Izrada videoigre obrane dvorca

Brzoja, Marin

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marin Brzoja

Izrada videoigre obrane dvorca

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marin Brzoja

Matični broj: 0016136145

Studij: Informacijski sustavi

Izrada videoigre obrane dvorca

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Mario Konecki

Varaždin, rujan 2022.

Marin Brzoja

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog završnoga rada jest izrada 2D (dvodimenzionalne) računalne videoigre unutar alata Unity. Rad se bavi praktičnom izradom videoigre kao i teorijskim dijelom u kojem se govori o procesu izrade videoigre. Unutar praktičnog dijela će se opisati izrada grafike, animacija, likova, mehanika i svega onog potrebnoga da bi igra bila zanimljiva i igriva. Za ovu igru se unutar alata Unity koristi programski jezik C# jer je on potreban da se stvore skripte koje kasnije koristimo za kretanje, fiziku, likove, animacije i grafiku. Unutar samog rada objašnjen je način rada na koji se stvaraju objekti u Unity alatu, za što se koriste i kako se vrši međusobna komunikacija između tih stvorenih objekata.

Ključne riječi: AI (Artificial Intelligence), Mehanika, Dizajn, Grafika, Programiranje igre, Videoigra, Unity alat, C#

Sadržaj

1. Uvod 1
2. Metode i tehnike rada 2
3. Alat Unity 3
3.1. Povijest Unity 3
3.2. UI (korisničko sučelje) 5
3.2.1. Traka sa alatima (eng. Toolbar) 5
3.2.2. Hijerarhijski prozor (eng. Hierarchy window) 6
3.2.3. Prikaz scene (eng. Scene view) 7
3.2.4. Prikaz igre(eng. Game view) 7
3.2.5. Inspekcijski prozor (eng. Inspector window) 8
3.2.6. Prozor projekta (eng. Project window) 9
3.2.7. Prozor konzole (eng. Console window) 9
4. Razvoj igre 11
4.1. Dizajn mape 11
4.1.1. Paleta pločica (eng. Tile palette) 12
4.2. Dizajn korisničkog sučelja 15
4.3. Razvoj lika 16
4.3.1. Dizajn lika 16
4.3.1.1. Spritesheet 17
4.3.1.2. Sprite renderer 18
4.3.1.3. Player Input 18
4.3.1.4. Rigidbody 2D 19
4.3.1.5. Box Collider 2D 19
4.3.1.6. Animator 20
4.3.2. Skripte lika 21
4.3.3. Dizajn i skripte oružja lika 24
4.3.4. Dizajn i skripte neprijatelja 26
4.3.4.1. Skripta neprijatelja 28
5. Zaključak 31
6. Popis literature 32
7. Popis slika 33

1. Uvod

Ovaj završni rad bazira se na teorijskom i praktičnom dijelu izrade 2D videoigre. Radi se o videoigri žanra *obrane dvorca (eng. Castle defense)* gdje je igrač smješten u limitirani prostor iz kojega ne može otići već se mora braniti od najezde neprijatelja koji želi ući u njegov prostor i načiniti mu štetu. Igrač se mora braniti sa oružjem kojeg ima od početka te je sama igra gotova kada su svi neprijatelji pobijeđeni ili je igra gotova u onom trenutku kada igrač izgubi svoje životne bodove jer se nije mogao obraniti od neprijatelja. Sama igra je dvodimenzionalna te omogućuje kretanje u samo četiri smjera te njihove kombinacije.

Unutar rada objašnjen je alat u kojem je igra napravljena, alat *Unity* i funkcionalnosti vezane uz sam alat kojima je igra izgrađena. Prolaziti će se kroz različite prozore i njihove funkcije i opcije. Detaljno su objašnjene izrade likova, mapa, animacija, scena, skripti i kontrolera koji u svojoj interakciji čine cjelinu igre. Same skripte su napisane u programskom jeziku *C#* unutar programskog okruženja *Microsoft Visual Studio 2022.* Čitateljima se želi pobliže pojasniti izrada videoigara i mogućnosti alata *Unity* te ujedno prikazati kôd same igre koja biva napravljena.

Motivacija zbog koje je odabrana tema ovog završnoga rada jest programiranje te interes za videoigre i cijeli žanr videoigara te ujedno i želja za učenjem novih stvari. Ujedno mi je ova tema omogućila priliku da se okušam u području dizajna te sam ovim radom uspio stvoriti programski kôd koji nije samo apstraktan već je lako razumljiv i objašnjen u cijelosti.

2. Metode i tehnike rada

Kako bi se videoigra napravila potrebno je mnogo teorijskog znanja ne samo o videoigrama, već i o tehnikama i programima koje se koriste kako bi se one mogle stvoriti. Zato je za početak bilo potrebno proučiti nekoliko znanstvenih radova, videa i priručnika kako bi se dobila dovoljno dobra podloga za pravljenje ovoga rada te nakon što su se proučile informacije iz tih izvora trebaju se izvući najkorisnije stavke koje najviše odgovaraju u izradi nivoa, likova, animacija i ostalih 2D objekata koji se koriste unutar same igre kako bi dobili kompletnu cjelinu. Nakon što se sve potrebne informacije sakupe o izradi igre, potrebno je proučiti sami alat *Unity* unutar kojega se obavlja sama izrada igre.

Unity je moćan alat koji se koristi ne samo za 3D i 2D izradu igara već je ono i IDE (engl. integrated development environment – integrirano okruženje za razvoj). Unutar samoga alata *Unity* se nalazi dio zvan „Asset store“ koji se koristio pri izradi igre tako što su se uzele besplatne kreacije drugih kreatora kako bi se poboljšao izgled same igre. Tokom izrade unutar *Unity-ja* potrebno je praviti razne skripte preko kojih se obavljaju ostale funkcije alata *Unity* poput kolizije, detekcije, triggera i manipulacije podataka likova, a te skripte su se razvijale koristeći programski jezik *C#* unutar programskog okruženja *Microsoft Visual Studio* unutar kojeg su bile predefinirane klase i funkcije koje su olakšale rad nad objektima.

Neke grafičke komponente su preuzete sa internetskih izvora poput *Unity Asset store* dok su ostale preuzete sa stranica neovisnih kreatore koje svoje materijale izlažu besplatno za upotrebu. Za ostale grafičke komponente se koristio alat *Piskel* koji služi za izradu i bojanje Spriteova(grafičkih znakova, sličica) u piksel formatu. Za uređivanje teksta se koristio *TextMeshPro* i *Text(legacy)* unutar samog alata *Unity.*

3. Alat Unity

U ovom dijelu objašnjen je način rada sa alatom *Unity* te ujedno i funkcionalnosti koje su korištene pri izradi videoigre. Alat *Unity* sam po sebi je vrlo kompleksan zbog količine informacija i znanja potrebnog za korištenje alata dok se kreira videoigra jer je *Unity* zapravo skup alata koji su međusobno povezani i zajedno se moraju koristiti kako bi završni proizvod bio funkcionalan.

Osim što je *Unity* game engine softver ono je ujedno i IDE (eng. integrated development environment, hrv. Integrirano programsko okruženje) jer svojim korisnicima opisuje korisničko sučelje i omogućuje svojim korisnicima pristup i korištenje svih potrebnih alata za razvoj.

Unity ujedno spaja nekoliko područja preko kojega se mogu ubacivati vlastiti 2D i 3D radovi, slike, animacija zvukova, dodavanje svjetla, logike igre, uređivanja sadržaja i optimizacije izrađeni unutar ostalih programa poput Photoshop-a, Blender-a, Audacity-ja i sl. Ti radovi se sa lakoćom mogu prenijeti u Unity i sa njima stvarati ili ih samo dodati u scenu ili okruženje.

Unity je zbog svoje sveobuhvatnosti alata i visoke primjene tehnologija ne samo nad žanrom videoigara već i video animacija i simulacija postao jedan od najpopularnijih game engine-a na današnjem tržištu. [1] Neki od naslova razvijenih unutar alata *Unity*:

- Fall Guys: Ultimate Knockout (2020.)
- Among Us (2018.)
- Untitled Goose Game (2019.)
- Monument Valley 2 (2014.)
- Ori and the Will of the Wisps (2020.)
- Cuphead (2017.)
- Pokémon Go (2016.)

Unutar ovog ili kasnijih dijelova rada biti će detaljno prikazan razvoj pojedinačnih dijelova alata i igre.

3.1. Povijest Unity

[2] Kreatori *Unity-ja* David Helgason, Nicholas Francis, i Joachim Ante osnivali su kompaniju „Unity Technologies“ 2004. godine u Kopenhagenu. Cilj im je bio stvoriti pristupačan

game engine sa profesionalnim alatima za amaterske programere igara tako da ostvare "demokratizaciju razvoja igara" sa svojim alatom.

Isprva su htjeli živjeti od izrade igara, ali na svom putu su vidjeli manu sa tadašnjom tehnologijom te su uvidjeli potrebu za izradom vlastite tehnologije. Njih troje uvijek su mislili da će “napraviti igricu i onda izdati licencu za tehnologiju" i da je "igra bila neophodna za dokazivanje tehnologije". Na kraju nisu izradili igru, već su stvorili "alata za izradu igara".

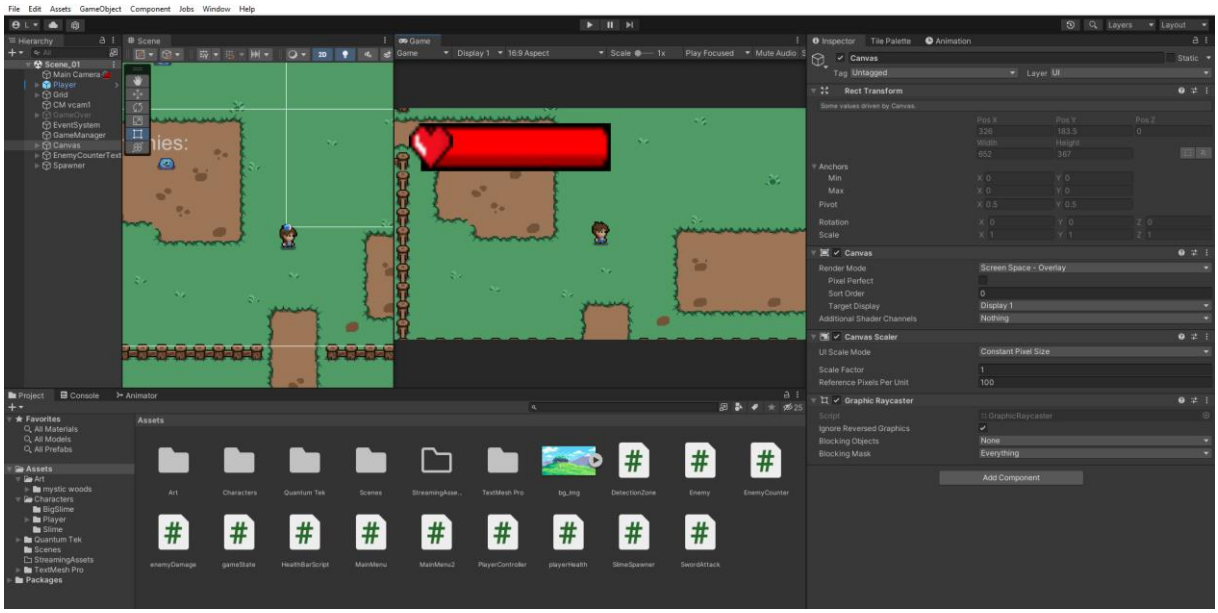
U svojim počecima bivao je korišten samo za Mac iOS kao njihov softver za igre, međutim tijekom svog razvoja i rastom u popularnosti je porastao u alat koji podupire mnogobrojne ostale platforme poput stolnih računala, mobilnih uređaja, konzola i platformi virtualne stvarnosti.

