Izrada 2D igre za mobilne uređaje korištenjem UNITY razvojnog okruženja

Prce, Nevenko

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Dubrovnik / Sveučilište u Dubrovniku**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:155:858890

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-20**

Repository / Repozitorij:

Repository of the University of Dubrovnik

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNRSTVO

NEVENKO PRCE

IZRADA 2D IGRE ZA MOBILNE UREĐAJE KORIŠTENJEM UNITY RAZVOJNOG OKRUŽENJA

ZAVRŠNI RAD

Dubrovnik, rujan, 2020.

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNRSTVO

IZRADA 2D IGRE ZA MOBILNE UREĐAJE KORIŠTENJEM UNITY RAZVOJNOG OKRUŽENJA

ZAVRŠNI RAD

Studij: Primijenjeno/Poslovno Računarstvo

Kolegij: Razvoj mobilnih aplikacija

Mentor: doc. dr. sc. Krunoslav Žubrinić

Student: Nevenko Prce JMBAG: 0275037504

Dubrovnik, rujan, 2020.

SAŽETAK

Ovaj rad opisuje problematiku izrade video igara pomoću alata *Unity*. Rad započinje kratkim uvodom nakon kojega slijedi teorijski dio u kojem se objašnjava problematika rada. Opisuje se povijest mobilnih igara i alati koji se koriste u empirijskom dijelu rada. U empirijskom dijelu se opisuje postupak izrade igre koristeći alat *Unity*. Opisuje se izrada *sprajtova*, animacija i skripti, izrada mapa te postupak obrade kolizije.

Ključne riječi: razvoj računalnih igara, *Unity*, računalne igre

ABSTRACT

This paper describes the problem of making video games using the *Unity* tool. The paper begins with a short introduction followed by a theoretical part in which the problems of the work are explained. The history of mobile games and the tools used in the empirical part of the paper are described. The empirical part describes the process of making a game using the *Unity* tool. It describes the creation of sprites, animations and scripts, the creation of maps and the process of collision processing.

Keywords: development of computer games, *Unity*, computer games

Sadržaj

SAŽETAK I

ABSTRACT I

1 UVOD 1

2 TEORIJSKI DIO RADA 2

2.1 Mobilne igre 2

2.1.1 Industrijski standard 4

2.1.2 Uobičajena ograničenja mobilnih igara 4

2.2 *Unity* 5

2.2.1 Podržane platforme u *Unityju* 5

2.2.2 *Unity Asset Stor*e 6

2.3 Tipovi mobilnih igara 6

2.3.1 Akcijske igre 7

2.3.2 Ležerne (*Casual*) igre 7

2.3.3 Puzzle igre 7

3 EMPIRIJSKI DIO RADA 8

3.1 Opis igre 8

3.2 Sprajtovi, animacije i skripte u *Unityju* 9

3.2.1 Sprite editor 9

3.3 Glavni lik 12

3.3.1 Animacije 12

3.4 *Rigidbody* 2D 15

3.5 *Box Collider* 2D 16

3.6 Kretanje glavnog lika 17

3.7 Neprijatelji 19

3.7.1 Animacije 20

3.7.2 Kretanje neprijatelja 21

3.8 Izrada mape igre 25

3.8.1 Tilemap u Unityju 26

3.8.2 Tile Palette 27

3.9 Korištenje kolizija za prelazak iz jednog u drugi dio mape 28

3.10 2D svjetlo u *Unityju* 29

4 ZAKLJUČAK 31

5 LITERATURA 32
6 PRILOZI 33
6.1 Popis slika 33

1 UVOD

Mobilne aplikacije su aplikacije programske podrške za pametne telefone, tablet računala i druge mobilne uređaje. Prvobitno su služile za brzu provjeru električne pošte i sl., ali njihova velika potražnja dovela je do proširenja i na druga područja kao što su navigacijski uređaji, igre za mobitele, gledanje video sadržaja ili pretraživanje interneta.

