tijela, gdje hitac u glavu radi veću štetu te donosi više novaca.

Igrač može imati maksimalno dva oružja: primarno i sekundarno. Sekundarno oružje je pištolj koji je dostupan igraču kroz cijelu igru i koji ima beskonačnu količinu metaka. Loša strana pištolja je da on donosi malu količinu novaca i nanosi veoma malu štetu zombijima, zbog čega bi se trebao koristiti samo ako igraču ponestane metaka u primarnom oružju. Primarno oružje ima konačnu količinu metaka, no radi puno više štete od pištolja. Igrač ne započinje s nijednim primarnim oružjem, a kako bi ga dobio mora ga kupiti pomoću novca.

Zombiji na beskonačnu udaljenost lociraju i love igrača. Kada dođu na dovoljnu blizinu, udaraju ga te nanose štetu na životne bodove igrača. Igrač započinje sa 100 životnih bodova. Ukoliko igrač ima manje od maksimalne količine životnih bodova, nakon pet sekundi životni bodovi se počinju vraćati tako dugo dok ne dođu ponovno do maksimalne količine.

Zombiji dolaze u valovima što znači da se svake 42 sekunde instancira novih N zombija, gdje je količina određena ovisno o trenutnom broju vala. Svakim valom povećavaju

se životni bodovi zombija, a samo pri određenim valovima povećava se količina i šteta koju zombiji nanose.

Kako bi igrač imao šanse protiv sve jačih i jačih zombija, pri pogodcima zombija dobiva novac koji može zamijeniti za bolja oružja. Prodavaonice oružja nalaze se na određenim statičnim mjestima na mapi i svaka prodavaonica prodaje samo jedno oružje. Kako bi ga igrač kupio, mora doći blizu prodavaonice i kliknuti na tipku „F“.

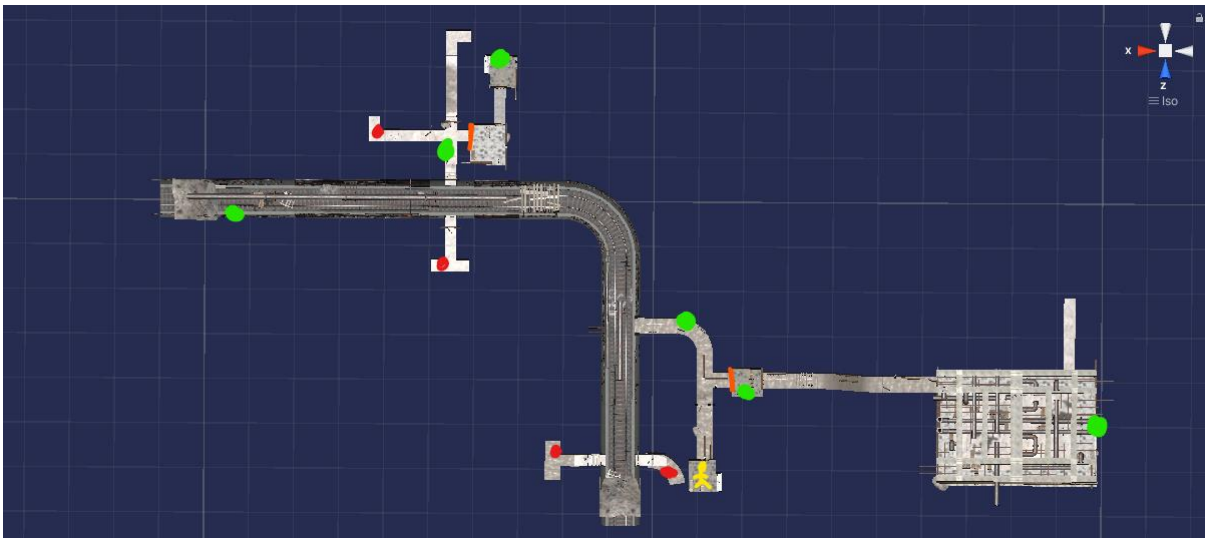
Osim oružja, igrač može svoj novac uložiti i u otvaranje novih vrata. Kada se uloži novac u vrata, pritiskom na tipku „F“ ona se otvaraju i otkrivaju novi dio mape koji prije nije bio dostupan. Na novim dijelovima mape nalazi se i novo, bolje oružje koje će igraču pomoći dulje preživjeti.

Igra završava samo ako igraču životni bodovi padnu na nulu ili manje, a u suprotnom igra se može nastaviti beskonačno s beskonačnom količinom valova zombija. Kada igrač umre, prikazuje mu se izbornik u kojem odlučuje želi li pokušati ponovno ili ugasiti igru.

## 3.4. Izrada mape

Kao što je prije navedeno, mapa ove igre je *outsourceana* kako igra ne bi gubila na svojoj estetici. Mapa je bila kupljena sa stranice „https://www.cgtrader.com/“ kojoj je svrha prodavanje 3D modela objekata, pa čak i cijelih mapa.

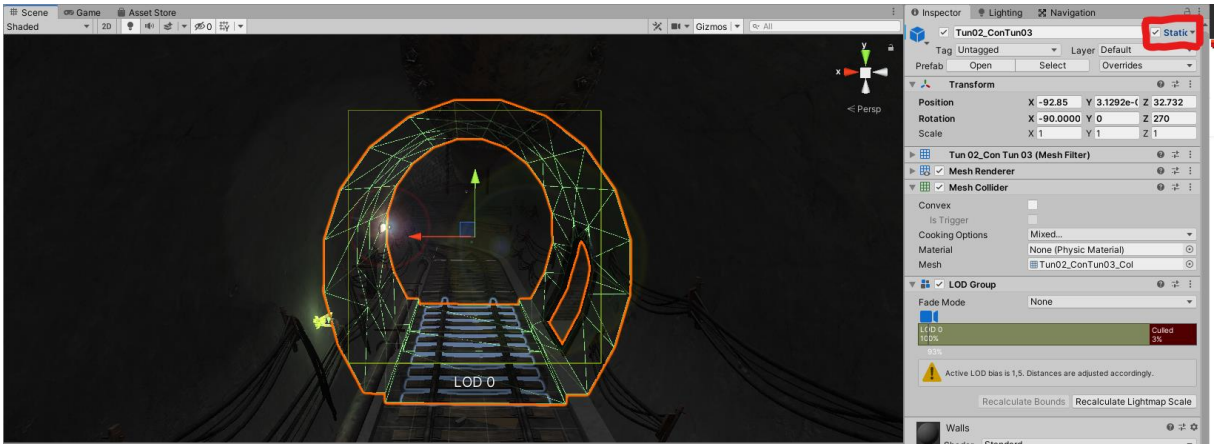
Kako bismo uvezli mapu formata *.unitypackage* u Unity, potrebno je pritisnuti desni klik na projektni prozor i odabrati „*Import Package*“ te našu datoteku mape. Ova mapa je stigla s demo scenom koja je već imala raspoređene objekte u mapu, no mapa je bila prevelika za ciljani žanr ove igre. Upravo zbog toga, skupine objekata igre bile su premještene na druge lokacije na mapi. Točnije cijele sobe sa svim njihovim objektima igre bile su zajedno pomaknute, čime je bio postignut novi tlocrt koji se može vidjeti na sljedećoj slici.



Slika 12: Tlocrt mape (Izvor: Vlastita slika zaslona, 2021)

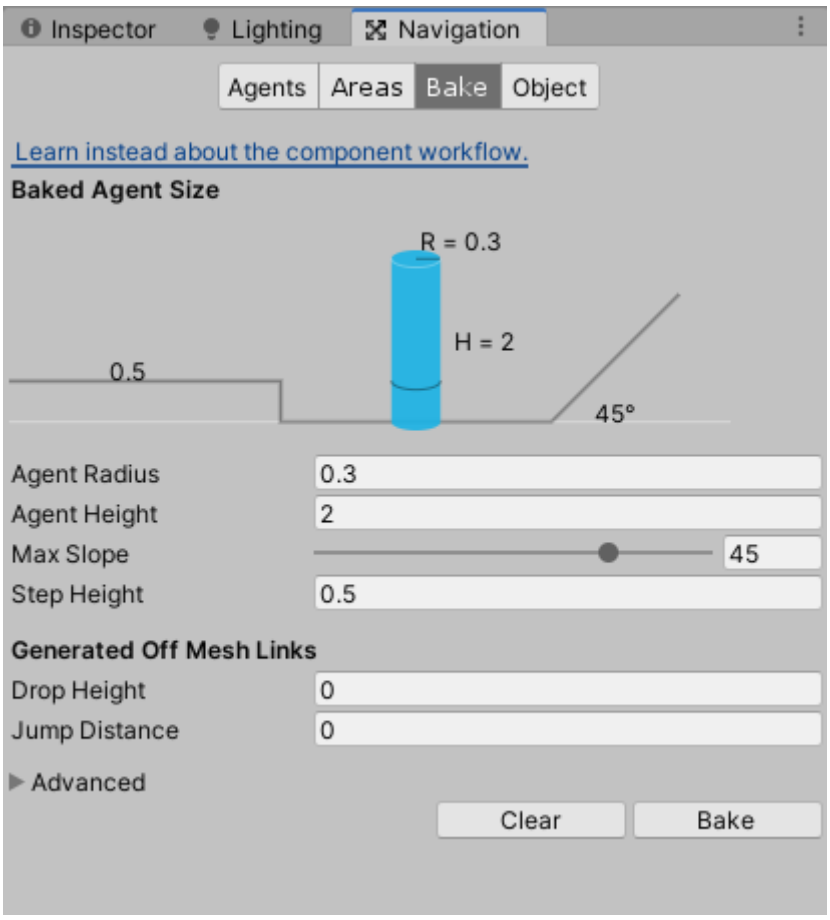
Prilikom snimanja zaslona ove slike korišten je izometrički pogled na mapu koji je prigodan za slike tlocrta. Na ovom tlocrtu možemo vidjeti točke i crte različitih boja koje će biti objašnjene. Žuti čovječuljak je mjesto na kojem igrač započinje svoju igru. Svijetlozelene točke na tlocrtu označavaju mjesta na kojima igrač može kupiti novo oružje. Crvene točke označavaju mjesta na kojima se pojavljuju zombiji, a narančaste crte su vrata koja se kupnjom otvaraju i proširuju prostor igre.

Kako bismo na mapi mogli kreirati (eng. *bake*) *NavMesh* po kojem će se zombiji znati kretati i navigirati, potrebno je prvo staviti svojstvo *static* na sve objekte igre koji će utjecati na kretnju zombija što uključuje i pod po kojem se zombiji kreću, ali i sve zapreke. Na sljedećoj slici možemo vidjeti kako odabrati svojstvo *static* na objektu:




Slika 13: Static svojstvo objekta igre (Izvor: Vlastita slika zaslona, 2021)

Nakon što je svojstvo *static* označeno na svim objektima igre, potrebno je u inspektorskom prozoru odabrati grupu „*Navigation*“ te podgrupu „*Bake*“.