Kroz svoje postojanje *Unity* alat imao je nekoliko verzija svog programa, svaka sa svojim specifičnim značajkama:

- Unity 2.0 (2007.) - Izdanje je uključivalo optimizirani game engine terena za detaljna 3D okruženja, dinamičke sjene u stvarnom vremenu, usmjerena svjetla i reflektore te video reprodukciju
- Unity 3.0 (2010.) – izdano proširenje grafičkih značajki game engine-a za stolna računala i konzole za videoigre
- Unity 4.0 (2012.) – izdani alati koji su dopuštali praćenje reklamnih kampanja i dubinsko povezivanje, gdje su korisnici bili izravno povezani s postova na društvenim mrežama na određene dijelove unutar igrica i jednostavno dijeljenje slika u igri
- Unity 5 (2015.) - nudi globalno osvjetljenje u stvarnom vremenu, preglede mapiranja svjetla, Unity Cloud, novi audio sustav i Nvidia PhysX 3.3 motor za fiziku te se kroz WebGL su *Unity* developeri mogli bez problema postaviti svoje igre na web pretraživač
- Unity (2017.) – posljednja verzija *Unity* platforme nudi nove alate kao što je Timeline, koji je programerima omogućio povlačenje i ispuštanje animacija u igre, i Cinemachine, sustav pametne kamere unutar igara i broje vizualne skripte i sisteme koji prate te skripte

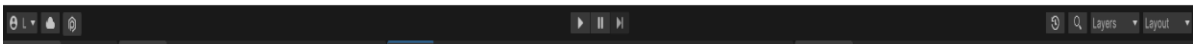
3.2. UI (korisničko sučelje)

U nastavku ovoga rada slijedi detaljan opis sučelja programskog alata *Unity* i njegovih glavnih komponenata. Te komponente su: traka sa alatima, prozor hijerarhija, prozor prikaza igre, prozor prikaza scene, prozor inspektora, prozor projekta, prozor konzole. [3]



Slika 1 - Korisničko sučelje programa Unity

3.2.1. Traka sa alatima (eng. Toolbar)



Slika 2 - Traka sa alatima

Traka sa alatima pruža pristup Unity računu korisnika, Unity Cloud Servisima, kontrolama nad reprodukcijom, pauzom i kontrolom nad koracima, Plastic SCM, povratak povijesti, te Unity Search, izborniku vidljivosti slojeva i izborniku rasporeda uređivača (koji pruža neke alternativne izglede za prozore uređivača i omogućuje spremanje vlastitih prilagođenih rasporeda).

 -omogućava pristup Unity računu

 - otvara prozor koji omogućava pristup Unity servisima

 - otvara prozor za „Plastic SCM“ kojime se prikazuje popis novih ili promijenjenih datoteka unutar lokalnog projekta

 - kontrola nad reprodukcijom, pauzom i koracima scene u pogledu igre (Game view)

 - pruža vizualni način kretanja kroz povijest poništavanja ili ponavljanje više radnji u uređivaču odjednom

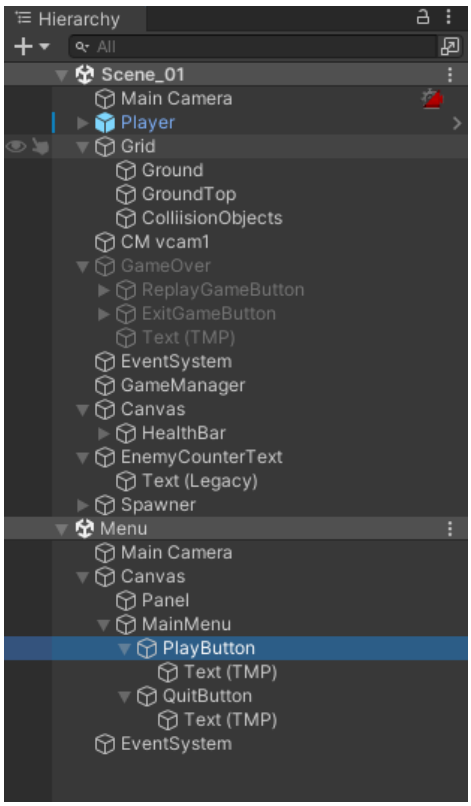
 - korisniku omogućava pretraživanje tražene stavke na globalnoj razini, može se pretraživati po labeli, imenu ili tipu objekta koji je tražen

- omogućava korisniku mogućnost upravljanja i odvajanja objekata u sceni igre pomoću slojeva

- omogućava korisniku kontrolu nad prozorima programa Unity te ih može prilagoditi po preferenci i spremanje takvog rasporeda za kasnije korištenje

3.2.2. Hijerarhijski prozor (eng. Hierarchy window)

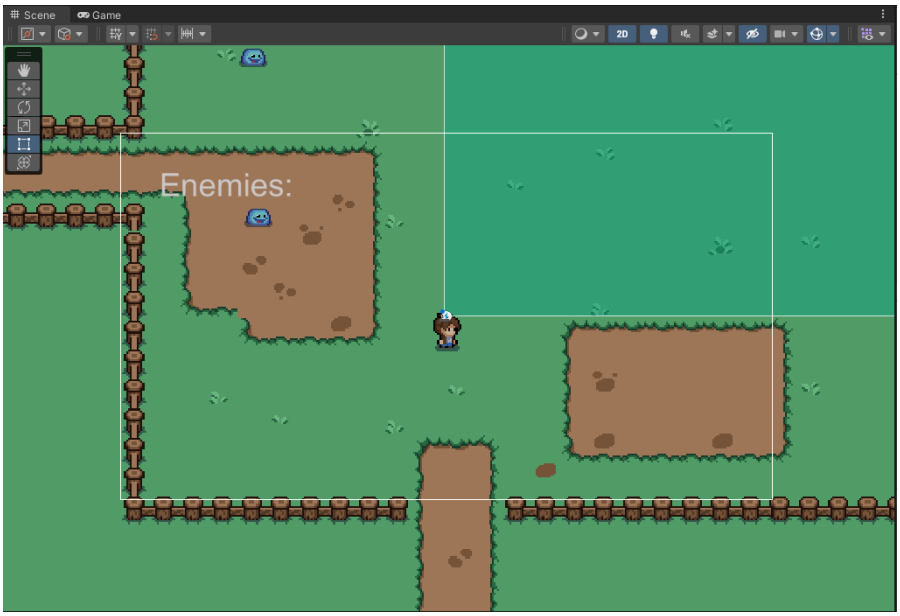
Hijerarhijski prozor prikazuje svaki objekt igre (GameObject) u sceni, kao što su modeli, kamere ili montažne jedinice(eng. Prefabs). Može se koristiti prozor Hijerarhije za sortiranje i grupiranje objekata igre koje se koriste u sceni. Unity koristi koncept hijerarhije roditelj-dijete. Kada se dodaju ili uklanjaju objekti igre u prikazu scene, također se dodaju ili uklanjaju iz prozora hijerarhije. Hijerarhijski prozor također može sadržavati druge scene, sa svakom scenom koji sadrže svoje vlastite objekte.



Slika 3 - Hijerarhijski prozor

3.2.3. Prikaz scene (eng. Scene view)

Prikaz scene se koristi za postavljanje krajolika, likova, kamera, svjetla i bilo koje druge vrste objekata igre. Unutar tog pogleda se odabiru, manipuliraju i mijenjaju objekti igre kako bi se mogla napraviti igrica.

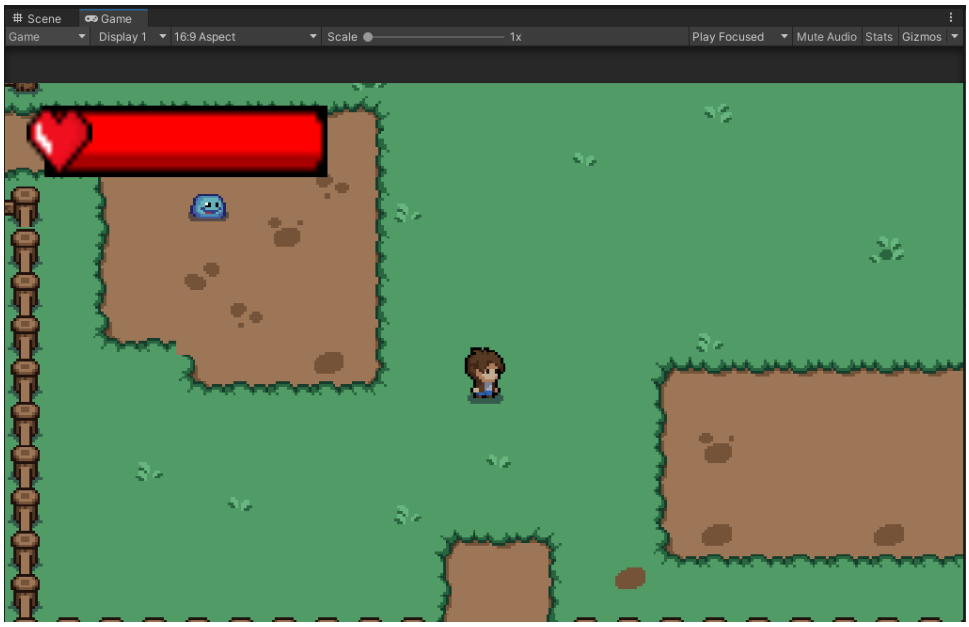


Slika 4 - Prikaz scene (eng. Scene view)

Jedna od najbitnijih značajki prikaza scene jest kamera pomoću koje vidimo ono što sami igrač vidi tj. ono što korisnik vidi u danom trenutku. Pomoću slojeva odnosno gumba *Layers* određujemo hijerarhiju vidljivosti, tj. koji objekti bivaju na vrhu, koji u pozadini ili jednostavno koji objekti imaju prioritet da budu vidljivi.