Teorijski dio rada opisuje mobilne igre, njihovu razradu, tipove računalnih igara koja sam istraživao kao i alata koje sam koristio.

U empirijskom dijelu rada ulazim dublje u sam postupak izrade mobilne igre koristeći alat *Unity*. [1] *Unity* je danas jako zastupljen alat korišten u razvoju različitih vrsti igara. Alat je popularan jer je lako dostupan, a omogućuje izradu 2D i 3D igara na različitim platformama. Danas je posebno popularan za razvoj mobilnih igara na različitim operacijskim sustavima posebno *Androidu* i *Apple* iOS.

Nakon uvoda, u teorijskom dijelu rada se kratko opisuje povijest razvoja računalnih igara s naglaskom na mobilne igre te *Unity* alat za razvoj igara. U empirijskom dijelu rada opisan je postupak izrade računalne igre koristeći *Unity* alat. Opisan je postupak izrade grafičkih elemenata igre i glavni dijelovi programskog koda uključujući korištenje 2D osvjetljenja lika. U zaključnom dijelu rada navedeni su glavni naglasci iz rada te je dan osvrt na napravljenu aplikaciju.

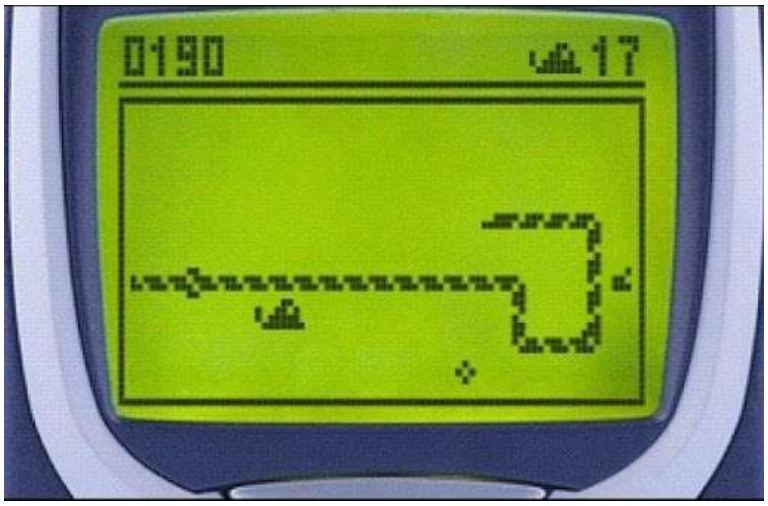
2 TEORIJSKI DIO RADA

U teorijskom dijelu rada opisat ću kratku povijest razvoja računalnih igara s naglaskom na igre na mobilnim uređajima te objasniti zbog čega su mobilne igre tako uspješne. i Objasnit ću zbog čega sam koristio *Unity* kao glavni alat za izradu igre.

2.1 Mobilne igre

Mobilne igre su računalne igre koje se igraju na mobilnim telefonima, tabletima i drugim prenosivim uređajima.

Sama povijest mobilnih igara seže od igre *Tetris* iz 1994. godine koja je bila dostupna na uređaju *Hagenuk* MT-2000 [1]. 1997. godine *Nokia* objavljuje igru *Snake* koja je bila instalirana na većini mobilnih uređaja koje je izrađivala *Nokia*. Posljedica toga je bila da je ta igra bila jedna od najviše igranih igara na mobilnim uređajima u to vrijeme te se mogla naći na više od 350 milijuna mobilnih uređaja diljem svijeta [2]. Na slici 1 prikazan je izgled igra *Snake* na ekranu mobilnog telefona.