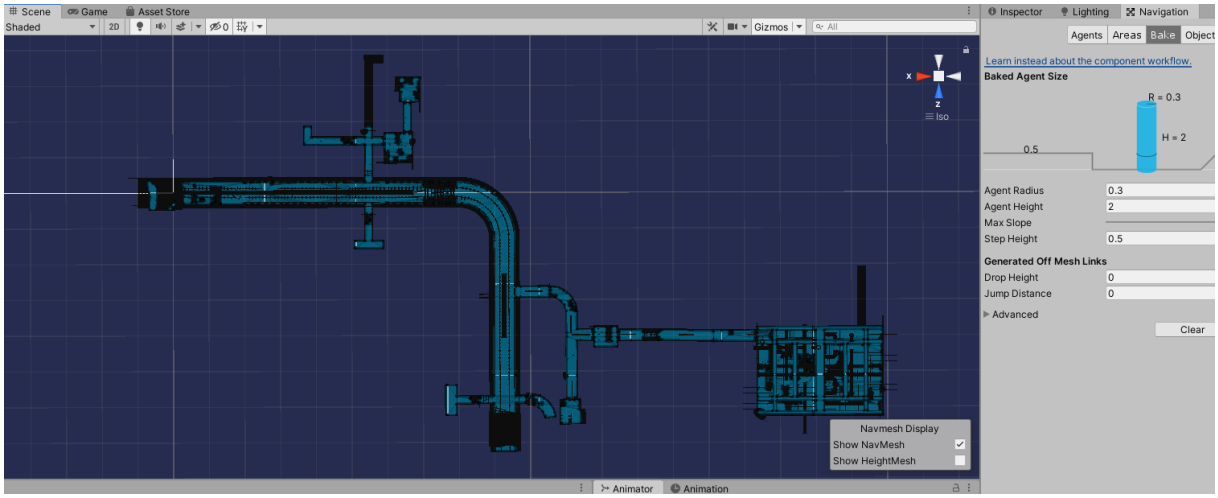

Slika 14: *Navigation bake* obrazac (Izvor: Vlastita slika zaslona, 2021)

U ovom obrascu moramo odabrati svojstva kako bi se generirao dobar NavMesh gdje nam grafički prikaz služi kao pomoć shvaćanja funkcija svakog svojstva. Ovim izbornikom definiramo svojstva koja će imati NavMeshAgent, u našem slučaju zombi. Sva polja ovog obrasca odnose se na fizička svojstva zombija te njihovih sposobnosti u smislu kretanja.

Prvo svojstvo je radijus koji označava širinu zombija što je potrebno kako zombi ne bi hodao tik do zidova na način da polovica zombija izvire kroz drugu stranu zida. Drugo svojstvo je visina i ona će generirati NavMesh samo na mjestima koja su manje visine od NavMeshAgenta kako bi on mogao proći kroz ta mjesta bez da mu glava prolazi kroz zid. Treće svojstvo definira agentovu sposobnost kretanja po tlu s nagibom čime označavamo koliko strme padine i uspone agent može prohodati. „*Step Height*“ još jedno je bitno svojstvo koje označava koliko visoke stepenice agent može prohodati. Ovo je bitno primarno za stepenice i

rubnike koji su u suštini veći od „*Max Slope*“ zato što su pod kutom od 90 stupnjeva. Sljedeća dva svojstva nama nisu bitna zato što zombiji neće imati ove sposobnosti.

Kada su odabrana sva svojstva možemo pritisnuti „*Bake*“. To je gumb za generiranje NavMesha koji odgovara tim svojstvima. U našem slučaju NavMesh izgleda ovako:




Slika 15: NavMesh mape (Izvor: Vlastita slika zaslona, 2021)

Na slici 15. *NavMesh* je svijetloplave boje i označava gdje se zombi može kretati. Kako bi se *NavMesh* prikazao na zaslonu u scenskom pogledu „*Navigation*“, podgrupa mora biti otvorena. Još jedna stvar koja se može zamijetiti na ovoj slici je da postoje dva crna hodnika, od kojih je jedan na desnoj strani slike. Crna boja proizlazi iz činjenice da na ovim mjestima nema *NavMesha*, čime smo osigurali da ovdje zombiji neće moći hodati. Uz to, svi rubovi i hodnici tunela nisu svijetloplavi što je izravno uzrokovano svojstvom „*Agent Radius*“ kojeg smo prije generiranja postavili na vrijednost 0.3 kako zombiji ne bi  hodali po rubovima sobe. [24]

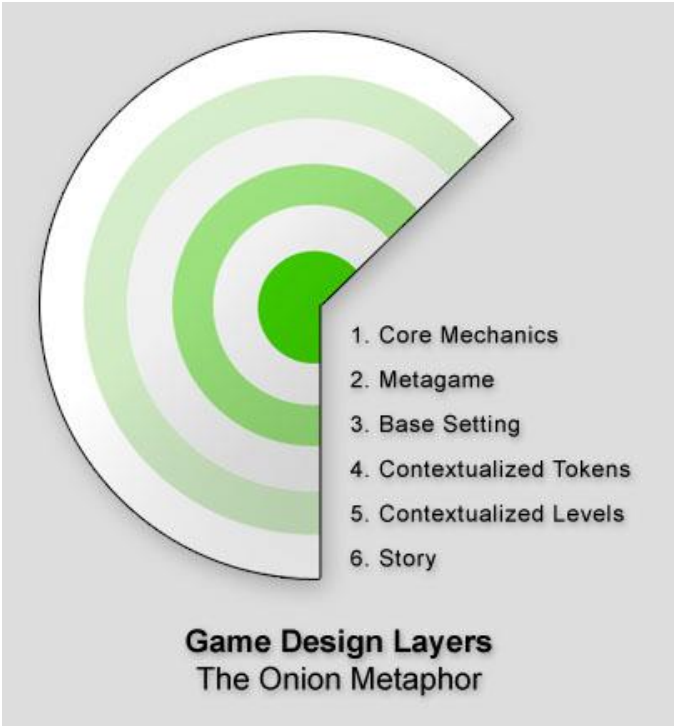
## 3.5. Implementacija igre

Implementacija koda u igru jedan je od najbitnijih dijelova igre, zato što je kod taj koji čini igru dinamičnom i igrivom. Prije nego se započne kodiranje, mora se postaviti određeni redoslijed razvijanja skripti, točnije potrebna nam je lista prioriteta. Ovo je ključno jer ukoliko smo ograničeni vremenskim okvirom, tada nećemo imati vremenskih problema pri implementiranju najbitnijih mehanika igre, zato što je njihov prioritet najviši i one su iz tog razloga bile prve implementirane.

### 3.5.1. *Onion* dizajn

*Onion* (hrv. luk) dizajn nazvan je ovim imenom zato što često opisuje slojevitost igre, gdje je srž igre centralni sloj, a kretanjem prema vanjskim slojevima više se odmičemo od jezgre igre. Cijela bit ove vrste dizajna je raspored resursa koji seže od jezgre igre pa sve do same priče. Recept za neuspjeh prema [25] jest u sve slojeve uložiti po malo vremena kako bismo imali najbolje od oba svijeta.

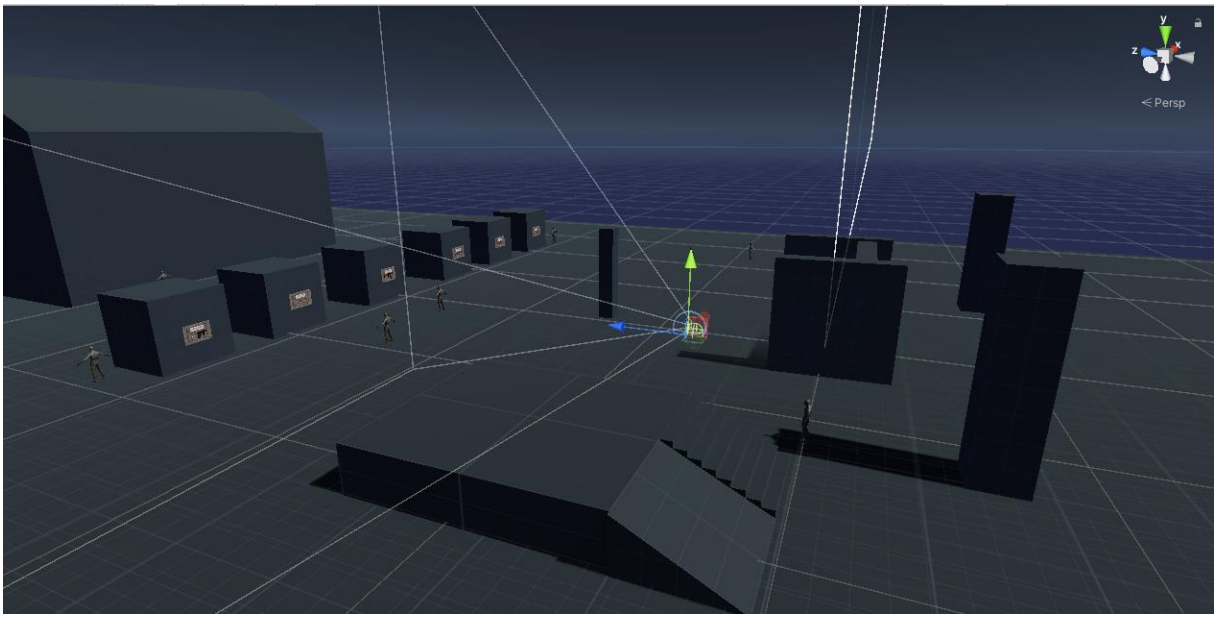
Slojeve koji čine igru možemo podijeliti na sljedeće: jezgra igre, pravila kraja igre, okolnosti igre, grafika i zvukovi, scenariji (razine) igre, priča. Na sljedećoj slici možemo vidjeti kako *onion* dizajn izgleda vizualizirano:




Slika 16: *Onion* dizajn (Izvor: *Lost Garden*, 2005.)

### 3.5.2. Prototipna *sandbox* mapa

Prema prilogu 1. eng. *Prototyping Sandbox map* je korisna mapa kojoj je primarni cilj brži razvoj jezgre igre. Ovo je početna mapa u kojoj se pripremaju skripte i objekti igre koji će nakon razvoja samo biti prebačeni u pravu igru. Glavna svrha prototipne *sandbox* mape je brže testiranje svih skripti. Na ovoj mapi fokus je na funkcionalnosti, a ne na estetici pa time pri razvoju primjerice nećemo koristiti modele zombija kao neprijatelje, već samo kapsule koje će imati iste funkcionalnosti kao zombiji. Prototipna *sandbox* mapa ovog završnog rada izgleda ovako:



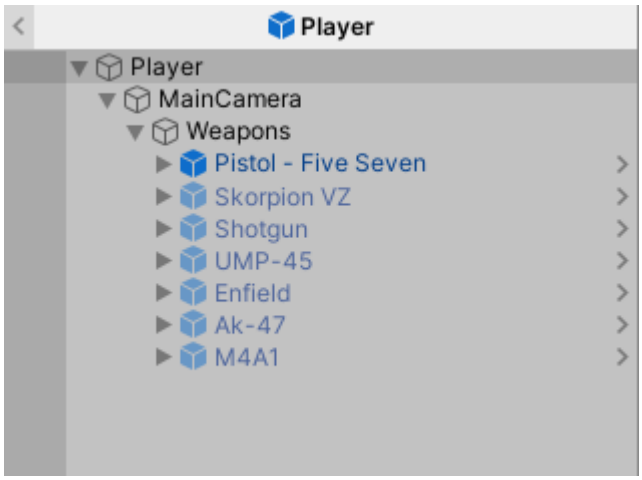

Slika 17: Prototipna *sandbox* mapa

Na ovoj mapi možemo vidjeti mnogo elemenata istog stila. Oni su svi dio *Unity Standard Asset* paketa koji dolazi sa mnogim objektima koji su veoma dobri za prototipiranje igre. Na prototipnoj mapi bitno je postaviti razne vrste ovih objekata kao što su naprimjer rampe, stepenice, zidovi itd. kako bismo mogli testirati sva moguća međudjelovanja objekata igre.

### 3.5.3. Igrač

Igrač (eng. *Player*) će biti objekt igre u kontroli osobe koja igra igru. U ovome dijelu objasnit ćemo što sve ovaj objekt igre nosi te sve skripte koje su na njega povezane.