3.2.4.Prikaz igre(eng. Game view)

Prikaz igre omogućuje prikazati završnu verziju igre, tj. kako će igra izgledati kada ona bude pokrenuta. Sami prikaz igre možemo postaviti kroz kamere scene, a sama simulacija prikaza započinje klikom gumba *Reproduciraj (eng. Play).*

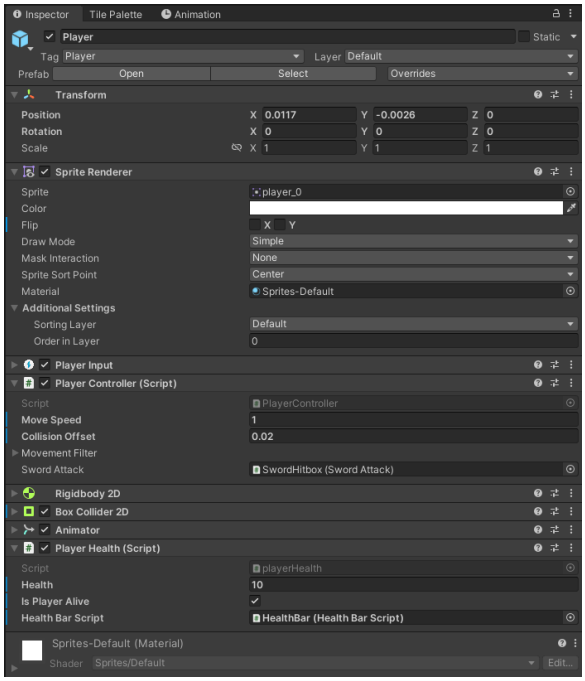


Slika 5 - Prikaz igre (eng. Game view)

U načinu reprodukcije sve promjene koje bivaju napravljene su privremene i poništavaju se kada se izađe iz načina reprodukcije.

3.2.5. Inspekcijski prozor (eng. Inspector window)

Inspekcijski prozor omogućava pregled i uređivanje svojstava i postavki za gotovo sve unutar Unity Editora, uključujući objekte igre, komponente alata *Unity,* materijale unutar datoteke „Assets“ i postavke odabrane stavke i preferenci u uređivaču.

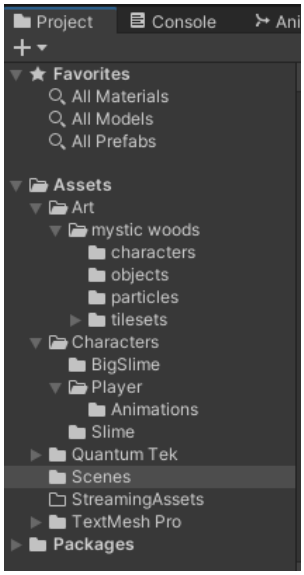


Slika 6. - Inspekcijski prozor (eng. Inspector window)

Unutar slike 6 nalazi se pogled na inspekcijski prozor lika „*Player*“ u igri unutar kojega možemo vidjeti dijelove i komponente od kojih se sastoji poput: njegove pozicije, Sprite renderer-a, Player input-a, Rigidbody 2D, Box Collider 2D, Animator-a; i dvije skripte: *Player health i Player Controller*. Svi dijelovi će biti objašnjeni kasnije u radu.

3.2.6.Prozor projekta (eng. Project window)

Prozor projekta prikazuje sve datoteke povezane sa projektom i glavni je način navigacije između datoteka i pronalaska svih potrebnih materijala koji će biti korišteni u izradi rada.

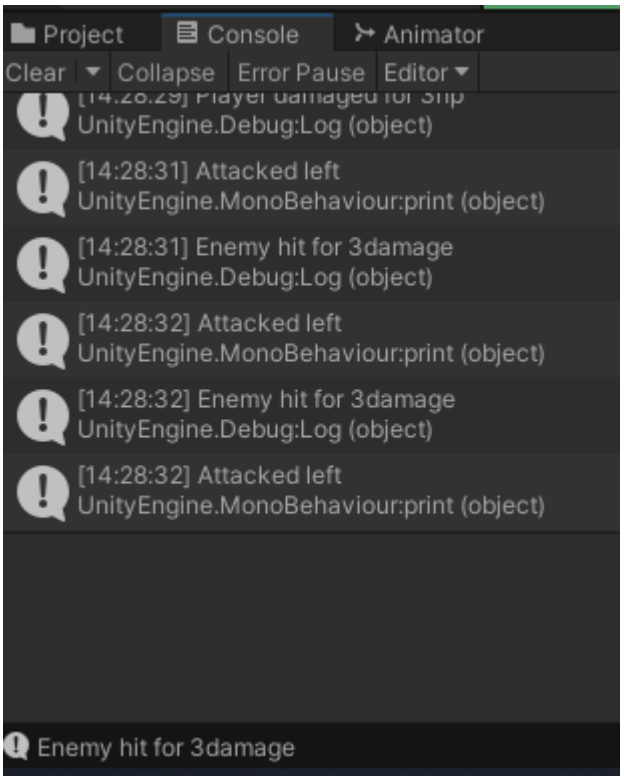


Slika 7 - Prozor projekta (eng. Project window)

Datoteke se mogu dodavati u projekt tako da se "dovuku" željene datoteke u njega koristeći takozvanu „drag and drop“ metodu.

3.2.7. Prozor konzole (eng. Console window)

Prozor konzole se nalazi na dnu i prikazuje sve greške, poruke i upozorenja koje generira *Editor.* Prozor konzole se može koristiti prije i tijekom izvođenja programa kako bi se pronašle i riješile greške unutar projekta.



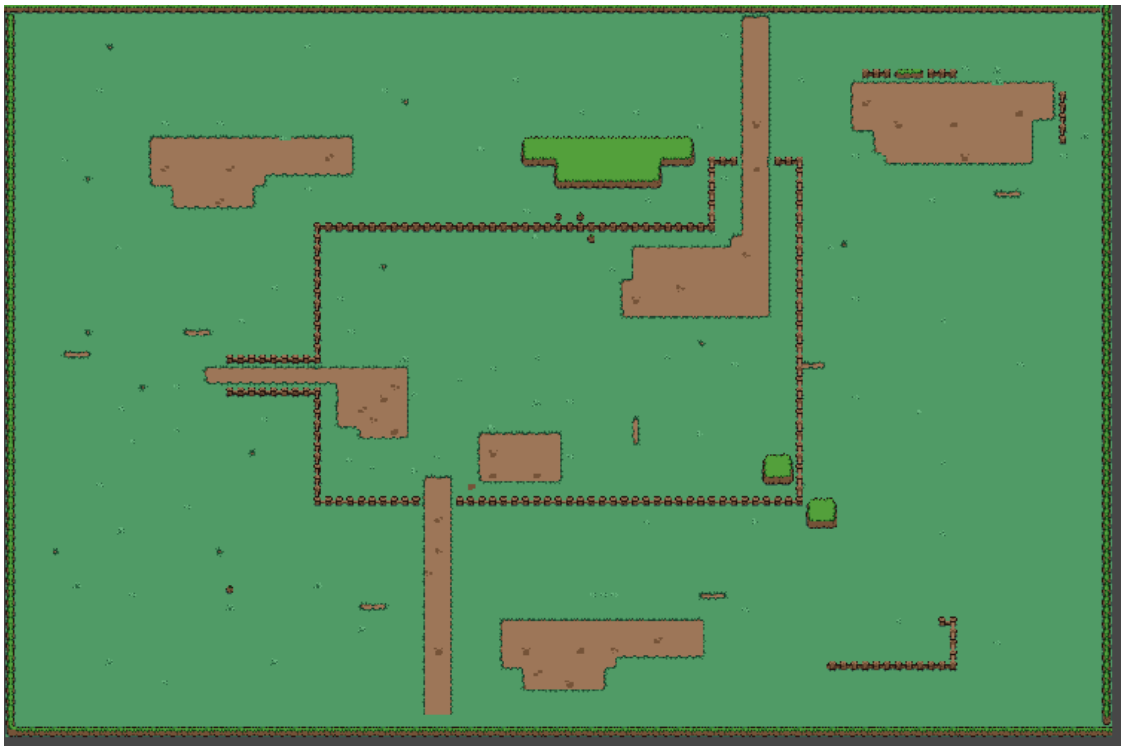
Slika 8 - Prozor konzole (Console window)

4. Razvoj igre

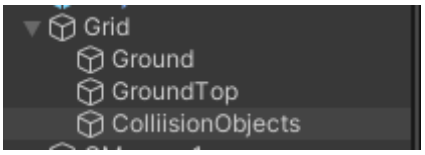
U ovom će se poglavlju objasniti cjelokupan proces izrade 2D videoigre koja započinje sa kreiranjem i postavljanjem scene tj. mape na kojoj će se nalaziti igrač, neprijatelji i neprelazivi objekti. Zatim će se pobliže objasniti pravljenje igrača, njegovih kretnji, napada i animacija. Zatim slijedi objašnjenje i izrada neprijatelja, njihovih animacija i njihova umjetna inteligencija. Na kraju će se pojasniti izrada menija kojemu igrač pristupa prije nego što započne igru. Sve skripte su napisane unutar okruženja *Microsoft Visual Studio 2022.* programskim jezikom *C#*, te će se objasniti njihov kôd . Za kraj će biti objašnjeni događaja odnosno *Eventi* koji nastupaju i koji prikazuju igraču što se događa u sceni.

4.1. Dizajn mape

Dizajn je vrlo bitan segment svake videoigre jer je prvo što igrač vidi na ekranu i stječe određeni dojam o igri. U dizajn spada izrada same mape i programska logika kako ona interakciju ima sa igračem.



Slika 9 - Mapa igre (vlastita izrada)



Slika 10 - Mapa igre u hijerarhiji projekta

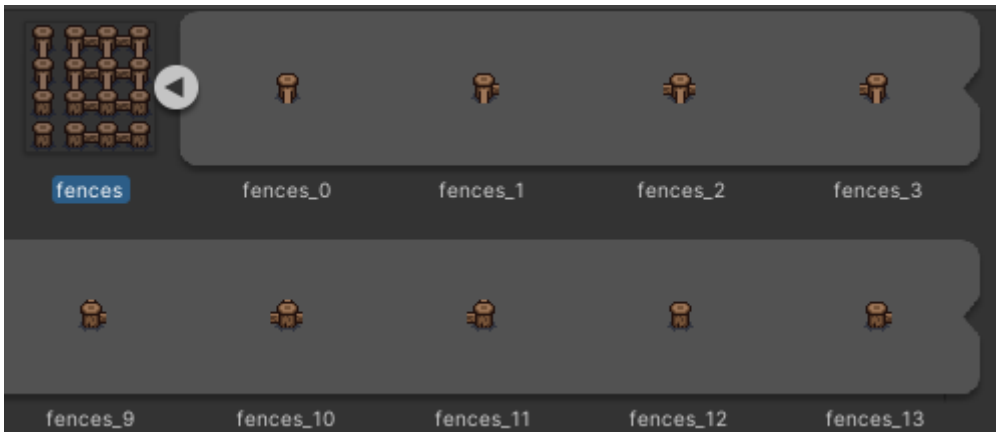
Mapa igre se sastoji od 3 sloja kojima je roditelj prazni objekt zvani „Grid“ koji ih sve sjedinjuje. Svaki od slojeva unutar sebe sadrži takozvane „pločice“ pomoću kojih je stvorena mapa.