Slika 1 Nokia Snake

Danas, mobilne igre se u većini slučajeva preuzimaju sa *app storea* . Preuzimanje igara prvo se komercijaliziralo u Japanu izlaskom NTT DoCoMo I-mode platforme 1999. godine, i do početka 2000-ih godina bilo je dostupno u većini svijeta. Sve do izlaska iOS *App Storea* 2008 sve je to bilo prilično marginalno. Kako prvo mjesto gdje su se mogle preuzimati aplikacije, *App Store* je totalno promijenio ponašanje korisnika i jako brzo otvorio novo tržište za mobilne igre, sa tim da je skoro svaki korisnik iOS platforme počeo preuzimati mobilne aplikacije igre s *App Storea* [3]. Među ključnim hardverskim značajkama *iPhonea* bila je velika veličina memorije (engl. *Random Access Memory* - RAM) u usporedbi s većinom ostalih pametnih telefona na tržištu, kao i veći zaslon, što ga čini sposobnim za pokretanje složenijih aplikacija, te novi *iOS* operativni sustav koji imao mogućnost *multitaskinga*, daleko nadmašujući bilo koji drugi uređaj na tržištu u to vrijeme. i*Phone* je također sadržavao razne senzore poput akcelerometra, a isključujući prvu generaciju,

uključio je i kapacitivni zaslon osjetljiv na dodir za koji nije bio potreban nikakav poseban alata za korištenje i kojim se moglo upravljati prstom, a kasniji modeli dodali su podršku za višestruko očitavanje [5]. Uz izdanje, *Apple* je potvrdio da se aplikacije trećih strana mogu razvijati za iOS operativni sustav telefona i distribuirati putem *App Storea*, koji je korisnicima bio dostupan u srpnju 2008. *Apple* je odredio uvjete korištenja alata i *App Storea* jeftino i jednostavno za poticanje razvoja aplikacija trećih strana. U listopadu 2009. *App Store* trgovina je predstavila *"in-app purchases"* (*IAP*), mikrotransakcije koje aplikacija može ponuditi transakcijom izvršenom putem izloga *App Storea*. Neki postojeći programeri aplikacija jako brzo su to iskoristili. *Tapulous* je objavio muzičku igru *Tap Tap Revenge 3* ubrzo nakon ove promjene koja je uključivala IAP za dobivanje novih pjesama [6]. Sličan *IAP* je dodan u *Gogle Play Storeu* na *Androidu*.

U prosincu 2009. godine *Rovio Entertainment* objavio je *Angry Birds* na *App Storeu*, igri koja uključuje lansiranje crtanih ptica na strukture zauzete svinjama koje su ukrale njihova jaja kako bi nanijele što veću štetu. Igra je bila nadahnuta igrom *Crush the Castle* i drugim sličnim igrama [7]. Nakon što je *Rovio* prenio igru na *Android* platformu, uveli su verziju s podrškom za oglase koja se mogla preuzeti besplatno, ali korisnik je mogao platiti da se uklone oglasi. *Rovio* je prihodovao i od IAP-a i od oglasa u listopadu te je 2010. godine zarađivao oko 42 milijuna američkih dolara mjesečno. Druga igra *Cut the Rope*, objavljena istovremeno na *iOS-u* i *Androidu*, slijedila je model izdavanja besplatne verzije s nekoliko razina i kupnjom u igri za otključavanje ostatka igre. Bila je to jedna od najbrže prodavanih igara na *iOS App Storeu* u to vrijeme [8].

Dok su igre poput *Angry Birds* i *Cut the Rope* postizale uspjeh na mobilnim uređajima, razvoj društvenih mreža korištenjem naprednih web tehnologija na osobnim računalima, doveo je do izdanja niza besplatnih igara u web preglednicima. Jedan od najistaknutijih primjera toga je *Zynga's FarmVille*, objavljen 2009. godine. Radi se o simulacijskoj igri upravljanja farmama, ali igraču je u igri bio omogućen ograničen dnevni broj akcija. Kako bi povećali dozvoljeni broj akcija, igrači su mogli angažirati svoje prijatelje na *Facebooku* da dobiju dodatni broj akcija. Igra se smatrala vrlo uspješnom, s više od 80 milijuna igrača do veljače 2010. [9]