Hijerarhijski prozor objekta igre sastoji se od više komponenata. Na sljedećoj slici možemo vidjeti njegovu listu:

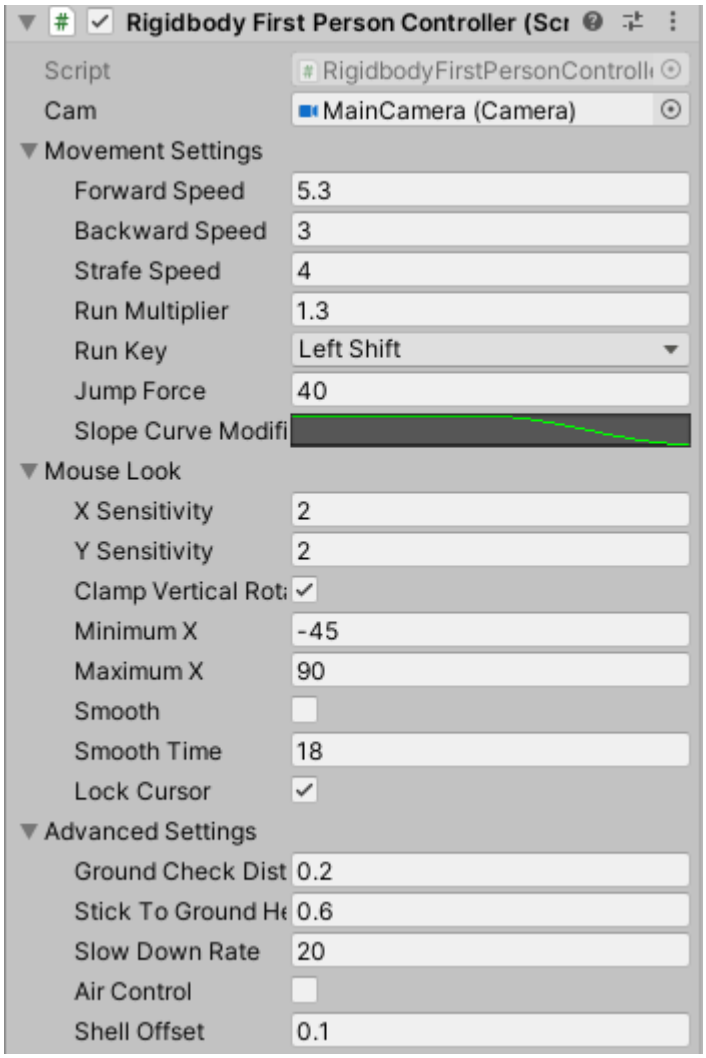



Slika 18: Hijerarhijski prozor *Player* (Izvor: Vlastita slika zaslona, 2021)

Započnimo od najvišeg pretka ovog objekta igre. Sam *Player* jest „*Empty*“ ili prazan objekt igre koji, kao što mu i samo ime govori je objekt koji nema izgled već ima samo lokaciju, tj. *Transform* komponentu. Ovdje je bitno napomenuti da u Unityju objekt ne može postojati bez svoje *Transform* komponente; točnije ona se s nijednog objekta ne može ukloniti. Prazni objekt igre obično se koristi kao spremnik za grupiranje elemenata, no u našem slučaju on je ovdje kako kamera ne bi morala nositi sve komponente igrača. [26]

Na *Player* objektu igre, uz komponentu *Transform* i skripte nalaze se još dvije komponente, a to su *Rigidbody* i *Capsule Collider*. *Rigidbody* je komponenta koja služi za simuliranje fizike na objektu igre. Ovo uključuje gravitaciju koja će vući tijelo prema dolje, ali i fiziku kolizija za koje je još potrebna *Collider* komponenta kako bi *Rigidbody* bio u potpunosti funkcionalan. [27] Komponenta *Capsule Collider* u kombinaciji sa *Rigidbody* komponentom služi za omogućavanje fizičkih kolizija ili sudara u igri. Ovo točnije znači da dva *Collidera* dvaju objekata neće moći odjednom biti na istom mjestu, već će se odbijati ukoliko dođe do sudara. Riječ „*capsule“* samo označava koji će fizički oblik ovaj objekt nositi pri kolizijama. U našem slučaju to je kapsula.[28]

Na Player objektu još imamo i skriptu „*Rigidbody First Person Controller (Script)*“. Ova skripta zaslužna je za: kretanje igrača u svim smjerovima, trčanje te skakanje. Skripta je veoma lagana  za korištenje te je za funkcionalnost potrebno samo povezati kameru na polje „*Cam*“ i podesiti željene parametre svojstava.




Slika 19: Inspektorski prozor skripte RFPC (Izvor: Vlastita slika zaslona, 2021)

Kako bismo postavili kameru, potrebno je na polje „*Cam*“ samo dovući kameru iz hijerarhijskog prozora ili jednom pritisnuti na ovo polje i odabrati glavnu kameru. Ostala svojstva odabiru se proizvoljno od kojih su najbitnija svojstva pod grupom „*Movement Settings*“. Još jedno bitno svojstvo nalazi se na dnu, a to je „*Shell Offset*“. Ovo svojstvo postavljeno je na vrijednost 0.1 jer se pri testiranju igrač često zaglavio u zidu. Ovim svojstvom ograničavamo igrača da drugom *Collideru* može doći na najbližu udaljenost od 0.1 jedinice.


Slika 20: Prikaz igre - igrač (Izvor: Vlastita slika zaslona, 2021)

Sam igrač nema model, već se objekt igrača sastoji samo od oružja koje je namješteno da bude unutar pogleda kamere.

### 3.5.3.1. Skripta PlayerHealth.cs

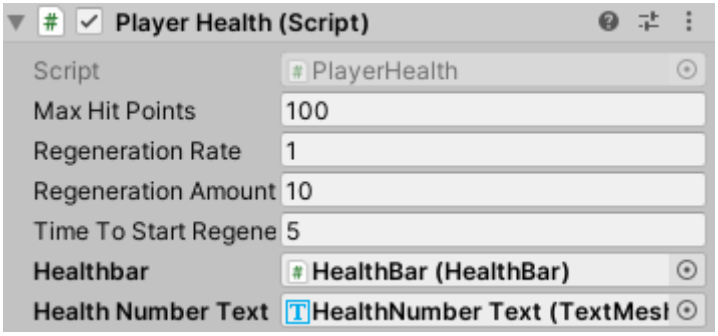
```
public class PlayerHealth : MonoBehaviour
{
    [SerializeField] float maxHitPoints = 100f;
    [SerializeField] float regenerationRate = 1f;
    [SerializeField] float regenerationAmount = 1f;
    [SerializeField] float timeToStartRegeneration = 5f;

    [SerializeField] HealthBar healthbar;
    [SerializeField] TextMeshProUGUI healthNumberText;

    private float hitPoints = 100f;
    private float timestamp = 0.0f;

    private void Start()
    {
        InvokeRepeating("Regeneration", 0, regenerationRate);
    }
```

```
    public void TakeDamage(float damage)
    {
        hitPoints -= damage;
        timestamp = Time.time;
        UpdateHealthBarGUI();

        if (hitPoints <= 0)
        {
            GetComponent<DeathHandler>().HandleDeath();
        }
    }

    void Regeneration()
    {
        if (hitPoints < maxHitPoints && Time.time > (timestamp +
timeToStartRegeneration))
        {
            hitPoints += regenerationAmount;
            UpdateHealthBarGUI();
        }
    }

    private void UpdateHealthBarGUI()
    {
        healthbar.SetHealth(hitPoints);
        healthNumberText.text = hitPoints.ToString();
    }
}
```

Pošto se radi o prvoj skripti u ovom radu, pomnije ćemo pojasniti karakteristike Unityja. Prvo što se može zamijetiti jest da naša klasa nasljeđuje od klase *MonoBehaviour*. Prema zadanim postavkama svaka nova skripta automatski nasljeđuje upravo od ove klase. *Monobehaviour* je jedna od najbitnijih klasa u Unityju jer nam ona dopušta da pristupamo funkcijama kao što su „*Start*()“, „*Update*()““ „*OnEnable*“ itd. Ove funkcije koriste se u gotovo svim skriptama i bit će objašnjene kada će se koristiti u skriptama ovoga rada.[30]

PlayerHealth.cs klasa je zadužena za životne bodove (eng. *health point*) igrača. Sadrži sve od primanja štete (eng. *damage*), regeneracije pa do ažuriranja igračevog sučelja na zaslonu.

Skriptu započinjemo deklaracijom varijabli. Umetanjem oznake „[*SerializeField*]“ prije varijabli omogućavamo si lakše podešavanje svih varijabli pri testiranju. Ovo je veoma bitno jer svaki puta kada želimo promijeniti vrijednost određenih varijabli, to možemo učiniti preko inspektorskog prozora.




Slika 21: Inspektorski prozor *Player Health* (Izvor: Vlastita slika zaslona, 2021)

Još jedan od razloga za dodavanje ove oznake varijablama jest direktno referenciranje, gdje ukoliko nam treba referenca na određenu sliku ili objekt igre, potrebno ga je samo dovući u ovo polje ili ga označiti iz izbornika.

U funkciji *Start* započinje se funkcija *InvokeRepeating(string name, float N, float M)* iz klase „*MonoBehaviour“*. Ova metoda zaslužna je za pozivanje metode određenog imena nakon N sekundi i zatim svakih M sekundi. U našem slučaju ona se poziva odmah u metodi *Start*() što znači da započinje pri pokretanju igre. Ova metoda pozivat će funkciju *Regeneration()* svakih *regenerationRate* sekundi te će time vraćati životne bodove igraču ukoliko on nema maksimalnu količinu ovih bodova. [31]

Uz ove metode možemo još izdvojiti metodu *TakeDamage(float damage)* koja u ovoj funkciji nema poziva, no ona će biti korištena iz drugih skripti. U njoj se smanjuju životni bodovi igrača i zapisuje se vrijeme kada je igrač ranjen kako bi funkcija *Regeneration()* znala kada započeti regeneraciju. Ukoliko nakon oduzimanja životnih bodova oni padnu ispod 0, igrač umire i poziva se funkcija iz skripte „*DeathHandler.cs*“.


Slika 22: Prikaz igre – umanjeni životni bodovi (Izvor: Vlastita slika zaslona, 2021)

### 3.5.3.2. Skripta PlayerMoney.cs

```
public class PlayerMoney : MonoBehaviour
{
    [SerializeField] int moneyAmount = 0;
    [SerializeField] TextMeshProUGUI moneyText;
    private void Start()
    {
        MoneyGUIUpdate();
    }

    public void AddMoney(int amount)
    {
        moneyAmount += amount;
        MoneyGUIUpdate();
    }

    public void SubtractMoney(int amount)
    {
        moneyAmount -= amount;
        MoneyGUIUpdate();
    }
```

```
    public int MoneyAmount()
    {
        return moneyAmount;
    }

    private void MoneyGUIUpdate()
    {
        moneyText.text = "$" + moneyAmount;
    }
}
```

Skripta *PlayerMoney.cs* zadužena je za praćenje količine novca igrača. Ovdje se nalaze funkcije za dodavanje novca, smanjivanje novca i ažuriranje igračevog sučelja.

Sve od ovih funkcija same su po sebi razumljive. Ukratko, *AddMoney(int amount)* funkcija dodaje *amount* novca i ažurira sučelje. *SubtractMoney(int amount)* oduzima *amount* novca i ažurira sučelje. *MoneyAmount()* vraća količinu novca kako ne bi bilo izravnog pristupanja varijabli, a *MoneyGUIUpdate()* ažurira tekst na igračevom sučelju, tj. platnu (eng. *Canvas*).