4.1.1. Paleta pločica (eng. Tile palette)

[4] Paleta pločica je bitan dio *Unity* game engine-a pomoću kojeg se mogu kreirati mape vlastite izrade. Svaka pločica je napravljena koristeći slike iz datoteke *Assets* nad kojima koristimo „Sprite Editor“ pomoći kojega režemo pojedine dijelove kako bi dobili zasebne sličice.

Slika 11 - Sprite Editor

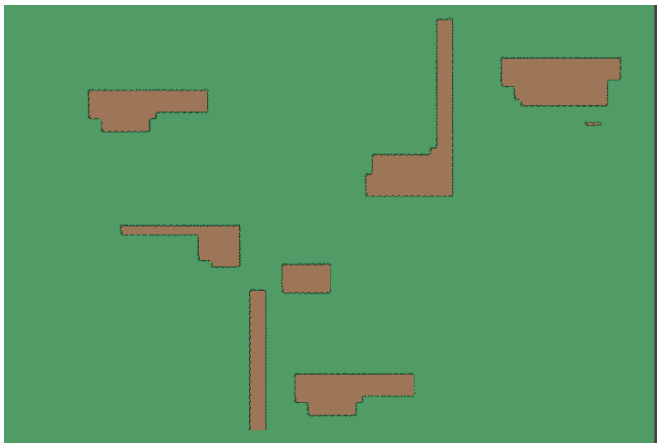
Pritiskom na gumb „Slice“ odabire se opcija kojom se žele izrezati sličice na željeni način te se pritišće gumb „Apply“ u gornjem desnom kutu kako bi se sličice izrezale i stvorile u datoteci projekta ispod originalne slike.



Slika 12 - Sprite Editor, izrezane sličice

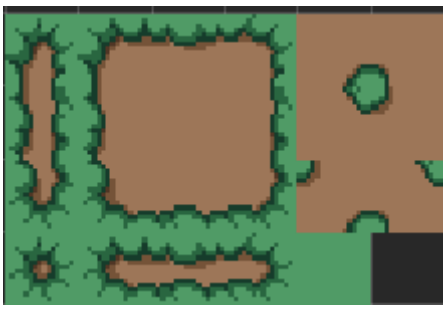
Slika 13 - Pločice igre(tiles) unutar palete pločica

Unutar „Ground“ objekta se nalazi takozvani „donji dio“ mape na kojeg se smještaju gornji dijelovi i zidovi. Primarno je sačinjen od pločica zemlje i trave različitog smještaja i jednostavnog je izgleda te je građenje mape po želji vrlo intuitivno.



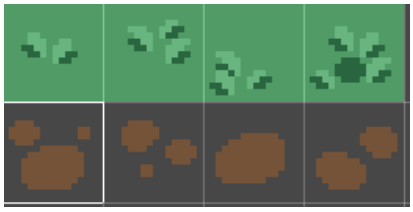
Slika 14 - Ground sloj mape

Ujedno je postavljen poredak sloja objekta „Ground“ na negativnu vrijednost od -90 kako bi se osiguralo da je ono na dnu u hijerarhiji prikaza onoga što je na sceni.



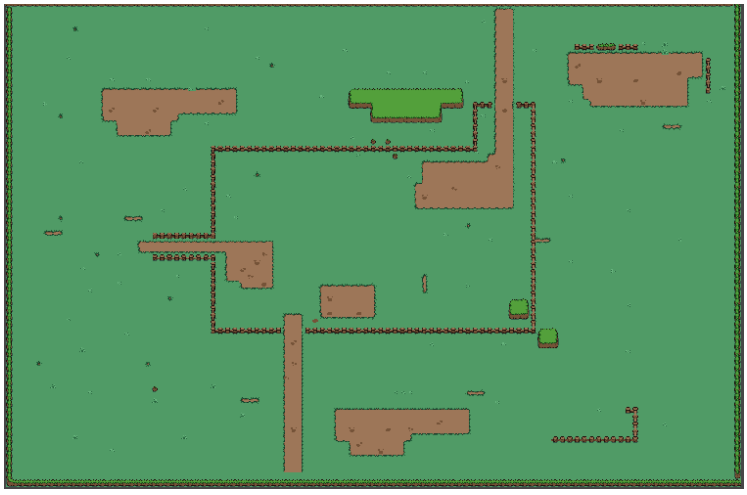
Slika 15 – Skup sličica "Ground" dijela mape

Unutar „GroundTop“ dijela nalaze se pločice koje „stoje“ na donjem dijelu mape tj. pločice sa kamenjem i cvijećem koji daje raznolikost prikaza mape. Ono ima položaj sloja postavljen na -80 kako bi stajalo nad „Ground“ slojem uz mogućnost da se zidovi tj. kolizijski objekti i dalje mogu postaviti nad taj sloj.

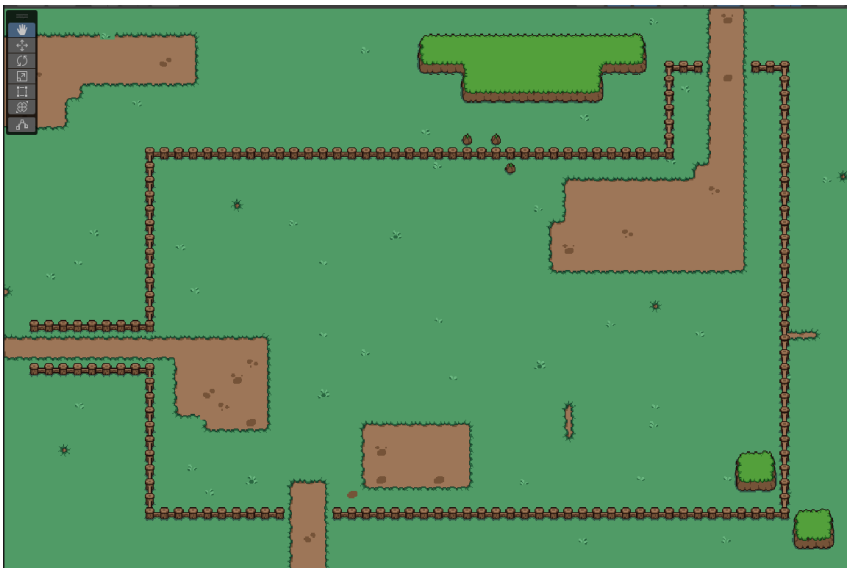


Slika 16 - "GroundTop" dio mape

Posljednji sloj je sloj „CollisionObjects“ unutar kojeg se nalaze sve pločice sa kojima igrač ima koliziju. Ta kolizija omogućava restrikciju igračeve kretnje kako bi ostao u dvorcu i ne odlutao van njega. Sloj je postavljen na razinu -20 kako bi stajao nad ostala dva sloja.

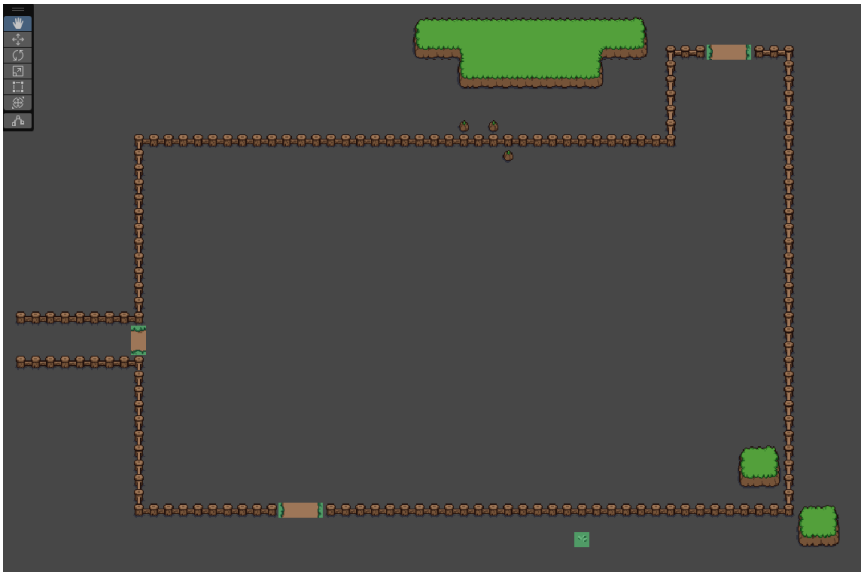


Slika 17 - GroundTop i CollisionObjects



Slika 18 - Bliži pogled područja za igru

Područje za igru omeđeno je sa ogradom unutar koje će se igrač moći kretati i boriti sa protivnicima.



Slika 19 - Kolizijski objekti na mapi područja za igru

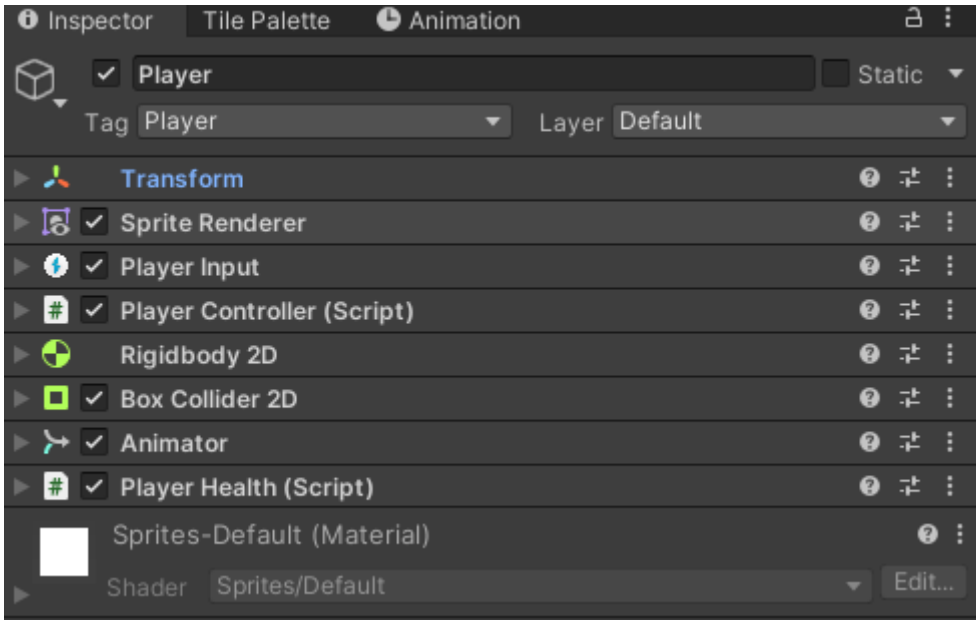
4.2. Dizajn korisničkog sučelja

Korisničko sučelje je točka interakcije između korisnika i programa nad kojem se izvode radnje. U ovome radu to su dvije glavne stavke. Jedna od njih je pravokutnik u gornjem lijevom kutku koji pokazuje koliko je života ostalo igraču, dok je druga broj preostalih neprijatelja koje igrač treba pobijediti.