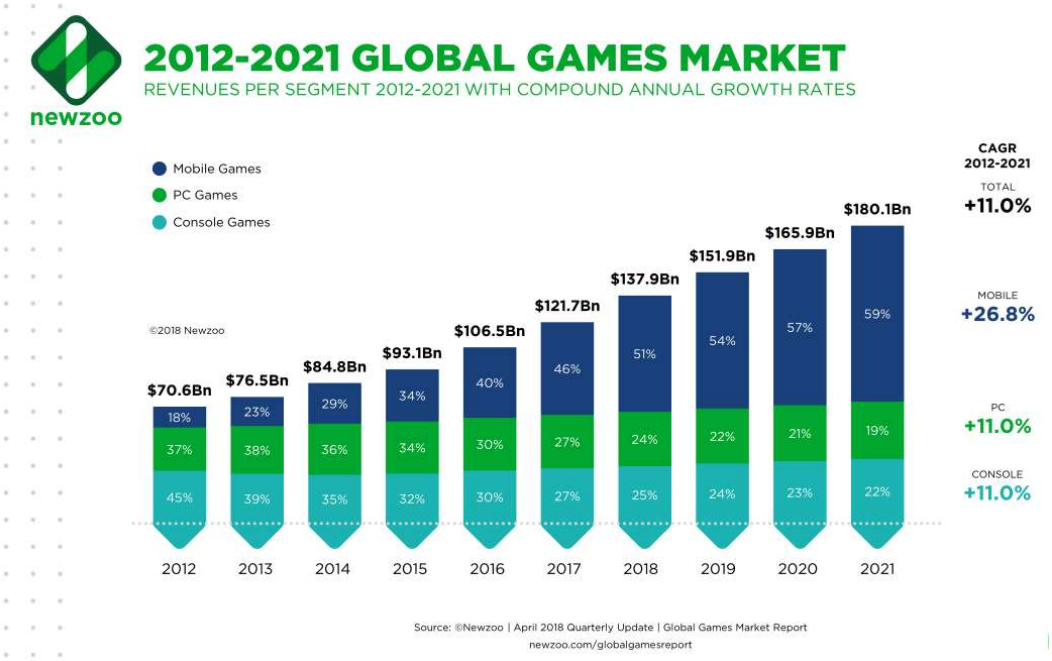
Supercell je objavio igru *Clash of Clans* 2012. *Clash of Clans* strateška je igra koja u svojoj osnovi ima elemente upravljanja gradom i obrane tornja dok igrač nadgleda bazu borbenog klana. Da bi dobio resurse za održavanje i nadogradnju baze, igrač može poslati svoje snage da napadnu bazu drugog igrača, čime se asinkrono rukuje snagama protivničkog igrača kojima upravlja računalo. Ako napadački igrač pobijedi, oni kradu neke resurse od igrača koji gubi, dok će igrač koji gubi, kada sljedeći put pristupi igri, saznati za te gubitke. Kako bi potaknuli suradnju, igrači se mogu pridružiti "klanovima" koji pomažu u automatskom napadu ili obrani. Do rujna 2014., aplikacija je zarađivala 5,15 milijuna američkih dolara dnevno, a mnogi su korisnici izjavili kako igraju igru tisućama sati od njenog lansiranja [10].

Pod licencom *The Pokémon Company* i *Nintenda*, *Niantic* je u srpnju 2016. objavio *Pokémon Go* kao besplatnu aplikaciju za mobilne telefone. Nakon što je već imao iskustva s razvojem igara temeljenih na lokaciji s prethodnim *Ingress* naslovom, *Niantic* je pomoću GPS-a telefona mapirao obližnja mjesta u blizini igrača gdje su mogli pronaći i pokušati uhvatiti *Pokémone* koje bi mogli koristiti u virtualnim lokalnim *Pokémon* dvoranama (koje

su također pozicionirane na određenim GPS lokacijama). U igri su *Pokémoni* prikazani igraču pomoću proširene stvarnosti korištenjem kamere, tako da igrač zna kada je pronašao *Pokémona* i kada ga je uhvatito. Kupnje putem aplikacije može se koristit za kupnju poboljšanih lopti koji se koriste za hvatanje *Pokémona* te drugih pojačanja i predmeta koji pomažu igračima pri hvatanju i treniranju *Pokemona*. *Pokémon Go* imao je rekordan broj igrača, s početnim izdanjima za *iOS* i *Android* preko 100 milijuna igrača širom svijeta u roku od mjesec dana od izlaska [11].