Slika 23: Prikaz igre – igrač sa $1000 (Izvor: Vlastita slika zaslona, 2021)

#### 3.5.3.3. Skripta Ammo.cs

```
public class Ammo : MonoBehaviour
{
    [SerializeField] int ammoAmount = 90;
    [SerializeField] TextMeshProUGUI ammoText;

    private int magazineSize;
    private int magazineBullets;

    public int GetCurrentAmmo()
    {
        return ammoAmount;
    }

    public int GetCurrentMagazineBullets()
    {
        return magazineBullets;
    }




    public int GetMagazineSize()
    {
        return magazineSize;
    }

    public void SetAmmoAmount(int boughtAmmo, int boughtMagazineSize)
    {
        ammoAmount = boughtAmmo;
        magazineSize = boughtMagazineSize;
        magazineBullets = boughtMagazineSize;

        AmmoGUIUpdate();
    }

    public void AmmoGUIUpdate()
    {
        ammoText.text = magazineBullets.ToString() + "/" +
ammoAmount.ToString();
        if(magazineBullets < magazineSize * 0.25)
```

```
        {
            ammoText.color = new Color(200, 0, 0, 255);
        }
        else
        {
            ammoText.color = new Color(255, 255, 255, 255);
        }
    }

    public void ReduceCurrentAmmo(string weaponName)
    {
        if(weaponName != "Pistol - Five Seven")
        {
            magazineBullets--;
            AmmoGUIUpdate();
        }
        else
        {
            ammoText.text = "\u221e/\u221e";
        }
    }

    public void Reload()
    {
        if(ammoAmount - magazineSize >= 0)
        {
            ammoAmount -= magazineSize;
            magazineBullets = magazineSize;
            AmmoGUIUpdate();
        }
        else
        {
            magazineBullets = ammoAmount;
            ammoAmount = 0;
            AmmoGUIUpdate();
        }
    }
}
```

Skripta *Ammo.cs* zaslužna je za postavljanje streljiva (eng. *Ammo*) pri kupnji, smanjivanje streljiva pri pucanju, ponovno punjenje puške (eng. *Reload*) te ažuriranje korisničkog sučelja. Bitno je napomenuti da je igra konceptualizirana na način da trenutno oružje, čije su skripte na samim objektima oružja, poziva ove funkcije te da ova skripta prikazuje streljivo trenutno aktivnog oružja. Ova skripta ima i jednu iznimku, a to je da pištolj ima beskonačno metaka te je on kroz igru sekundarno oružje koje je ovdje ukoliko igraču ponestane metaka. Igrač barata samo dvama tipovima oružja u svakom trenutku igre. Primarno oružje koje se mora kupiti i ima streljivo te sekundarno - prije navedeni pištolj.

Prve tri funkcije služe samo za pristupanje privatnim varijablama iz drugih skripti čime se držimo pravila objektno-orijentiranog programiranja.

Funkcija *SetAmmoAmount(int boughtAmmo, int boughMagazineSize)* bit će pozivana iz *WeaponShop.cs* skripte i ona služi za postavljanje streljiva pri kupnji oružja.

*ReduceCurrentAmmo(string weaponName)* funkcija koristi se za smanjivanje streljiva pri pucanju. Kao argument proslijeđeno je ime oružja kako pri pucanju iz sekundarnog oružja pištolja ne bismo gubili streljivo.

*Reload()* funkcija služi za ponovno punjenje puške streljivom. Ovdje je uzet malo drugačiji i realniji pristup od klasičnih igara pucanja. Ukoliko igrač puni pušku streljivom prije nego što je potrošio sve metke, ti metci su izbrisani, tj. ne vraćaju se u kumulativnu količinu dodatnog streljiva koje to oružje ima.

Funkcija *AmmoGUIUpdate()* zapisuje koliko metaka oružje ima u formatu „*količina metaka u šaržeru / količina preostalih metaka*“. U slučaju da u šaržeru (eng. *Magazine)* ima manje od 25% metaka, boja se mijenja u crveno.


Slika 24: Prikaz igre – mali broj metaka (Izvor: Vlastita slika zaslona, 2021)

#### 3.5.3.4. Skripta DeathHandler.cs

```
public class DeathHandler : MonoBehaviour
{
    [SerializeField] Canvas gameOverCanvas;

    private void Start()
    {
        gameOverCanvas.enabled = false;
    }

    public void HandleDeath()
    {
        gameOverCanvas.enabled = true;
        Time.timeScale = 0;
        FindObjectOfType<WeaponSwitcher>().enabled = false;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
```

Ova kratka skripta zaslužna je za zaustavljanje igre, prikaza platna kraja igre te zaključavanje kursora. Ona se poziva iz skripte *PlayerHealth.cs* ukoliko igrač ima manje od 0 životnih bodova.

Linijom „*Time.timeScale = 0;*“ Unityu zadajemo da igra teče brzinom 0 sekundi (u igri) po sekundi (u realnom vremenu), točnije da je igra zaustavljena. Pošto se radi o igri pucanja i nema kursora, bitno je u ovome trenutku ponovno uključiti kursor kako bi igrač mogao ponovno započeti igru.


Slika 25: Prikaz igre – zaslon nakon smrti (Izvor: Vlastita slika zaslona, 2021)

### 3.5.3.5. Skripta WeaponSwitcher.cs

```
public class WeaponSwitcher : MonoBehaviour
{
    [SerializeField] int currentWeapon = 0;
    [SerializeField] int boughtWeapon = 0;
    [SerializeField] TextMeshProUGUI ammoText;
    [SerializeField] Ammo ammo;

    void Start()
    {
        SetWeaponActive();
    }

    void Update()
    {
        int previousWeapon = currentWeapon;

        ProcessKeyInput();
        ProcessScrollWheel();
        if(previousWeapon != currentWeapon)
        {
            SetWeaponActive();
        }
    }

    private void SetWeaponActive()
    {
        int weaponIndex = 0;

        foreach (Transform weapon in transform)
        {
            if (weaponIndex == currentWeapon)
            {
                weapon.gameObject.SetActive(true);
                if (weapon.gameObject.name != "Pistol - Five Seven")
                {
                    ammo.AmmoGUIUpdate();
                }
                else
                {
                    ammoText.text = "\u221e/\u221e";
                    ammoText.color = new Color(255, 255, 255, 255);
                }
            }
            else
            {
                weapon.gameObject.SetActive(false);
            }
            weaponIndex++;
        }
    }

    private void ProcessScrollWheel()
    {
        if (Input.GetAxis("Mouse ScrollWheel") != 0)
        {
            if (currentWeapon == 0)
            {
                currentWeapon = boughtWeapon;
            }
```

```
            else
            {
                currentWeapon = 0;
            }
        }
    }

    private void ProcessKeyInput()
    {
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            currentWeapon = boughtWeapon;
        }
        if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            currentWeapon = 0;
        }
    }

    public void SetBoughtWeapon(int boughtWeaponIndex)
    {
        boughtWeapon = boughtWeaponIndex;
        currentWeapon = boughtWeapon;

        SetWeaponActive();
    }
}
```

Ova skripta nalazi se na praznom objektu *Weapons* koji je dijete *MainCamerae* sa slike 18. Prazni objekt *Weapons* koji nosi ovu skriptu je spremnik svih oružja koje bi igrač mogao posjedovati, no samo jedno oružje aktivirano je u jednom trenutku.

Kao što i prijevod imena skripte navodi, ova skripta zadužena je za mijenjanje oružja kojim igrač barata, bilo to pri kupnji ili u borbi sa zombijima.

U funkciji *Start()*, tj. na početku igre, oružje se postavlja na oružje s indeksom 0. Na ovom indeksu nalazi se početni pištolj.

Nakon nje nalazi se funkcija *Update()* s kojom se u ovome radu po prvi put susrećemo. Ovo je jedna od najvažnijih funkcija i ona se poziva jednom svaki put kada se generira slika na zaslonu (eng. *Frame)*. Ovo je veoma česta funkcija koja se događa otprilike 60 do 300 puta u sekundi i upravo zbog toga ona u pravilu ne bi smjela sadržavati blokove naredbi velikih složenosti jer ukoliko se sve naredbe ne izvrše do sljedećeg *framea*, taj će *frame* biti odgođen što će rezultirati podrhtavanjem. [33]

U našoj funkciji *Update()* ispitujemo je li korisnik pritisnuo tipke ili zarotirao kotačić na mišu. Ukoliko je zarotirao kotačić ili pritisnuo oružje koje nije već trenutno aktivno, tada aktivira oružje koje je pritisnuo.

Funkcija *SetWeaponActive()* pomoću *foreach* petlje postavlja oružje koje smo pritisnuli na aktivno, dok sva ostala oružja postavlja na neaktivno.

Funkcija *ProcessScrollWheel()* pri rotaciji mijenja oružje između nultog (pištolja) i trenutne puške koju igrač posjeduje, dok na sličan način *ProcessKeyInput()* pritiskom na broj „1“ na tipkovnici odabire primarno (kupljeno) oružje ili pritiskom na broj „2“ odabire sekundarno (pištolj).

Zadnja funkcija *SetBoughtWeapon(int boughtWeaponIndex)* zadužena je za postavljanje i aktiviranje novokupljenog oružja i ona se poziva iz skripte *WeaponShop.cs*.

### 3.5.3.6. Skripta Weapon.cs

```
public class Weapon : MonoBehaviour
{
    [SerializeField] Camera FPCamera;
    [SerializeField] float range = 100f;
    [SerializeField] float damage = 30f;
    [SerializeField] float headshotDamageMultiplier = 1.5f;
    [SerializeField] ParticleSystem muzzleFlash;
    [SerializeField] GameObject hitEffect;
    [SerializeField] Ammo ammoSlot;
    [SerializeField] PlayerMoney playerMoney;
    [SerializeField] float shotFrequency = 0.5f;
    [SerializeField] int moneyPerHit = 10;
    [SerializeField] int headshotMoneyMultiplier = 2;
    [SerializeField] float reloadTime = 2;

    bool isReloading = false;
    bool canShoot = true;
    float headshotDamage;
    AudioSource weaponAudio;
    AudioSource reloadAudio;

    private void OnEnable()
    {
        canShoot = true;
        headshotDamage = damage * headshotDamageMultiplier;
        var sources = GetComponents<AudioSource>();
        weaponAudio = sources[0];
        if (sources.GetUpperBound(0) == 1)
        {
            reloadAudio = sources[1];
        }
    }

    void Update()
    {
        if (Input.GetMouseButton(0) && canShoot == true)
        {
            if(ammoSlot != null)
            {
                if(ammoSlot.GetCurrentMagazineBullets() != 0 || gameObject.name ==
"Pistol - Five Seven")
```

```
                {
                    StartCoroutine(Shoot());
                }
                else
                {
                    StartCoroutine(Reload());
                }
            }
        }
        if (Input.GetKeyDown(KeyCode.R))
        {
            if(ammoSlot.GetCurrentAmmo() > 0 && ammoSlot.GetCurrentMagazineBullets() <
ammoSlot.GetMagazineSize() && !isReloading)
            {
                StartCoroutine(Reload());
            }
        }
    }

    IEnumerator Reload()
    {
        if (gameObject.name != "Pistol - Five Seven")
        {
            isReloading = true;
            canShoot = false;
            reloadAudio.Play();
            yield return new WaitForSeconds(reloadTime);
            ammoSlot.Reload();
            canShoot = true;
            isReloading = false;
        }
    }

    IEnumerator Shoot()
    {

        canShoot = false;
        if((ammoSlot.GetCurrentMagazineBullets() > 0 || gameObject.name == "Pistol -
Five Seven") && !isReloading )
        {
            ProcessRaycast();

            muzzleFlash.Play();
            weaponAudio.Play();
            ammoSlot.ReduceCurrentAmmo(gameObject.name);
        }
        yield return new WaitForSeconds(shotFrequency);
        canShoot = true;
    }

    private void ProcessRaycast()
    {
        RaycastHit hit;
        if (Physics.Raycast(FPCamera.transform.position, FPCamera.transform.forward,
out hit, range))
        {
            CreateHitImpact(hit);
            EnemyHealth target = hit.transform.GetComponent<EnemyHealth>();
            if (target == null && hit.transform.name != "Z_Head") return;

            if(hit.transform.name == "Z_Head")
```

```
            {
                target = hit.transform.GetComponentInParent<EnemyHealth>();
                target.TakeDamage(headshotDamage);
                playerMoney.AddMoney(moneyPerHit * headshotMoneyMultiplier);
            }
            else
            {
                target.TakeDamage(damage);
                playerMoney.AddMoney(moneyPerHit);
            }
        }
        else
        {
            return;
        }
    }

    private void CreateHitImpact(RaycastHit hit)
    {
        GameObject impact = Instantiate(hitEffect, hit.point,
Quaternion.LookRotation(hit.normal));
        Destroy(impact, .1f);
    }
}
```

Weapon.cs skripta je skripta koja se nalazi na svakom oružju sa slike 18. Ona je primarno zadužena za ponovno punjenje puške i pucanje, ali i za sve dodatne efekte koji se događaju pri tome.