Slika 20 - Korisničko sučelje igre

4.3. Razvoj lika

Proces razvoja kontrolera odnosno lika kojim će igrač tj. korisnik upravljati se može podijeliti u nekoliko dijelova koji će opisati načine i metode izrade likova te njihova implementacija u samu igru i skripte premoću kojih se upravlja sa likom. Skripte su pisane u *C#* programskom jeziku unutar programskog okruženja *Microsoft Visual Studio 2022.*



Slika 21 - Komponente lika

4.3.1. Dizajn lika

Poglavlje dizajniranja lika je vrlo bitan dio izrade videoigre jer je lik ono sa čime igrač odnosno korisnik upravlja te je jednako bitno kao i proces programiranja. Zbog tog razloga dizajn lika je preuzet sa interneta.

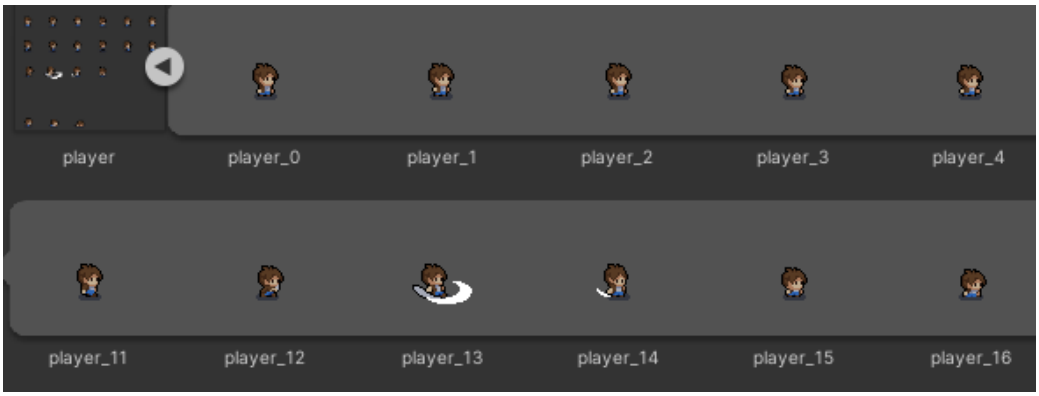
4.3.1.1. Spritesheet

Slika 21. prikazuje takozvani „Spritesheet“ tj. list sličica na kojemu je prikazan lik u različitim pozicijama dok hoda, stoji, trči, napada ili pada. Veličina lista sličica se može prilagoditi unutar inspektora da ne izgubi oštrinu i da bude bolje prikazano unutar same igre. Te sličice se dobivaju kroz prije spomenuti alat zvan „Sprite Editor“ unutar kojeg se postavlja „Spritesheet“ i po želji se ručno ili automatski mogu izrezati sličice na željenu veličinu kao što je vidljivo na slici 22.

Slika 22 - Spritesheet lika (preuzeto sa interneta)

Slika 23 - Prikaz izrezivanje sličica lika unutar Sprite Editora

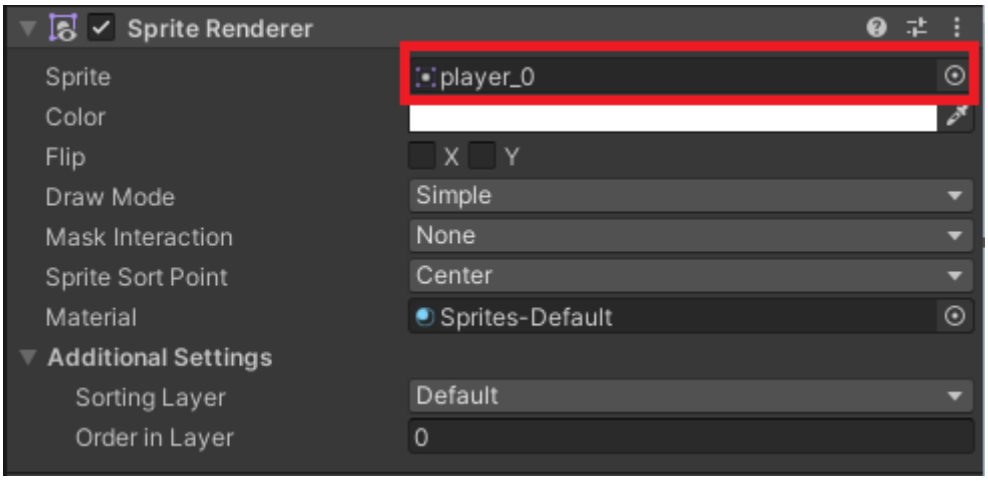
Nakon što se izrežu sličice *Unity* alat ih sprema ispod glavne slike tj. slike roditelja unutar datoteke projekta.



Slika 24 - Sličice unutar Unity-ja

4.3.1.2. Sprite renderer

Prilikom stvaranja lika potrebno je stvoriti prazan objekt unutar projetka i dodati mu komponentu „Sprite renderer“. Unutar „Sprite renderera“ je potrebno je povući jednu od sličica na mjesto gdje se nalazi „Sprite“. Pomoću te komponente lakše se određuje veličina i oblik lika te se automatski pridodavaju svojstva koja će biti spomenuta kasnije u radu.

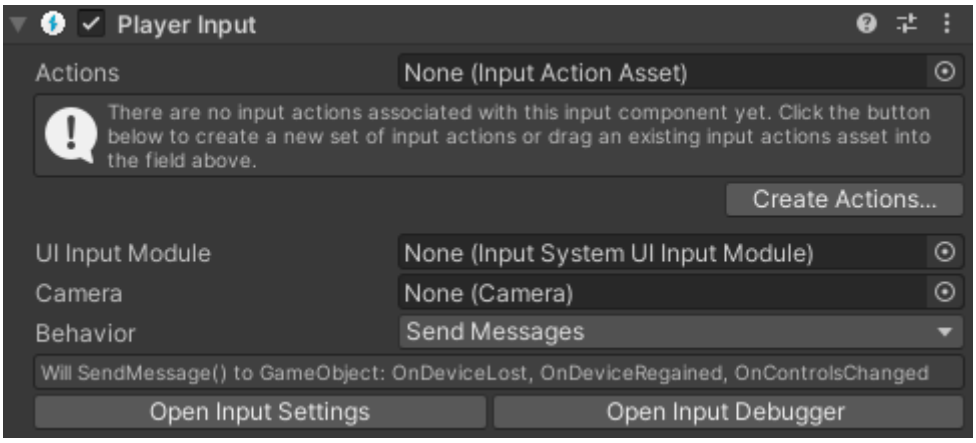


Slika 25 - Sprite renderer lika

4.3.1.3. Player Input

„Player Input“ komponenta koristi *MonoBehaviour* baznu klasu od koje sve preostale *Unity* skripte proizlaze kako bi se mogla obaviti interakcija između programera i alata *Unity.* „Player Input“ automatski upravlja omogućavanjem i onemogućavanjem radnji, a također upravlja instaliranjem povratnih poziva na radnje. Kada više „Player Input“ komponenti koristi iste radnje, komponente automatski stvaraju privatne kopije radnji.

Unutar ovog rada „Player Input“ je postavljen da korespondira sa akcijama pritiska gumbi na tipkovnici tako da se prilikom pritiska tipke „W“ lik kreće gore, „S“ dolje, „D“ desno i „A“ lijevo, a prilikom pritiska lijevog klika miša napada u okrenutom smjeru.

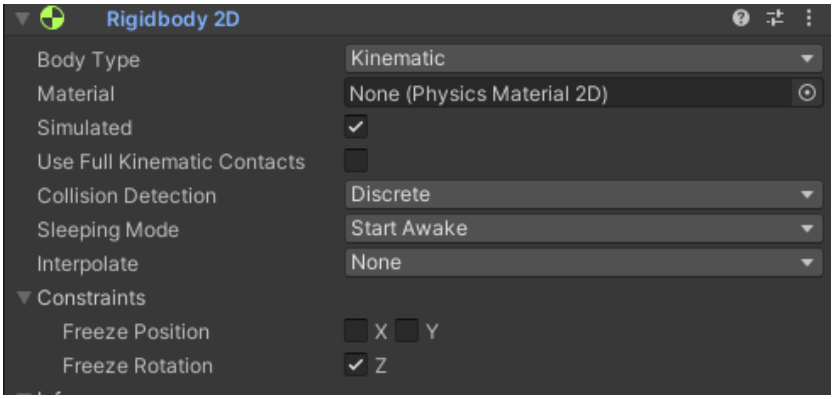


Slika 26 - Player Input lika

4.3.1.4. Rigidbody 2D

[5]„Rigidbody 2D“ je komponenta koja se pridodaje liku kako bi se nad njim vršile fizičke simulacije poput gravitacije i trenja te se kroz njega mogu lakše obrađivati sudari sa okolinom ukoliko i sudareni objekt isto sadrži odgovarajući „Rigidbody 2D“ komponentu.

Kako u ovom radu se lik ne može rotirati, potrebno je staviti kvačicu na „Freeze rotation“ ali ne i na poziciju X i Y jer one mogu biti promjenjive.

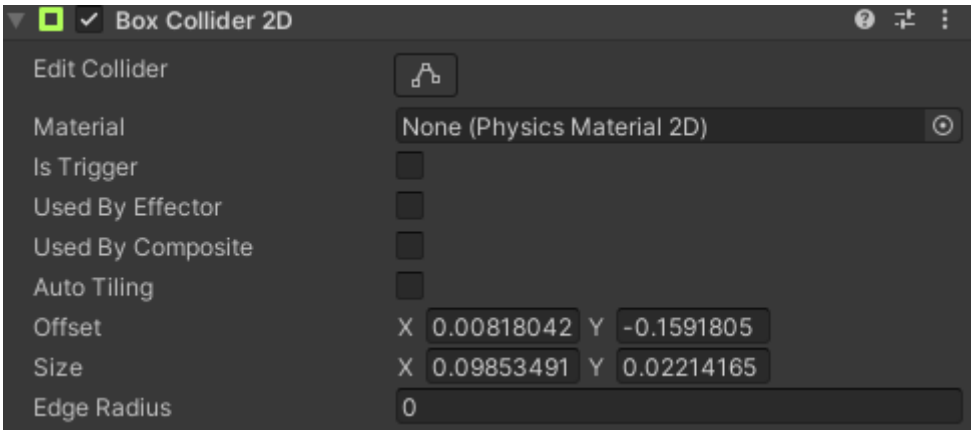


Slika 27 - Rigidbody 2D

4.3.1.5. Box Collider 2D

„Box Collider 2D“ je Collider koji je u interakciji s 2D sustavom fizike. Ima oblik pravokutnika s definiranim položajem, širinom i visinom u lokalnom koordinatnom prostoru Spritea. Sami pravokutnik je poravnat s osi, a rubovi su mu paralelni s X ili Y osi lokalnog prostora. Prostor je lako namjestljiv klikom na „Edit Collider“ gumb.