2.1.1 Industrijski standard

Mobilne platforme postale su najznačajniji segment industrije video igara jer pametni telefoni šire tržište igara više od konzola i osobnih računala. Mobilne igre su ostvarile 60% globalnog prihoda od video igara 2019. godine, sto je 49 milijardi dolara prihoda. Predviđa se da će se do 2024 godine tržište još povećati za 2,9% [4]. na slici 2 vidi se rast prihoda prodaje mobilnih igara od 2012. godine do predviđanja za 2021. godinu.



Slika 2 Prikaz rasta prodaje mobilnih igara tijekom godina [5]

2.1.2 Uobičajena ograničenja mobilnih igara

Mobilne igre obično su jednostavnije u odnosu na igre za računala i igraće konzole, a mnoge daju prednost inovativnom dizajnu i lakoći igranja nad spektakularnom grafikom. Ograničenja pohrane, memorije i ograničenja na veličinu datoteka izravno utječu na direktno prebacivanje igara namijenjenih osobnim računalima i konzolama na

mobilne platforme. Jedan od glavnih problema je kao utjecati tj. uvjeriti korisnika da je igra vrijedna kupnje zbog prethodno navedenih restrikcija kod razvoja.

2.2 *Unity*

Unity je alat za izradu računalnih igara koji podržava izradu igara na više platforma. Izradila ga je kompanije *Unity Technologies*, a prvi put je predstavljen 2005. godine. Do 2018, *Unity* ima podršku za više od 25 različitih platformi. *Unity* se može koristiti za izradu 2D, 3D igara s elementima virtualne stvarnosti, igara s elementima proširene stvarnosti, za izradu simulacija i drugo [5]. Sam *Unity* u novijim verzijama podržava i simuliranje različitih događaja pa je time postao i standard za izradu simulacija u područjima poput filmske industrije, auto industrija, arhitekture, inženjerstva i sl. Posljednja stabilna verzija za vrijeme pisanja ovog rada je *Unity* 2020.1.6.

Na slici 3 prikazan je izgled prozora unutar *Unity* okruženja.

Slika 3 Prikaz izgleda prozora alata Unity

U 2018. godini Unity je bio korišten za izradu otprilike pola novih mobilnih igara na tržištu, kao i 60% sadržaja sa virtualnom i proširenom stvarnosti [6].