Započnimo funkcijom *OnEnable().* Ova funkcija također je dio *Monobehaviour* klase i poziva se kada se određeni objekt igre aktivira. Ovo nam je korisno zato što oružja aktiviramo i deaktiviramo pomoću skripte *WeaponSwitcher.cs* . U našem slučaju ova skripta zadužena je za postavljanje početnih varijabli i pribavljanje audio izvora. U ovome dijelu potrebna nam je i uvjetna naredba zato što pištolj oružje nema dva izvora, dok sva ostala oružja imaju jedan audio izvor za pucanje, a drugi za ponovno punjenje puške. [34]

U *Update()* funkciji, kao i kod *WeaponSwitcher.cs* provjerava se igračev *input*. U slučaju da igrač pritisne lijevi klik i u mogućnosti je pucati, oružje puca ili se ponovno puni ovisno o količini metaka u šaržeru.

Unutar *Update()* funkcije nalaze se dvije funkcije pod nazivom *StartCoroutine()*, koje su zaslužne za pozivanje funkcije koja može biti pauzirana. Pauziranje se izvodi pomoću ključne riječi „*yield*“ kojom se izvršavanje pauzira do sljedećeg *framea* nakon čega pozvana korutina (eng. *Coroutine*) nastavlja svoje izvršavanje od linije koda nakon „*yield*“ ključne riječi. Ovo se može još dodatno vremenski produžiti na način da pri *„yield return*“ vraćamo novi objekt tipa *„new WaitForSeconds(float)*“. Ukoliko učinimo upravo to, izvršavanje će se pauzirati za određen broj sekundi. Funkcija *StartCoroutine()* korisna nam je upravo zato što možemo igraču

zabraniti ulazak u funkciju pomoću varijable tipa *bool* te mu ponovno dopustiti ulazak u funkciju nakon N sekundi. Korutine koje se pozivaju moraju biti tipa „*IEnumerator*“. [35]

Funkcija *Reload()* ne radi za pištolje pošto on ima beskonačan broj metaka. Unutar nje postavljaju se vrijednosti varijabli koje onemogućavaju ponovno pozivanje *Reload()* i *Shoot()* funkcija, pokreće se audio punjenja oružja te tek nakon *reloadTime* sekundi puška može ponovno pucati.

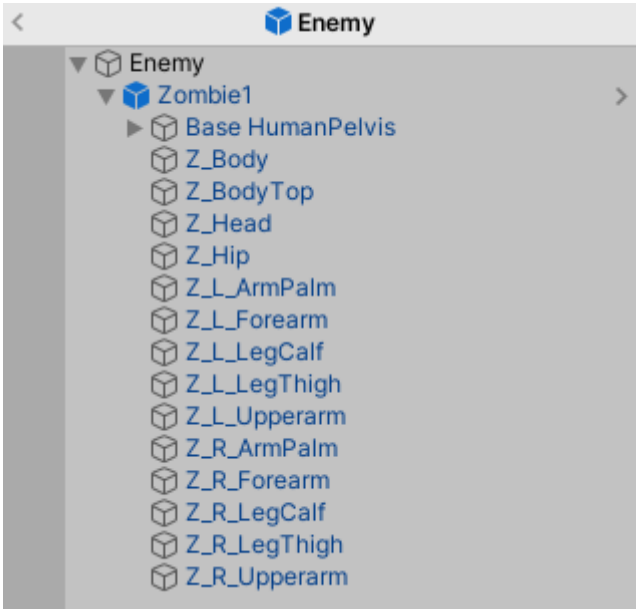
Funkcija *Shoot()* na isti način postavlja vrijednost varijable kako se ne bi moglo uzastopno pucati te zatim provjerava ima li oružje još metaka u šaržeru. Ukoliko ima još metaka i nije u stanju punjenja puške, puška puca i smanjuje broj metaka. Nakon *shotFrequency* sekundi puška može ponovno pucati. Pucanje također poziva funkciju *ProcessRaycast()* koja je zaslužna za ispaljivanje metka.

Funkcija *ProcessRaycast()* bavi se slanjem zrake u određenom smjeru na određenu udaljenost. U našem slučaju šalje se iz kamere u smjeru prema kojem je kamera okrenuta na udaljenost koju to oružje nosi. Na mjestu gdje zraka pogodi objekt igre generira se efekt hitca (mala eksplozija), uz to provjerava se je li pogođena meta zombi te ukoliko jest provjerava se je li pogođena glava zombija ili ostatak pa se ovisno o odgovoru primjenjuje odgovarajući blok koda. [36]


Slika 26: Prikaz igre – pucanje oružjem (Izvor: Vlastita slika zaslona, 2021)

### 3.5.4. Protivnik

Protivnik (eng. *Enemy*) je objekt igre koji će napadati igrača. Ovaj dio opisivat će komponente zombija i skripte koje su zadužene za praćenje i napadanje igrača. Na sljedećoj slici možemo vidjeti hijerarhijski prozor zombija.

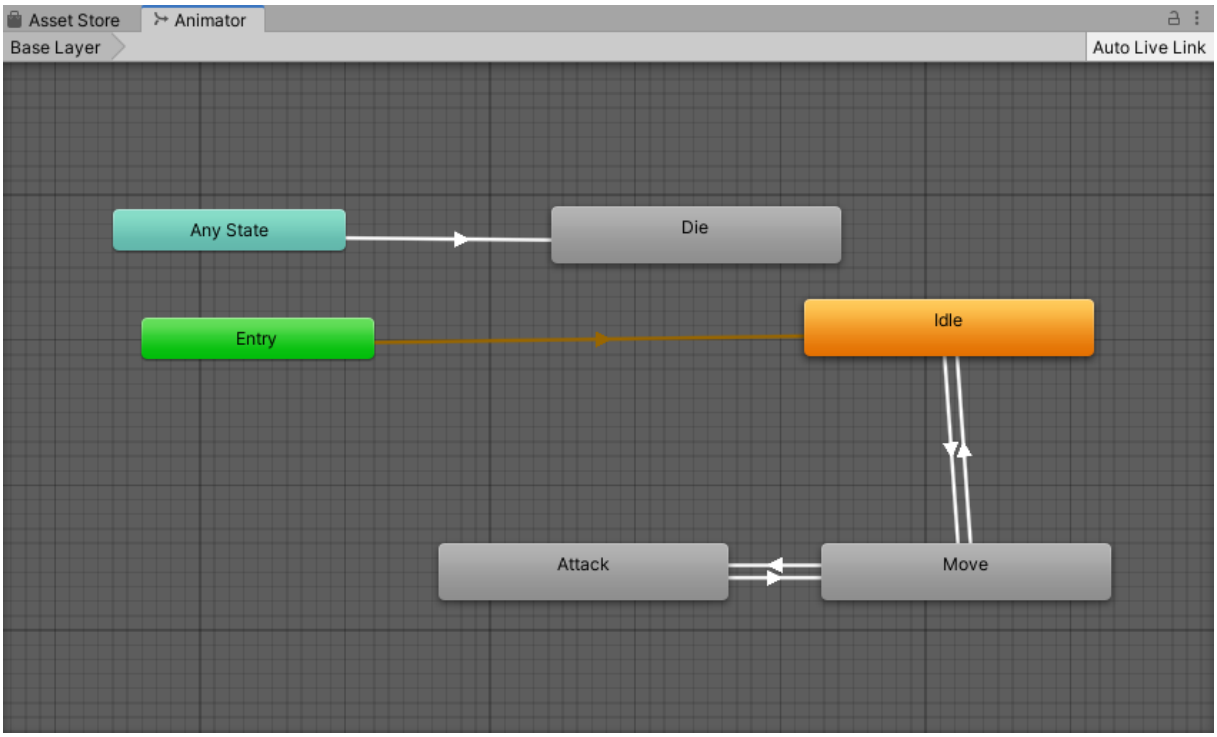



Slika 27: Hijerarhijski prozor Enemy (Izvor: Vlastita slika zaslona, 2021)

Ovaj objekt igre uvezen je iz besplatnog Unity paketa koji se nalazi u skladištu imovine. Dolazi s modelom, *colliderom* i pripadajućim animacijama. Skoro sve skripte i potrebne komponente nalaze se na najvišem pretku ovog objekta, a to je prazan objekt „*Enemy*“. Jedina bitna komponenta koja se ne nalazi na ovom objektu jest dodatan *Capsule Collider* koji se nalazi na glavi zombija. Ta nam je komponenta potrebna kako bismo razlikovali hitac u glavu od hitca u tijelo.

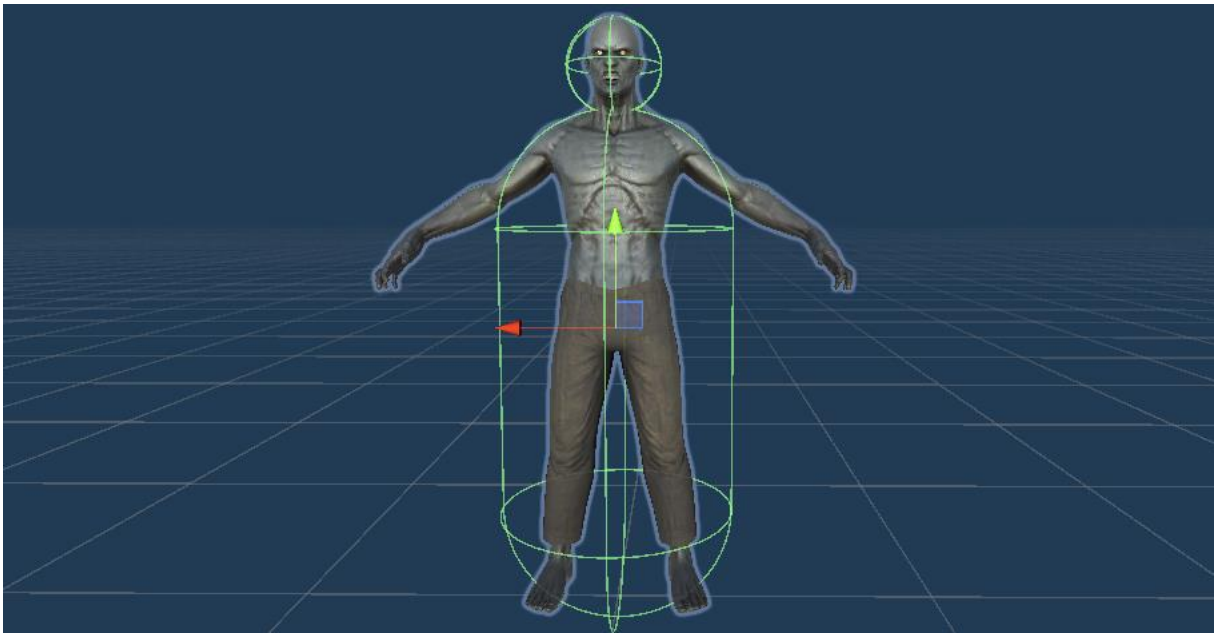
*NavMeshAgent* komponenta koja je na objektu „*Enemy*“, zadužena je za navigiranje zombija po *NavMeshu* koji je bio objašnjen u naslovu 3.3. Na njoj se nalaze mnoga svojstva koja su slična onom sa slike 19., a to su svojstva zadužena za kretanje kao što su brzina, udaljenost zaustavljanja itd. [37]

Komponenta *Animator* kontrolira animacije zombija koje ćemo pokretati iz skripte *EnemyAI.cs*. Zombi će koristiti samo tri animacije, a to su hodanje, napadanje i smrt. Kako bi zombi mogao prelaziti iz jedne animacije u drugu moramo grafički povezati dozvoljene prijelaze čime se kreira stroj stanja koji će zabranjivati da dođe do nedozvoljenih stanja.