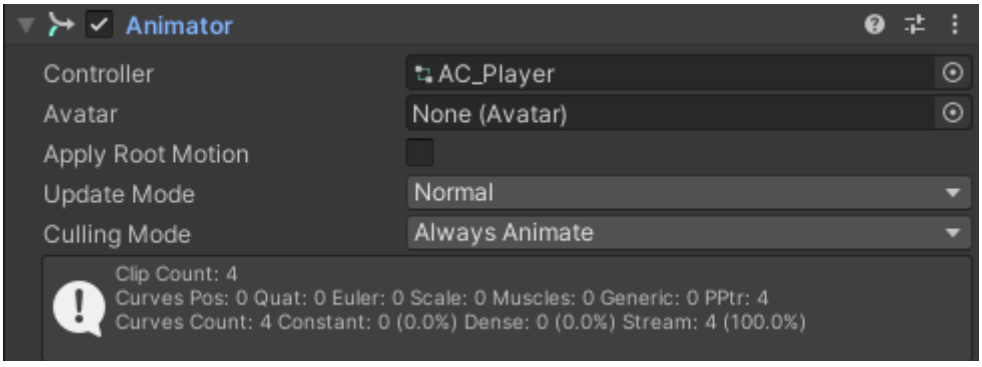
Slika 28 - Box Collider 2D lika (zeleni pravokutnik)



Slika 29 - Box Collider 2D inspektor

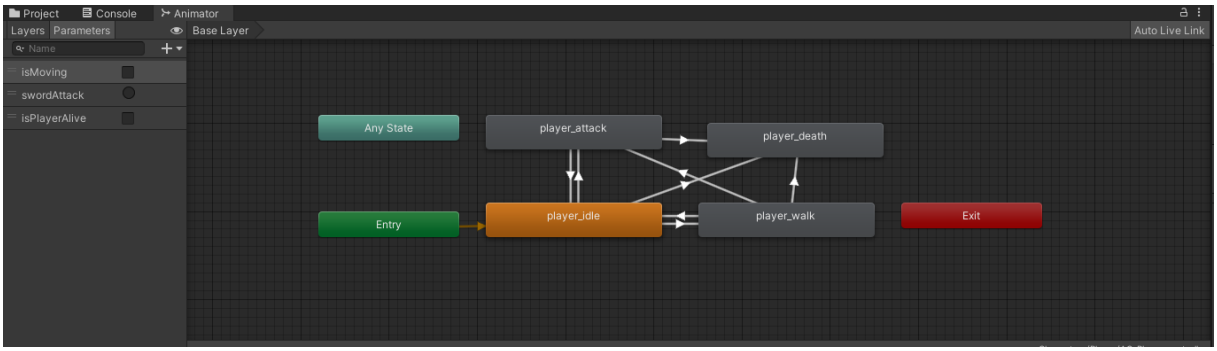
4.3.1.6. Animator

[6] Animator je klasa unutar alata *Unity* koja se pridodaje liku kao komponenta pomoću koje se upravljaju mehanizmi i događaji animacijskog sistema. Za potrebe animacije lika stvoren je kontroler „AC_Player“ kojim se upravlja animacijama lika i njihovim tranzicijama.



Slika 30 - Animator lika

Animacije unutar *Unity* alata rade na osnovi tranzicija, tj. kada se određeni događaj ili okidač pojave onda nastupa animacija. Ti događaji su bolje opisani unutar programskog kôda jer oni objašnjavaju kada nastupaju promjene i sa čime je koja animacija povezana. Primjeri animacija i njihovih tranzicija su vidljivi na slici 31.



Slika 31 - Animacije i tranzicije između animacija

[7] Svaka kućica sa riječi „player“ u sebi sadrži sličice animacije lika pomoću koje igrač može lakše dokučiti u kojem je stanju lik. Animacije međusobno povezuju preko linija koje se zovu tranzicije, a tranzicije se izvode prilikom okidanja nekog okidača, provjere broja ili stanja neke varijable. Te varijable su definirane unutar samog alata *Unity* i unutar skripti koje će se opisati u slijedećem poglavlju.

Kućice animacije su:

- „player_idle“ - animacija koja se aktivira pri pokretanju igre i kada je lik u stanju mirovanja tj. kada je *bool* varijabla „isMoving“ postavljena na neistinito
- „player_walk“ – animacija se pokreće u trenutku kada igrač pritisne tipke kretanja W, A, S ili D, a vraća se nazad u „player_idle“ kada kretnja staje
- „player_attack“ – animacija koja se pokreće u trenutku kada igrač pritisne tipku napada (lijevi klik miša), nakon toga se vraća u jednu od dvije animacije ovisno o tome da li se lik kreće ili ne
- „player_death“ – animacija koja se pokreće u trenutku kada igrač izgubi sve životne bodove

4.3.2. Skripte lika

Nakon što je razvijen lik, njegov dizajn, područja kolizije i animacije, potrebno ih je povezati unutar skripte. U ovome radu postoje dvije skripte za lika, jedna od njih se odnosi na životne bodove lika i prikaz njih na ekran preko objekta u igri zvanog „Health Bar“, dok je druga skripta odgovorna za kretanje, odnose sa kolizijom, brzinu lika, napad na neprijatelje, postavljanje *bool* vrijednosti za animacije unutar *Unityja* i prevrtanje „Sprite“-a lika.

Za početak je potrebno razjasniti četiri glavne funkcije *UnityEngine-a.* To su funkcije *Start(), Awake(), Update() i FixedUpdate().* Funkcija *Start()* se poziva na *frame-u* kada je skripta omogućena neposredno prije nego što se bilo koja metoda ažuriranja pozove po prvi put. Ona se poziva samo jedanput. Funkcija *Awake()* se poziva u onoj instanci kada skripta biva učitana i poziva se prije funkcije *Start().* Funkcija *Update()* se poziva na svakoj slici (eng.

Frame) dokle god se skripta izvršava te je odgovorna za logiku same igre poput provjere ako je igra završena, a *FixedUpdate()* je funkcija odgovorna za kretanje objekata u igri poput lika i neprijatelja jer se sama funkcija poziva 50 puta u sekundi te je preciznija za micanje objekata koje imaju svojstvo *Rigidbody.*

```
public float moveSpeed = 1f;
public float collisionOffset = 0.02f;
public ContactFilter2D movementFilter;
Vector2 movementInput;
SpriteRenderer spriteRenderer;
Rigidbody2D rb;
Animator animator;
bool canMove = true;
public SwordAttack swordAttack;
```

Slika 32 - PlayerController varijable

Unutar skripte lika nalazi se devet varijabli različitih tipova podataka. *moveSpeed, collisionOffset, movementFilter, movementInput* i *canMove* će biti korišteni za kretanje, dok će *spriteRenderer, rb i animator* biti odgovorni za animaciju lika, postavljanje okidača u *Unity-ju* i okretanje Sprite-ova te na kraju će varijabla *swordAttack* biti odgovorna za napadanje te će se preko nje pristupati klasi „SwordAttack“.

```
void OnMove(InputValue movementValue)
{
    movementInput = movementValue.Get<Vector2>();
}
```

Slika 33 - Funkcija dohvaćanja kretnje

Unutar funkcije *OnMove()* na slici 33. koja prihvaća unošene vrijednosti koje korisnik pritišće na tipkovnici pretvara u Vector2 koji se kasnije koristi za kalkulacije kretanja lika.

```
private bool TryMove(Vector2 direction)
{

    int count = rb.Cast(direction, movementFilter, castCollisions, moveSpeed * Time.fixedDeltaTime + collisionOffset);

    if (count == 0)
    {
        rb.MovePosition(rb.position + direction  * moveSpeed * Time.fixedDeltaTime);
        return true;
    }
    else
    {
        return false;
    }
}
```

Slika 34 - Funkcija pokušaja kretanja (vlastita izrada)

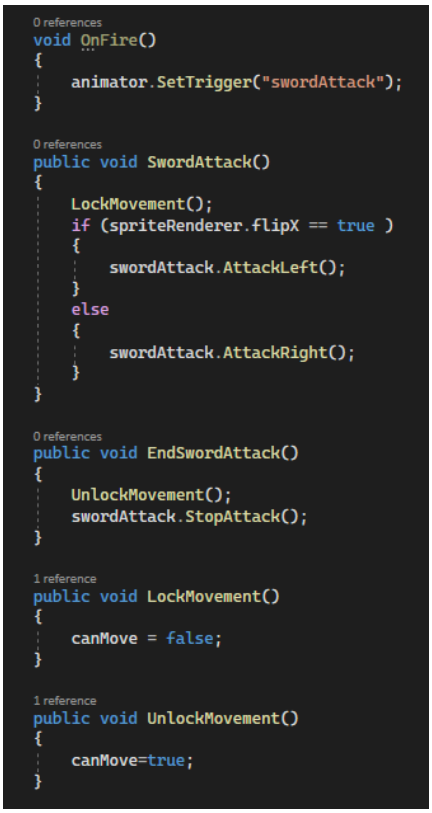
Kretanje se izvršava pomoću funkcije MovePosition unutar koje se tijelo lika pomiče sa *RigidBody* elementom *rb* na poziciju kojom se korisnik kreće koristeći *direction*. *Movespeed* određuje brzinu kretanja lika i na poslijetku Time.fixedDeltaTime je interval u sekundama u kojem se izvode ažuriranja fizike i drugih fiksnih brzina sličica u sekundi.

```
private void FixedUpdate()
{
    if (canMove)
    {
        //if movement is not 0, try to move
        if (movementInput != Vector2.zero)
        {
            bool success = TryMove(movementInput);
            if (!success && movementInput.x > 0)
            {
                success = TryMove(new Vector2(movementInput.x, 0));
            }
            if (!success && movementInput.y > 0)
            {
                success = TryMove(new Vector2(0, movementInput.y));
            }
            animator.SetBool("isMoving", success);

        }
        else
        {
            animator.SetBool("isMoving", false);
        }

        //set direction of sprite to movement direction
        if (movementInput.x < 0)
        {
            spriteRenderer.flipX = true;

        }
        else if (movementInput.x > 0)
        {
            spriteRenderer.flipX = false;
        }
    }
}
```

Slika 35 - FixedUpdate() funkcija kretanja lika (vlastita izrada)

Kretanje lika započinje provjerom varijable *canMove* koja je na početku igre postavljena na istinito te se provjera tokom cijelog tijeka igre. Ukoliko korisnik nije unio nikakvu naredbu kretnje postavlja se *bool* varijabla „isMoving“ preko animatora na neistinito koja će pokrenuti animaciju stajanja. Proces provjere ukoliko se lik može kretati se odvija unutar funkcije *TryMove* koji prima parametar *movementInput* i preko njega provjerava ukoliko se nalazi neki objekt sa kolizijom u onom trenutku kada smjer kretanja bude određen. Ukoliko ne postoji tada vraća istinito i liku je dopušteno kretanje u željenom smjeru i „isMoving“ se postavlja na istinito, a ako postoji tada je brzina kretanja lika 0 i ne kreće se. Zatim slijedi provjera ako igrač unosi dvije nesuprotne komande preko tipkovnice poput gore(W) i desno(D) te ako na jednoj se ne može kretati zbog kolizijskog objekta tada mu je dopušteno kretanje u onom smjeru gdje ne postoji koliziski objekt. To omogućava liku da može takoreći „kliziti“ po zidovima.

Ujedno time, unutar kôda je postavljena naredba na *spriteRenderer* sa kojime se upravlja samom slikom lika tj. kada se lik kreće u smjeru pozitivnom sa X osi (na desno) tada je slika i njegove animacije su okrenute prema desno, a kada se kreće u negativnom smjeru x osi (na lijevo) tada je njegova slika i sve animacije su zrcaljene i okrenute prema lijevo.



```
0 references
void OnFire()
{
    animator.SetTrigger("swordAttack");
}

0 references
public void SwordAttack()
{
    LockMovement();
    if (spriteRenderer.flipX == true )
    {
        swordAttack.AttackLeft();
    }
    else
    {
        swordAttack.AttackRight();
    }
}

0 references
public void EndSwordAttack()
{
    UnlockMovement();
    swordAttack.StopAttack();
}

1 reference
public void LockMovement()
{
    canMove = false;
}

1 reference
public void UnlockMovement()
{
    canMove=true;
}
```

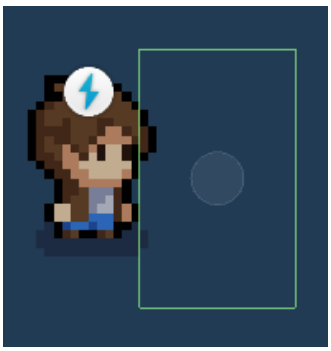
Slika 36 - Kôd lika napada

Unutar skripte lika se nalazi kôd koji upravlja sa oružjem kojeg koristi lik. *OnFire()* funkcija je funkcija klase *PlayerInput* i poziva se prilikom svakog klika miša. Na svaki klik se okida okidač „swordAttack“ kojom se poziva animacija lika da obavi animaciju napada u onom smjeru prema kojem je okrenut. Kako oružje ima svoju poziciju određenu u sceni potrebno je prikazati premještanje na stranu prema kojoj je okrenut lik. Funkcije *LockMovement()* i *UnlockMovement()* se koriste kako se igrač ne bi kretao tokom napadanja neprijatelja te one bivaju pozivane unutar alata *Unity* tokom izvođenja animacije napada tj. na početku napada se kretnja zaustavlja sa postavljanjem varijable *canMove* na neistinito, a na kraju napada se kretnja oslobađa sa postavljanjem varijable *canMove* na istinito.

4.3.3. Dizajn i skripte oružja lika

Dizajn oružja lika se odnosi na ono što lik koristi kako bi pobijedio neprijatelje koji ga dolaze napasti. Oružje ima svoju veličinu, količinu štete koju može nanijeti i ovisi o tome u kojem je smjeru lik okrenut. Oružje se unutar alata *Unity* sastoji od dvije komponente: „*Box Collider 2D“* i skripte. Kolizijska zona oružja je postavljena da bude malo veća od lika kako bi se mogli doseći neprijatelji koji se nalaze neposredno ispod i iznad lika. Međutim, kako je kolizijska zona potrebna da napravi štetu neprijatelju kada se dogodi kolizija između oružja i neprijatelja, kolizijska zona oružja je postavljena da bude se pojavi na okidač tj. kada se pritisne

lijevi klik miša. Samo oružje je smješteno unutar objekta lika kako bi se funkcije praćenja pozicije mogle lakše programirati.



Slika 37 - Kolizijska zona oružja

Unutar skripte *Enemy* nalaze se 3 varijable kojima je definirano područje oružja, njegova pozicija i količina štete koju nanosi igraču. Sve navedene varijable su prikazane na slici 38.

```
public Collider2D swordCollider;

public float damage = 1f;

Vector2 rightAttackOffset;
```

Slika 38 - Varijable unutar skripte oružja

Kada se igra učita, potrebno je namjestiti poziciju oružja gdje se ono nalazi, a kako je poželjno da oružje prati lika potrebno je navesti to u programu kada se učitaju podaci unutar funkcije *Start().*

```
Unity Message | 0 references
private void Start()
{
    rightAttackOffset = transform.position;
}
```

Slika 39 - Inicijalizacija oružja u kôd u

```
1 reference
public void AttackRight()
{

    swordCollider.enabled = true;
    transform.localPosition = rightAttackOffset;
}
1 reference
public void AttackLeft()
{

    swordCollider.enabled = true;
    transform.localPosition = new Vector3(rightAttackOffset.x * - 1, rightAttackOffset.y);
}
1 reference
public void StopAttack()
{
    swordCollider.enabled = false;
}
```

Slika 40 - Funkcije napada u različitim smjerovima

Kako se lik okreće u igri potrebno je okretati i oružje tokom napada, stoga se funkcije *AttackRight()* i *AttackLeft()* pozivaju unutar skripte igrača „playerController“. Kada je lik okrenut prema pozitivnoj vrijednosti osi X odnosno desno, tada se pojavljuje kolizijska zona oružja tokom izvođenja animacije napada, a na kraju izvođenja animacije ponovno nestaje, međutim kada je lik okrenut prema negativnoj vrijednosti osi X tada je okrenuta i pozicija kolizijske zone na lijevo pomoću postavljanja pozicije na negativnu vrijednost od početne tako da se množi trenutna pozicija od igrača na osi X sa vrijednosti od -1.

```
Unity Message | 0 references
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Enemy")
    {
        Enemy enemy = other.GetComponent<Enemy>();

        if (enemy != null)
        {
            enemy.Health -= damage;

        }

    }
}
```

Slika 41 - Funkcija opisa kolizije skripte oružja

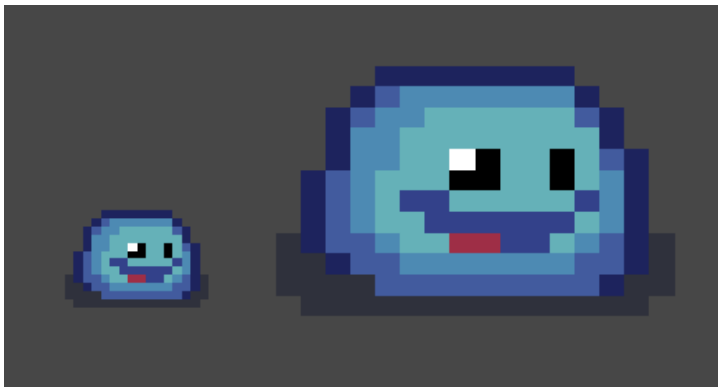
Funkcija na slici 41. opisuje što se događa kada kolizijska zona biva u kontaktu sa drugim kolizijskim objektom kojemu je *tag* odnosno oznaka jednaka „Enemy“. Ukoliko je oznaka neprijatelja prepoznata tada se neprijatelju smanjuju životni bodovi jednako šteti oružja ukoliko neprijatelj nije već izgubio sve životne bodove i bio uništen.

4.3.4. Dizajn i skripte neprijatelja

Neprijatelji unutar videoigre su oni objekti igre koji su isprogramirani da liku načine štetu koristeći skup pravila i ponašanja koji su definirani u skripti samog neprijatelja. Ti objekti tjeraju igrača da donosi odluke te same po sebi oblikuju iskustvo kako igrač igra igru. Postoje dvije

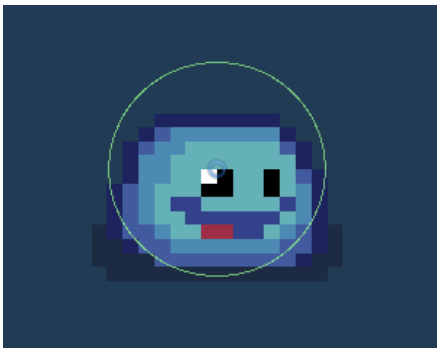
vrste tehnika kojima se stvara neprijateljska umjetna inteligencija, a to su: deterministička i nedeterministička tehnika. Determinističke tehnike su one tehnike u programiranju umjetne inteligencije kojima su akcije neprijatelja specifične i lako predvidljive igraču. Relativno su brzo implementirane te se mogu lakše razumjeti, testirati, i popraviti greške ukoliko postoje. Za razliku od determinističkih postoje i nedeterminističke tehnike programiranja umjetne inteligencije kojima je ponašanje nepredvidljivo i teško dokučivo igraču. U ovome radu korištena je deterministička tehnika dok se stvarala umjetna inteligencija.

U ovome radu postoje dvije vrste neprijatelja: „Slime“ i „BigSlime“. „Slime“ je neprijatelj koji ima malo životnih bodova ali je brz i lakše dolazi do igrača te radi malo štete igraču, dok je „BigSlime“ neprijatelj koji je spor, ali to nadoknađuje sa puno životnih bodova i velikom količinom štete koje može nanijeti igraču.



Slika 42 - Neprijatelji "Slime" i "Big Slime"

Svojstva neprijatelja su slična kao i kod svojstva lika kojem igrač upravlja. Neprijatelj ima vlastiti *RigidBody* koji mu omogućava kretnju uz pomoć skripte, ima vlastitog animator kontrolera koji se brine da se reproducira ona animacija koja je potrebna u trenutku njenog poziva te ima svoju vlastitu kolizijsku zonu. Međutim u slučaju neprijatelja kolizijska zona je ovalna jer se koristilo svojstvo *Capsule Collider 2D* umjesto *Box Collider 2D.*

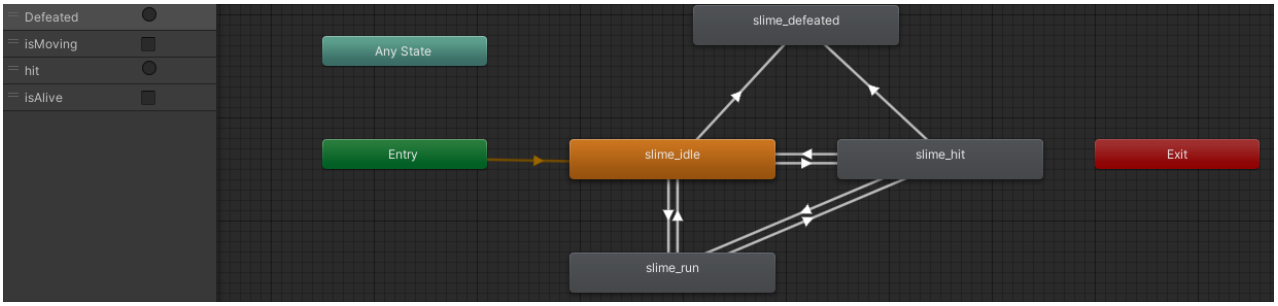


Slika 43 - Kolizijska zona neprijatelja

Napravljene su četiri animacije neprijatelja koje se reproduciraju tokom izvođenja igre prateći varijable postavljene u alatu *Unity* koje se kasnije upotrebljavaju unutar skripte.