Slika 28: Animatorski Prozor (Izvor: Vlastita slika zaslona, 2021)

Uz ove komponente nalaze se još i *Capsule Collider* zadužen za sudaranje te standardni *Mesh Renderer* zadužen za prikaz zombija na zaslonu koji su već prije bili objašnjeni.



Slika 29: Model protivnika (Izvor: Vlastita slika zaslona, 2021)

#### 3.5.4.1. Skripta EnemyHealth.cs

```
public class EnemyHealth : MonoBehaviour
{
    [SerializeField] public float hitPoints = 100f;
    [SerializeField] float timeTillDestroyObject = 10f;
    NavMeshAgent navMeshAgent;
    EnemyAI enemyAI;
    bool isDead = false;

    public void TakeDamage(float damage)
    {
        hitPoints -= damage;
        if (hitPoints <= 0)
        {
            Die();
        }
    }

    private void Die()
    {
        if (isDead) return;
        isDead = true;
        GetComponent<Animator>().SetTrigger("die");
        DisableComponents();
        Destroy(gameObject, timeTillDestroyObject);
    }

    private void DisableComponents()
    {
        GetComponent<NavMeshAgent>().enabled = false;
        GetComponent<EnemyAI>().enabled = false;
        GetComponent<CapsuleCollider>().enabled = false;
    }
}
```

Ova skripta služi za praćenje životnih bodova zombija te njegovu smrt i brisanje objekta sa scene.

Prva funkcija *TakeDamage(float damage)* korištena je za umanjivanje životnih bodova zombija za svotu *damage*. U slučaju da je rezultat nakon oduzimanja manji ili jednak 0 zombi umire.

Funkcija umiranja naziva se engleskom riječju *Die()* i pristupa komponenti *Animator* kako bi postavila okidač za smrt. U tom trenutku iz bilo kojeg stanja zombi prelazi u animaciju umiranja i pada na pod. Objekt se uništava funkcijom *Destroy()* nakon *timeTillDestroyObject* sekundi vremena, no prije nego se objekt uništi on je i dalje u sceni zbog čega je potrebno isključiti sve ove komponente kako se zombi u smrti ne bi nastavio micati ili zauzimati prostor. [38]

### 3.5.4.2. Skripta EnemyAI.cs

```
public class EnemyAI : MonoBehaviour
{
    [SerializeField] public Transform target;
    [SerializeField] float turnSpeed = 5f;
    NavMeshAgent navMeshAgent;
    float distanceToTarget = Mathf.Infinity;

    void Start()
    {
        navMeshAgent = GetComponent<NavMeshAgent>();
    }

    void Update()
    {
        distanceToTarget = Vector3.Distance(target.position, transform.position);
        EngageTarget();
    }

    private void EngageTarget()
    {
        FaceTarget();
        if (distanceToTarget >= navMeshAgent.stoppingDistance)
        {
            ChaseTarget();
        }
        else
        {
            AttackTarget();
        }
    }

    private void ChaseTarget()
    {
        GetComponent<Animator>().SetBool("attack", false);
        GetComponent<Animator>().SetTrigger("move");
        navMeshAgent.SetDestination(target.position);
    }

    private void AttackTarget()
    {
        GetComponent<Animator>().SetBool("attack", true);
    }

    private void FaceTarget()
    {
        Vector3 direction = (target.position - transform.position).normalized;
        Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0,
direction.z));
        transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation,
Time.deltaTime * turnSpeed);
    }
}
```

Skripta *EnemyAI.cs* nosi ulogu traženja i napadanja mete. Za ovo koristi *NavMeshAgent* komponentu i prilikom promjena stanja mijenja i animacije.

U funkciji *Update()* pri generiranju svakog *framea* računa se udaljenost do igrača koja se koristi u funkciji *EngageTarget()*. U tom trenutku ova funkcija ima pristup udaljenosti igrača i računa je li ona veća od udaljenosti *stoppingDistance* koja je zadužena da zombi ne može doći do centra igrača već da stane u blizini. Ukoliko je udaljenost veća od udaljenosti zaustavljanja, zombi se primiče još bliže, dok u suprotnom zombi napada metu.

Funkcija *ChaseTarget()* zaustavlja animaciju napada (ukoliko je bila postavljena na *true*) te postavlja animaciju na onu za pokretanje. Pristupanjem komponenti *NavMeshAgent* određujemo destinaciju zombija prema kojoj mora hodati.

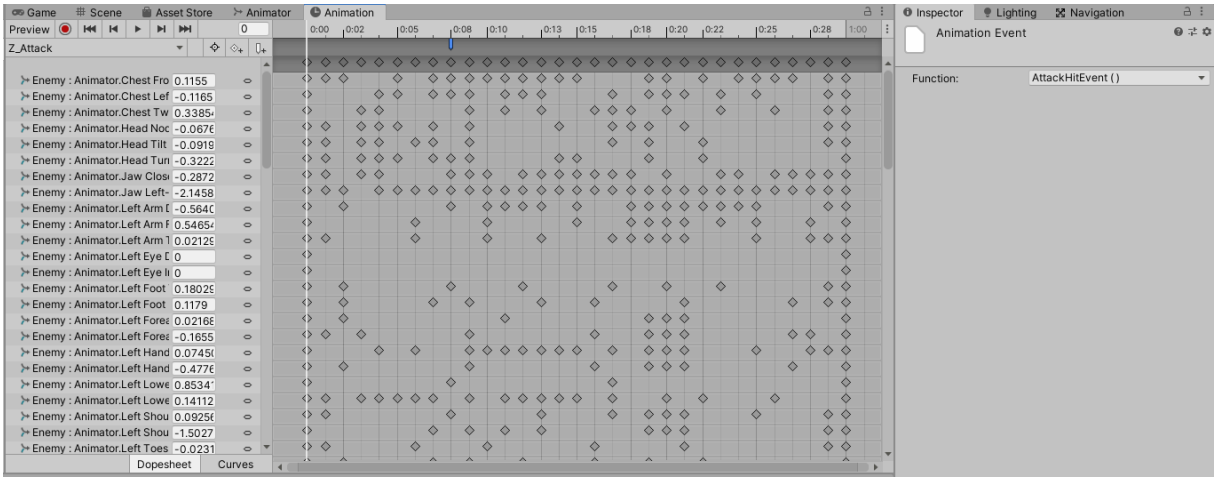
Funkcija *AttackTarget()* samo postavlja animaciju u napadanje dok se ostatak napada događa u odvojenoj skripti. Zadnja funkcija *FaceTarget()* koristi se kako zombi ne bi hodao unatrag kada se kreće prema igraču. [39] Pri rotaciji koristi se funkcija *Slerp*() koja je zadužena za sferičnu linearnu interpolaciju, laički rečeno dijeli rotaciju na više dijelova kako se zombi ne bi u jednom *frameu* okrenuo prema meti, već postepeno kroz više *frameova*.

#### 3.5.4.3. Skripta EnemyAttack.cs

```
public class EnemyAttack : MonoBehaviour
{
    PlayerHealth target;
    [SerializeField] public float damage = 20f;

    void Start()
    {
        target = FindObjectOfType<PlayerHealth>();
    }

    public void AttackHitEvent()
    {
        if (target == null) return;
        target.TakeDamage(damage);
    }
}
```

Ova kratka skripta zadužena je za primjenjivanje štete na životne bodove igrača. U funkciji start komponenta *PlayerHealth.cs* se pronalazi kako bi bila na raspolaganju za funkciju.

Funkcija *AttackHitEvent()* je funkcija koja se ne poziva iz naših skripti, već iz same animacije zombijevog napadanja. Ovo se radi tako da se na vrhu animacije postavi događaj (eng. *Event*) na kojem se označi koja se funkcija poziva. Ovo je korisno upotrijebiti na funkcijama koje se izvršavaju u sredini toka neke animacije. Primjerice, u našoj animaciji igrač ne dobiva štetu na početku animacije, već tek pri udarcu u trenutku kada je ruka najdalje od zombijevog tijela. [40]




Slika 30: Animacijski Prozor (Izvor: Vlastita slika zaslona, 2021)

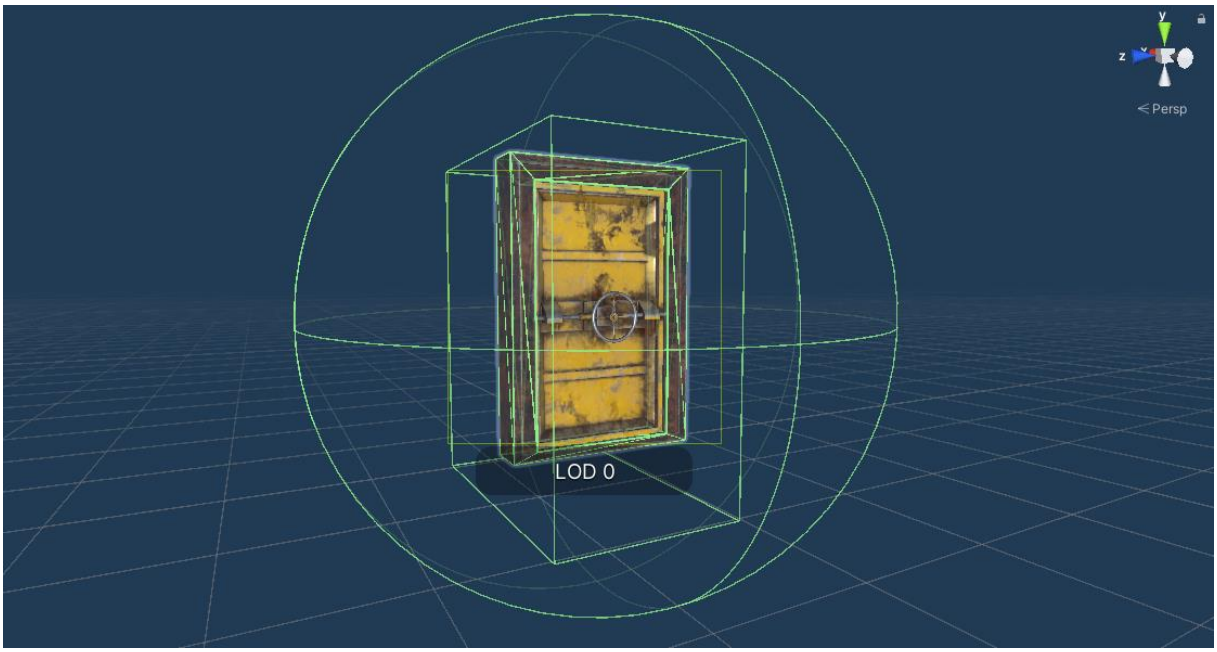
Plava oznaka pri vrhu slike 30. označava događaj, dok na desnoj strani slike možemo vidjeti funkciju koja se na tome događaju poziva.