One su:

- Slime_idle – animacija koja se reproducira dok neprijatelj stoji na mjestu
- Slime_run – animacija koja se reproducira dok se neprijatelj kreće
- Slime_hit – animacija koja se reproducira kada neprijatelj primi štetu
- Slime_defeated – animacija koja se reproducira kada neprijatelj biva pobijeđen



Slika 44 - Animator pogled neprijatelja

4.3.4.1. Skripta neprijatelja

Skripta započinje inicijalizacijom varijabli na početku skripte. Neke od vrijednosti su prije spomenute u radu pri opisu skripte lika poput *Animator, RigidBody2D, damage, moveSpeed, moveDirection, SpriteRenderer* ali i sama klasa lika kojim se računaju životni bodovi koje igrač ima *playerHealth* te i varijabla *Target* pomoću koje se pronalazi igrač.

```
Animator animator;
public Rigidbody2D rb;
public int damage;
public playerHealth phealth;
SpriteRenderer spriteRenderer;

public float maxHealthEnemy = 1f;
public float health = 1f;
bool isMoving = false;

bool isAlive = true;
[SerializeField] public float movespeed = 5f;
Transform target;
Vector2 moveDirection;
```

Slika 45 - Varijable objekta igre "Enemy"

Varijabla tipa float *maxHealthEnemy* se koristi kako bi se odredili maksimalni životni bodovi neprijatelja a *health* se koristi kako bi se provjerili trenutni životni bodovi. Varijable tipa bool *isMoving* i *isAlive* se koriste kako bi se unutar alata *Unity* pokrenule određene animacije.

```
Unity Message | 0 references
public void Start()
{

    animator = GetComponent<Animator>();
    animator.SetBool("isAlive", isAlive);
    health = maxHealthEnemy;
    spriteRenderer = GetComponent<SpriteRenderer>();

    target = GameObject.Find("Player").transform;
}
```

Slika 46 - Start() funkcija unutar skripte neprijatelja

Kada se igra pokrene potrebno je pridružiti vrijednosti prije navedenim varijablama na odgovarajuće vrijednosti.

```
Unity Message | 0 references
private void Update()
{
    if (target)
    {
        Vector3 direction = (target.position - transform.position).normalized;
        moveDirection = direction;
        if(moveDirection.x < 0)
        {
            spriteRenderer.flipX = true;
        }
        else if(moveDirection.x > 0)
        {
            spriteRenderer.flipX = false;
        }
        if(movespeed == 0)
        {
            isMoving = false;
            animator.SetBool("isMoving", isMoving);
        }
        else
        {
            isMoving = true;
            animator.SetBool("isMoving", isMoving);
        }
    }
}
```

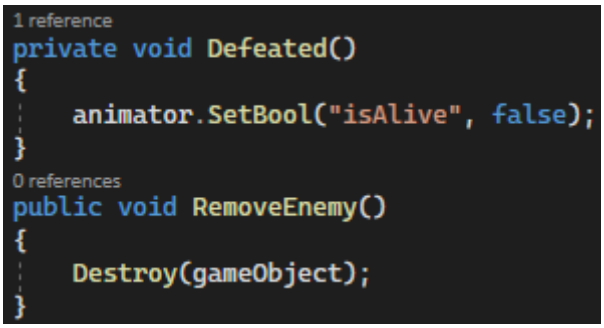
Slika 47 - Update funkcija neprijatelja

Unutar funkcije *Update()* se obavlja praćenje kretnji lika koristeći *target* koji je za vrijeme početka igre locirao poziciju lika te ukoliko taj objekt postoji će se odrediti smjer kretanja neprijatelja. Ujedno će se provjeriti orijentacija u kojem se smjeru kreće neprijatelj kako bi se slika neprijatelja okrenula pravilno. Ujedno postoje i provjere ukoliko se neprijatelj kreće i postavljaju se vrijednosti unutar animatora u skladu s time.

```
Unity Message | 0 references
private void FixedUpdate()
{
    if (target)
    {
        rb.velocity = new Vector2(moveDirection.x, moveDirection.y) * movespeed;
    }
}
```

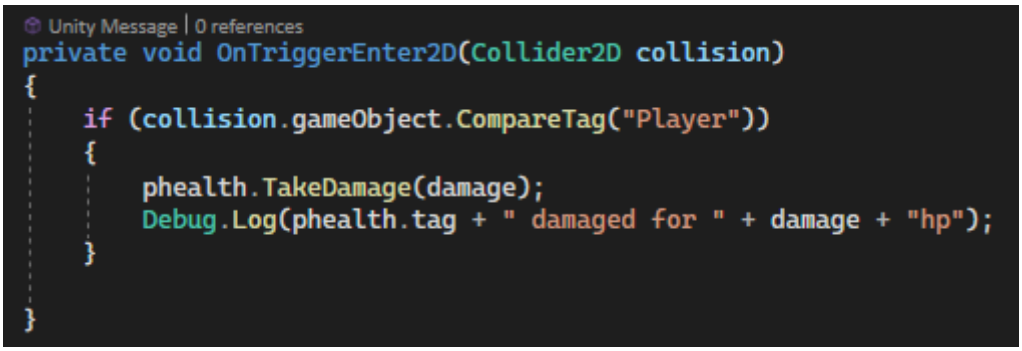
Slika 48 – FixedUpdate() funkcija neprijatelja

Unutar *FixedUpdate()* funkcije neprijatelja se nalazi jednostavna kalkulacija kretanja u smjeru u kojem se nalazi igrač.



Slika 49 - Poraz i uklanjanje neprijatelja

Funkcije *Defeated()* i *RemoveEnemy()* se pozivaju u onome trenutku kada neprijateljevi životni bodovi dosegnu vrijednost jednakoj ili niže od 0. Kada se dosegne ta vrijednost prvo se reproducira animacija poraza u animatoru tako da se namjesti vrijednost „isAlive“ na neistinito te nakon što se animacija završi objekt uklanja kako ne bi zauzimao prostor.



Slika 50 - Funkcija nanošenja štete liku

Funkcija kojom se šteti igraču *OnTriggerEnter2D()* jest korištena kako bi se igraču životni bodovi umanjili. Funkcija provjerava ukoliko je kolizijska zona igrača unutar zone neprijatelja te ukoliko jest tada se oduzimaju životni bodovi.

5. Zaključak

Predmet ovog rada bila je izrada 2D videoigre obrane dvorca u kojoj se igrač nalazi unutar mjesta iz kojeg ne može izaći te se mora braniti od neprijatelja koji ulaze u taj prostor. Međutim, osim same realizacije rada cilj je ujedno bio i pokazati proces same izrade i objasniti dijelove stvorene igre. Platforma koja se koristila za kreiranje igre jest alat *Unity* zajedno sa svim značajkama koje alat nudi. Za kreiranje skripti se koristio programski jezik *C#* unutar programskog okruženja *Microsoft Visual Studio 2022.*

U radu su detaljno objašnjeni alati i komponente koje su bile korištene za realizaciju igre. Prilikom razvoja bilo je potrebno proučiti *Unity* priručnik koji objašnjava alate, skripte komponente i *AssetStore* sa kojeg je bilo lako preuzeti podatke potrebne za stvaranje igre. Osim toga bilo je važno proučiti puno video uradaka koji objašnjavaju rad unutar samog alata i koji objašnjavaju greške koje su se pojavljivale za vrijeme izrade rada. Postupak izrade lika, mape, neprijatelja i same igre u cjelini je pokazao kako je pravljenje igre postupak koji mora zadovoljiti određene segmente kako bi se igra napravila boljom.

6. Popis literature

[1] https://www.create-learn.us/blog/top-games-made-with-unity/ dostupno 3.9.2022.

[2] A History of the Unity Game Engine, An Interactive Qualifying Project, John Haas, 2014.

[3] https://docs.unity3d.com/Manual/UsingTheEditor.html dostupno 25.8.2022.

[4] https://docs.unity3d.com/Manual/class-Tilemap.html dostupno 25.8.2022.

[5] https://docs.unity3d.com/Manual/class-Rigidbody2D.html dostupno 25.8.2022.

[6] https://docs.unity3d.com/Manual/class-Animator.html dostupno 25.8.2022.

[7] https://docs.unity3d.com/Manual/class-AnimatorController.html

dostupno 25.8.2022.

7. Popis slika

Slika 1 - Korisničko sučelje programa Unity 5
Slika 2 - Traka sa alatima 5
Slika 3 - Hijerarhijski prozor 6
Slika 4 - Prikaz scene (eng. Scene view) 7
Slika 5 - Prikaz igre (eng. Game view) 8
Slika 6. - Inspekcijski prozor (eng. Inspector window) 8
Slika 7 - Prozor projekta (eng. Project window) 9
Slika 8 - Prozor konzole (Console window) 10
Slika 9 - Mapa igre (vlastita izrada) 11
Slika 10 - Mapa igre u hijerarhiji projekta 12
Slika 11 - Sprite Editor 12
Slika 12 - Sprite Editor, izrezane sličice 13
Slika 13 - Pločice igre(tiles) unutar palete pločica 13
Slika 14 - Ground sloj mape 13
Slika 15 – Skup sličica "Ground" dijela mape 14
Slika 16 - "GroundTop" dio mape 14
Slika 17 - GroundTop i CollisionObjects 14
Slika 18 - Bliži pogled igrivog područja 15
Slika 19 - Kolizijski objekti na mapi igrivog područja 15
Slika 20 - Korisničko sučelje igre 16
Slika 21 - Komponente lika 16
Slika 22 - Spritesheet lika (preuzeto sa interneta) 17
Slika 23 - Prikaz izrezivanje sličica lika unutar Sprite Editora 17
Slika 24 - Sličice unutar Unity-ja 18
Slika 25 - Sprite renderer lika 18
Slika 26 - Player Input lika 19

Slika 27 - Rigidbody 2D 19
Slika 28 - Box Collider 2D lika (zeleni pravokutnik) 20
Slika 29 - Box Collider 2D inspektor 20
Slika 30 - Animator lika 20
Slika 31 - Animacije i tranzicije između animacija 21
Slika 32 - PlayerController varijable 22
Slika 33 - Funkcija dohvaćanja kretnje 22
Slika 34 - Funkcija pokušaja kretanja (vlastita izrada) 22
Slika 35 - FixedUpdate() funkcija kretanja lika (vlastita izrada) 23
Slika 36 - Kôd lika napada 24
Slika 37 - Kolizijska zona oružja 25
Slika 38 - Varijable unutar skripte oružja 25
Slika 39 - Inicijalizacija oružja u kôd u 25
Slika 40 - Funkcije napada u različitim smjerovima 26
Slika 41 - Funkcija opisa kolizije skripte oružja 26
Slika 42 - Neprijatelji "Slime" i "Big Slime" 27
Slika 43 - Kolizijska zona neprijatelja 27
Slika 44 - Animator pogled neprijatelja 28
Slika 45 - Varijable objekta igre "Enemy" 28
Slika 46 - Start() funkcija unutar skripte neprijatelja 29
Slika 47 - Update funkcija neprijatelja 29
Slika 48 – FixedUpdate() funkcija neprijatelja 29
Slika 49 - Poraz i uklanjanje neprijatelja 30
Slika 50 - Funkcija nanošenja štete liku 30