### 3.5.5. Vrata

Vrata u igri služe za stvaranje osjećaja barem male progresije u igri. Model vratiju i animacija otvaranja došli su sa kupljenom mapom. Pošto želimo da u početku vrata budu zatvorena, komponenta animator je isključena te će se uključiti tek u trenutku kada igrač pritisne gumb s dovoljnim iznosom novca.

Vrata na sebi imaju i komponentu *Box Collider* kako osoba ne bi mogla proći kroz vrata. Ostatak komponenti nalazi se na drugom objektu igre nazvanom „*Shop Trigger*“.

Prazan objekt *Shop Trigger* nema izgled već samo nosi komponentu *Sphere Collider* i skriptu *DoorShop.cs* . *Sphere Collider* koji se nalazi na objektu nije jednak onom ranije opisanom *Collideru*. On se razlikuje jer mu je svojstvo *IsTrigger* postavljeno na *true*. Rezultat ove postavke je da *Collider* ne stvara sudar ukoliko drugi objekt dođe u njegovu blizinu, već šalje poruku kada drugi *RigidBody* uđe u njegov radijus. Ovo ukratko znači da ukoliko igrač uđe u radijus ovog *Collidera,* pokreće se funkcija *OnTriggerEnter()*, dok se pri izlasku pokreće suprotna funkcija *OnTriggerExit()*. [41]




Slika 31: Model vratiju (Izvor: Vlastita slika zaslona, 2021)

### 3.5.5.1. Skripta DoorShop.cs

```
public class DoorShop : MonoBehaviour
{
    [SerializeField] Canvas shopCanvas;
    [SerializeField] TextMeshProUGUI buyTextMesh;
    [SerializeField] int doorPrice;
    [SerializeField] PlayerMoney playerMoney;
    [SerializeField] Animator animator;

    bool canBuy = true;
    bool isColliding = false;

    void Update()
    {
        if (isColliding)
        {
            if (Input.GetKeyDown(KeyCode.F) && canBuy)
            {
                if (playerMoney.MoneyAmount() >= doorPrice)
                {
                    canBuy = false;
                    playerMoney.SubtractMoney(doorPrice);
                    animator.enabled = true;
                    shopCanvas.enabled = false;
                }
            }
        }
    }
```

```
    private void OnTriggerEnter(Collider other)
    {
        if (canBuy)
        {
            shopCanvas.enabled = true;
            buyTextMesh.text = "Press F to open the door \n" + "\t    ($" + doorPrice
+ ")";
            isColliding = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (canBuy)
        {
            isColliding = false;
            shopCanvas.enabled = false;
        }
    }
}
```

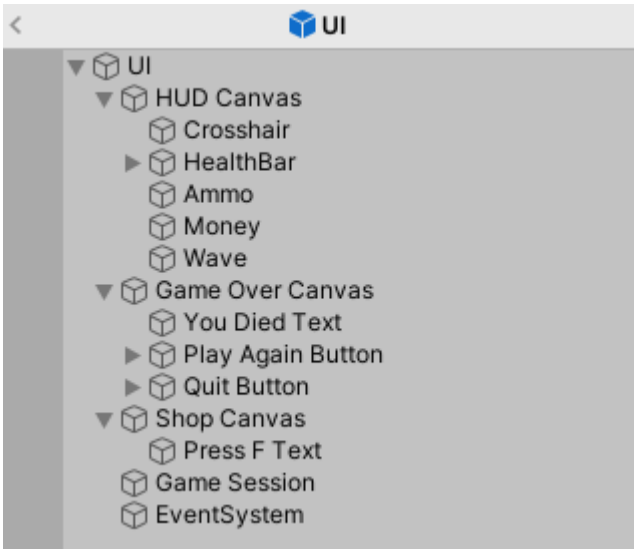
Funkcija *Update()* pri svakom *frameu* provjerava je li varijabla *isColliding* postavljena na true. Ukoliko jest, čeka da igrač stisne tipku „F“ kako bi kupio otvaranje vrata. Ovo se može desiti samo ukoliko igrač ima dovoljno novca pri čemu mu se taj novac oduzima i na vratima se aktivira komponenta „*Animator*“ koja uzrokuje pokretanje animacije otvaranja vratiju.

Funkcija *OnTriggerEnter()* poziva se kao što smo prije objasnili samo pri sudaru *RigidBodya* (igrača) s radijusom *Collidera*. U ovom slučaju na zaslonu igrača prikazuje se tekst na platnu koji sadrži informacije o tipki koju je potrebno pritisnuti i cijeni vratiju. Na izlasku iz radijusa *Collidera*, *OnTriggerExit()* ovo platno prestaje se prikazivati.


Slika 32: Prikaz igre – vrata (Izvor: Vlastita slika zaslona, 2021)

### 3.5.6. Grafičko korisničko sučelje

Grafičko korisničko sučelje u igri prisutno je kako bi davalo informacije igraču i u nekim slučajevima dopustilo mu da ima interakciju s igrom. Grafičko sučelje sastoji se od više platna (eng. *Canvas)* koja sadrže određene elemente koji su igraču potrebni u raznim trenucima u igri.




Slika 33: Hijerarhijski prozor UI (Izvor: Vlastita slika zaslona, 2021)

Objekt igre *„UI“* sastoji se od više platna koja se kroz igru u raznim trenucima pomoću prije navedenih skripti uključuju i isključuju. Platno „*HUD Canvas*“ zaduženo je za prikazivanje „*Heads-Up Displaya“* koji nije moguće ispravno prevesti na hrvatski jezik. Ovo platno u toku igre igraču daje informacije o njegovim najvažnijim resursima i stanjima. [42]

Platno „*GameOverCanvas*“ aktivira se pri smrti igrača i ostaje na zaslonu tako dugo dok igrač ne pritisne jedan od ponuđenih gumbova.

Platno „*ShopCanvas*“ igraču se prikazuje samo ukoliko je dovoljno blizu novog oružja ili vratiju za otvaranje.


Slika 34: Grafičko korisničko sučelje (Izvor: Vlastita slika zaslona, 2021)

#### 3.5.6.1. HealthBar.cs

```
public class HealthBar : MonoBehaviour
{
    public Slider slider;

    public void SetHealth(float health)
    {
        slider.value = health;
    }
}
```

Skripta „*HealthBar.cs*“ uz pomoć komponente „*Slider*“ koja se nalazi na objektu igre „*HealthBar*“ manevrira duljinom crvenog pravokutnika životnih bodova sa slike 34. Ovo se radi na način da se komponenta „*Slider*“ poveže s bojom punjenja ovog pravokutnika i zatim mijenjanjem vrijednosti *value* mijenjamo i dužinu crvenog pravokutnika. Ova funkcija poziva se iz *PlayerHealth.cs* pri svakom mijenjanju životnih bodova.

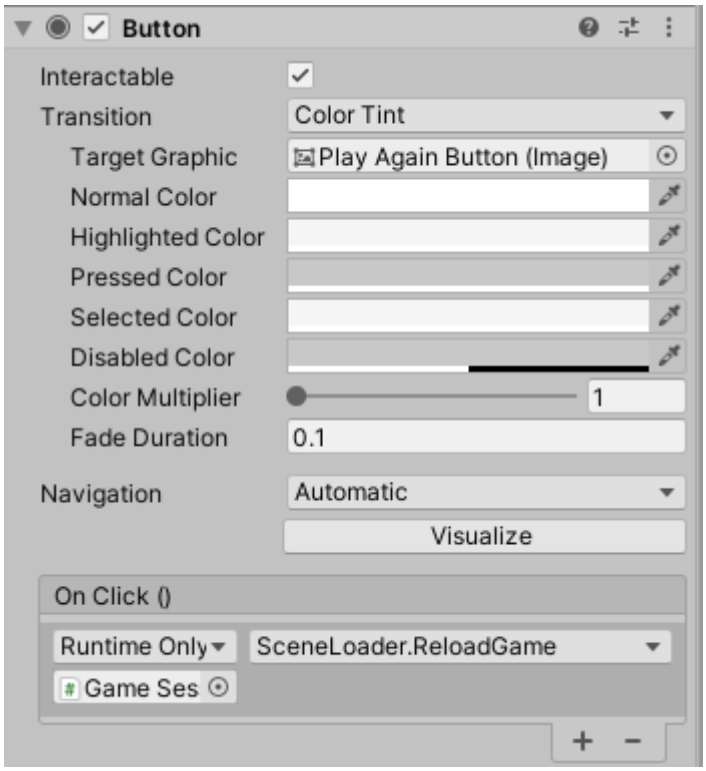
#### 3.5.6.2. Skripta SceneLoader.cs

```
public class SceneLoader : MonoBehaviour
{
    public void ReloadGame()
    {
        SceneManager.LoadScene(1);
        Time.timeScale = 1;
    }
```

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

Skripta „*SceneLoader.cs*“ je komponenta objekta „*GameSession*“ sa slike 33. Pozivanjem funkcija u toj skripti igra se ponovno učitava ili se u potpunosti gasi. Pozivanje ovih funkcija dešava se na gumbovima „*PLAY AGAIN*“ i „*EXIT*“ sa slike 34.

Funkcije se na gumbove postavljaju tako da u komponenti „*Button*“ kreiramo događaj i povežemo ga s funkcijom iz određene skripte, što se vidi na dnu sljedeće slike:




Slika 35: Button komponenta (Izvor: Vlastita slika zaslona, 2021)

## 3.5.7. Instanciranje pri izvođenju

Objekt „*RunTimeInstantiator*“ prazan je objekt kojem je jedina svrha da nosi skriptu „*RunTimeInstantiation*“. Pomoću te skripte zombiji se instanciraju pri izvođenju igre te na taj način postižemo efekt beskonačne igre gdje nema kraja protivnicima.

### 3.5.7.1. Skripta RunTimeInstantiation.cs

```
public class RunTimeInstantiation : MonoBehaviour
{

    [SerializeField] GameObject zombieObject;
    [SerializeField] GameObject playerObject;
    [SerializeField] TextMeshProUGUI waveTextMesh;

    private int zombiesPerWave = 8;
    private int zombieWaveHealth = 100;
    private int zombieWaveDamage = 20;
    private int currentWave = 1;

    [Serializable]
    public struct SpawnCoordinates
    {
        public float x;
        public float y;
        public float z;
    }
    [SerializeField]
    private SpawnCoordinates[] spawnLocation;

    void Start()
    {
        waveTextMesh.text = "Wave " + currentWave;
        InvokeRepeating("InstantiateWave", 0, 42f);
    }

    private void InstantiateWave()
    {
        StartCoroutine("InstantiateZombie");
    }

    IEnumerator InstantiateZombie()
    {
        for (int i = 0; i < zombiesPerWave; i++)
        {
            int randomLocation = UnityEngine.Random.Range(0, spawnLocation.Length);
            GameObject zombieInstance = Instantiate(zombieObject, new
Vector3(spawnLocation[randomLocation].x, spawnLocation[randomLocation].y,
spawnLocation[randomLocation].z), Quaternion.identity);
            zombieInstance.GetComponent<EnemyAI>().target = playerObject.transform;
            zombieInstance.GetComponent<EnemyHealth>().hitPoints = zombieWaveHealth;
            zombieInstance.GetComponent<EnemyAttack>().damage = zombieWaveDamage;

            yield return new WaitForSeconds(20 / zombiesPerWave);
        }

        currentWave++;
        waveTextMesh.text = "Wave " + currentWave;
        if (currentWave <= 6)
        {
            zombiesPerWave+=2;
            zombieWaveHealth += 15;

            if(currentWave == 4)
            {
                zombieWaveDamage += 10;
            }
```

```
        }
        else
        {
            zombieWaveHealth += 20;

            if (currentWave == 8)
            {
                zombieWaveDamage += 10;
            }
        }
    }
}
```

Svrha skripte je instanciranje sve jačih i jačih zombija što se događa na određenim predefiniranim lokacijama. Biranje između predefiniranih lokacija je nasumično.

Ključna riječ [*Serializable*] omogućava uređivanje polja varijabli unutar inspektorskog prozora. Ovdje se ona koristi za uređivanje polja lokacija instanciranja zombija. S obzirom na to da se radi o koordinatama, manevriranje između scenskog prozora i koda bilo bi problematično te se pomoću ove ključne riječi proces ubrzava i olakšava na način da je odmah prisutan u inspektorskom prozoru. [43]

U funkciji *Start()* tekst na *HUD* platnu postavlja se na „Wave 1“ i pokreće se *InvokeRepeating* funkcija kojom pozivamo metodu navedenu u argumentima svakih N sekundi. Funkcija koju ćemo ponavljati je *InstantiateWave*, tj. instanciranje vala i ona će se pozivati beskrajno svakih 35 sekundi. [44]

Funkcija *InstantiateWave()* samo poziva prije objašnjenu korutinu koja instancira jedan zombi. Pri ulasku u korutinu odmah se ulazi u *for* petlju koja se ponavlja *zombiesPerWave* puta. Ovo je zato što je blok naredbi unutar *for* petlje zadužen za instanciranje pojedinog zombija i postavljanje njegovih svojstava kao što su životni bodovi i šteta. Lokacija na kojoj se zombi instancira je nasumična između četiri određene lokacije.

Nakon što *for* petlja završi, varijabla *currentWave* se inkrementira i provjerava kako bi zombiji ojačali. Količina i životni bodovi zombija povećavaju se sve do petog vala nakon čega se nastavljaju povećavati samo životni bodovi zombija. Na četvrtom i osmom valu zombiju se povećava šteta.

# 4. Zaključak

Unity je veoma moćan alat za izradu širokog spektra žanrova igara. Ističe se svojom sposobnošću *crossplatforminga* pošto mu je struktura identična za više podržanih platformi što omogućava da projekt bude izgrađen na drugu platformu i to ponekad bez ikakvih promjena. Ovu sposobnost možemo vidjeti na igri *Hearthstone* koja je igriva na računalima, macintoshima i mobitelima.

Skladište imovine omogućuje da čak i ljudi koji ne znaju ništa o programiranju ili modeliranju izrade svoju igru kroz *outsourcing* i veliku pristupačnost modela, skripti, zvukova i mapa za sve žanrove igara. Primjer ovakve igre je „*The First Tree*“ u kojoj je glavna tema autobiografija kreatora. U njoj kreator navodi kako ne zna ni prvu stvar o programiranju, no to ga nije spriječilo da stvori veoma zanimljivu igru.

Pri razvoju igre bitno je započeti dizajnom u koji se mora utrošiti dovoljna količina vremena kako bi se postavili temelji igre. Nakon što su temelji igre kao jezgra igre osmišljeni, potrebno je i protumačiti kako će izgledati iskustvo igrača u igri. Bez dizajna igre, pri razvoju bi moglo doći do kontradikcija u funkcionalnostima što bi kreatoru stvorilo poteškoće zbog kojih bi se razvoj igre morao vratiti u fazu dizajna.

Bez programa kao što je Unity, indie programeri ne bi imali šanse u borbi na tržištu protiv divova kao što su Sony, Tencent, Nintendo i Activision. Skup alata koji omogućava izvanrednu brzinu razvoja igara za individualne programere koji u prošlosti nisu bili mogući, dozvoljavaju da bilo tko ima šansu svoju ideju pretvoriti u pravu igru unutar razumne količine vremena.

# Popis literature

[1] Nepoznat autor, „Video Game History“, 01.09.2017. [Na internetu]. Dostupno na: https://www.history.com/topics/inventions/history-of-video-games . [Pristupljeno: 02.08.2021.].

[2] R. Chikhani, „The History Of Gaming: An Evolving Community“, 31.10.2015. [Na internetu]. Dostupno na: https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/ . [Pristupljeno 02.08.2021.].

[3] J. Clement, „Global video game market value from 2020. to 2025.“, 01.06.2021. [Na internetu].
Dostupno na: https://www.statista.com/statistics/292056/video-game-market-value-worldwide/ . [Pristupljeno 02.08.2021.].

[4] A. Shiddiq, „Why does a game need a game engine?“, 03.05.2017. [Na internetu]. Dostupno na: https://www.quora.com/Why-does-a-game-need-a-game-engine/answer/Andy-Shiddiq . [Pristupljeno 02.08.2021.].

[5] Nepoznat autor, „Cross-Platform Considerations“, 03.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/CrossPlatformConsiderations.html [Pristupljeno: 06.08.2021.].

[6] Nepoznat autor, „Unity (game engine)“, ažurirano 04.08.2021. [Na internetu]. Dostupno na: https://en.wikipedia.org/wiki/Unity_(game_engine) . [Pristupljeno 06.08.2021.].

[7] Ryan Cheu, „Are there any big game studios that uses Unity3D?“, 14.01.2016. [Na internetu].
Dostupno na: https://www.quora.com/Are-there-any-big-game-studios-that-uses-Unity3D/answer/Ryan-Cheu . [Pristupljeno 06.08.2021.].

[8] Nepoznat autor, „Installing the Unity Hub“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/GettingStartedInstallingHub.html

[Pristupljeno: 08.08.2021.].

[9] Nepoznat autor, „Starting Unity“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/GettingStarted.html [Pristupljeno: 08.08.2021.].

[10] Nepoznat autor, „Unity's interface“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/UsingTheEditor.html [Pristupljeno: 08.08.2021.].

[11] Nepoznat autor, „The Toolbar“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/Toolbar.html [Pristupljeno: 08.08.2021.].

[12] Nepoznat autor, „The Hierarchy window“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/Hierarchy.html [Pristupljeno: 08.08.2021.].

[13] Nepoznat autor, „The Game view“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/GameView.html [Pristupljeno: 08.08.2021.].

[14] Nepoznat autor, „The Scene view“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/UsingTheSceneView.html [Pristupljeno: 09.08.2021.].

[15] Nepoznat autor, „Scene view control bar“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/ViewModes.html [Pristupljeno: 09.08.2021.].

[16] Nepoznat autor, „The inspector window“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/UsingTheInspector.html [Pristupljeno: 09.08.2021.].

[17] Nepoznat autor, „The Project window“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/ProjectView.html [Pristupljeno: 09.08.2021.].

[18] Nepoznat autor, „Unity's Asset Store“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/AssetStore.html [Pristupljeno: 09.08.2021.].

[19] Patch Quest, „How NOT to make an indie game“, 05.09.2020. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=NnI_1DOYt2A [Pristupljeno: 09.08.2021.]

[20] Nepoznat autor, „5 Steps to designing core gameplay mechanics“, 01.07.2019. [Na internetu]. Dostupno na: https://astudiolite.com/5-steps-to-designing-core-gameplay-mechanics/ [Pristupljeno: 16.08.2021.]

[21] Dave Eng, „The Player Experience“, 10.09.2019. [Na internetu] . Dostupno na: https://www.universityxp.com/blog/2019/9/10/the-player-experience
[Pristupljeno: 16.08.2021.]

[22] Nepoznat autor, „Engagement Curves“, 01.03.2020. [Na internetu]. Dostupno na: https://www.zenrhino.org/theory/curves [Pristupljeno: 16.08.2021.]

[23] Nepoznat autor, „Shooter game“, 28.08.2021. [Na internetu]. Dostupno na: https://en.wikipedia.org/wiki/Shooter_game#Subgenres [Pristupljeno: 16.08.2021.]

[24] Nepoznat autor, „Building a NavMesh“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html [Pristupljeno: 16.08.2021.].

[25] Daniel Cook, „A practical definition of innovation in game design“, 03.04.2005. [Na internetu] . Dostupno na: https://lostgarden.home.blog/2005/04/03/a-practical-definition-of-innovation-in-game-design/ [Pristupljeno: 16.08.2021.]

[26] Nepoznat autor, „What is the purpose of the Empty GameObject?“, 17.06.2017. [Na internetu]. Dostupno na: https://community.gamedev.tv/t/what-is-the-purpose-of-the-empty-gameobjec t/33846 [Pristupljeno: 16.08.2021.]

[27] Nepoznati autor, „Rigidbody“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/Rigidbody.html [Pristupljeno: 16.08.2021.]

[28] Nepoznati autor, „CapsuleCollider“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/CapsuleCollider.html [Pristupljeno: 16.08.2021.]

[29] Nepoznati autor, „RigidBody FPS Controller(standard assets) gets stuck on mesh colliders“, 14.05.2017. [Na internetu]. Dostupno na:

https://answers.unity.com/questions/1352682/rigidbody-fps-controllerstandard-assets-gets-stuck.html [Pristupljeno: 26.08.2021.]

[30] Nepoznati autor, „MonoBehaviour“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html [Pristupljeno: 26.08.2021.]

[31] Nepoznati autor, „MonoBehaviour.InvokeRepeating“, 06.08.2021. [Na internetu]. Dostupno na:

https://docs.unity3d.com/ScriptReference/MonoBehaviour.InvokeRepeating.html [Pristupljeno: 26.08.2021.]

[32] Nepoznati autor, „Time“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/Time.html [Pristupljeno: 26.08.2021.]

[33] Nepoznati autor, „MonoBehaviour.Update()“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html [Pristupljeno: 27.08.2021.]

[34] Nepoznati autor, „MonoBehaviour.OnEnable()“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnEnable.html [Pristupljeno: 27.08.2021.]

[35] Nepoznati autor, „MonoBehaviour.StartCoroutine()“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html [Pristupljeno: 27.08.2021.]

[36] Nepoznati autor, „Physics.Raycast“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [Pristupljeno: 27.08.2021.]

[37] Nepoznati autor, „NavMeshAgent“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html [Pristupljeno: 27.08.2021.]

[38] Nepoznati autor, „Object.Destroy“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/Object.Destroy.html [Pristupljeno: 27.08.2021.]

[39] Nepoznati autor, „How do you update NavMesh rotation after stopping distance is reached?“, 19.09.2013. [Na internetu]. Dostupno na: https://answers.unity.com/questions/540120/how-do-you-update-navmesh-rotation-after-stopping.html [Pristupljeno: 27.08.2021.]

[40] Nepoznati autor, „Using Animation Events“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/Manual/script-AnimationWindowEvent.html [Pristupljeno: 28.08.2021.]

[41] Nepoznati autor, „Collider.isTrigger“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/Collider-isTrigger.html [Pristupljeno: 28.08.2021.]

[42] Nepoznati autor, „What is the difference between a HUD and a GUI in a game?“, 16.11.2017. [Na internetu]. Dostupno na:

https://gamedev.stackexchange.com/questions/150987/what-is-the-difference-between-a-hud-and-a-gui-in-a-game [Pristupljeno: 28.08.2021.]

[43] Nepoznati autor, „Serializable“, 06.08.2021. [Na internetu]. Dostupno na: https://docs.unity3d.com/ScriptReference/Serializable.html [Pristupljeno: 28.08.2021.]

[44] Nepoznati autor, „MonoBehaviour.InvokeRepeating“, 06.08.2021. [Na internetu]. Dostupno na:

https://docs.unity3d.com/ScriptReference/MonoBehaviour.InvokeRepeating.html [Pristupljeno: 28.08.2021.]

# Popis slika

Popis slika treba biti izrađen po uzoru na indeksirani sadržaj, te upućivati na broj stranice na kojoj se slika može pronaći.

# Prilozi (1, 2, …)

Prilog 1: Tečaj - B. Tristem, R. Davidson, „Complete C# Unity Game Developer 3D“, 06.2021. [Na internetu]. Dostupno na: https://www.udemy.com/course/unitycourse2/ . [Pristupljeno: 30.08.2019.]