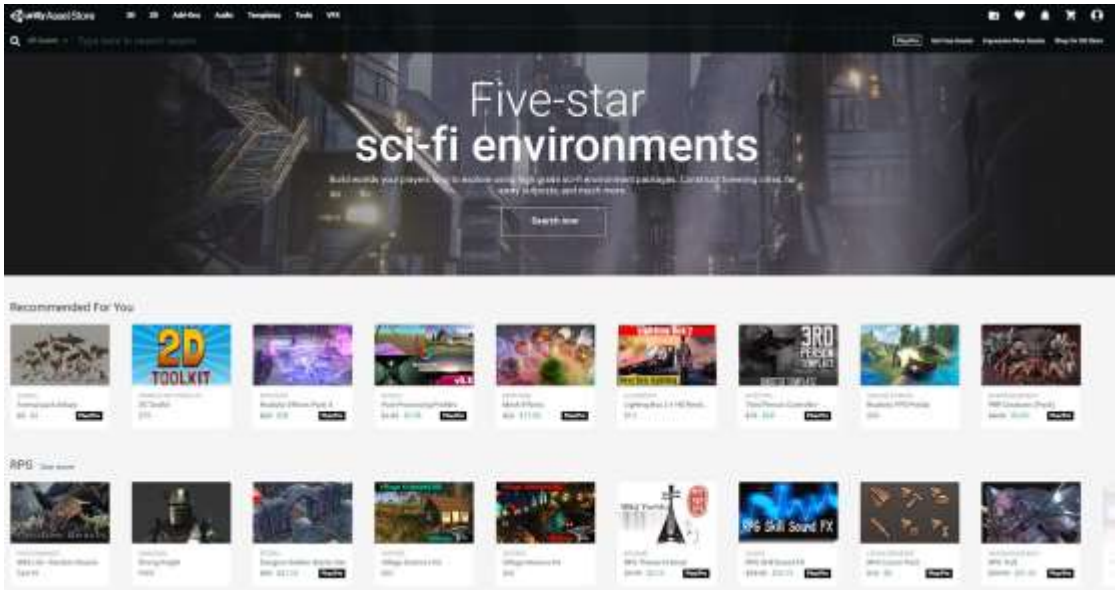
2.2.1 Podržane platforme u *Unityju*

Kao što sam prethodno spomenuto, *Unity* je alat koji podržava izradu aplikacija za više različitih platformi. Dostupan na *Windows* i *macOS* operacijskim sustavima, kao i na *Linuxu* s posebnom verzijom. Platforme za koje UNity može proizvesti izvršni kod igara

uključuju iOS, *Android*, *Tizen*, *Windows*, *Mac*, *Linux*, *WebGL*, razne verzije *PlayStationa*, *Xbox One*, 3DS, *Oculus Rift*, *Google Cardboard*, *Steam* VR, , *Windows Mixed Reality*, *Android* TV, *Samsung Smart* TV, tvOS, *Nintendo Switch*, *Apple ARKit*, *Vufioria* i druge.

2.2.2 *Unity Asset Stor*e

Korisnici *Unityja* mogu stvoriti i prodavati dijelove igara koje su oni napravili drugim korisnicima na *Unity Asset Storeu*. To uključuje 3D i 2D elemente koje korisnici mogu prodavati ili kupiti. *Unity Asset Store* je aktivan od 2010 godine i do 2018 godine je zabilježeno oko 40 milijuna preuzimanja sa digitalne prodavaonice [7]. Na slici 4 prikazan je izgled sučelja *Unity Asset Storea*.



Slika 4 Prikaz izgleda Unity Asset Store-a

2.3 Tipovi mobilnih igara

Tržište mobilnih igara ima puno različitih žanrova koji mogu ciljati bilo koji ukus. Uobičajena podjela mobilnih igara na kategorije je sljedeća:

- Akcijske,
- Avanturističke,
- Arkadne,
- *Battle Royale*,
- Ležerne (*Casual*),
- Kartaške,
- *Puzzle* ,
- Strategije

U nastavku poglavlja su detaljnije opisani neki od tipova igara koje su najčešće korišteni na mobilnih uređajima.

2.3.1 Akcijske igre

Iako su akcijske igre popularnije na osobnim računalima i konzolama, u mobilnom tržištu predstavljaju 11% tržišta. Akcijske igre je lakše doživjeti na većim ekranima i kontrole znaju biti preprecizne za korištenje kontrola dodira pa igrači akcijskih igara češće preferiraju druge načine igranja. [15]

2.3.2 Ležerne (*Casual*) igre

Ova kategorija igara dominira tržištem igara za mobilne uređaje. S 58% mobilnih uređaja, ležerne igre su žanr igara koje *Android* korisnici najviše igraju. Ovaj žanr uključuje sve, od igara poput *Candy Crush* do najpoznatijih igara na sreću u njihovim besplatnim verzijama, poput automata ili besplatnog bingo igara. Većina bingo i online casino platformi primijetili su ovaj trend i svi oni već imaju svoju aplikaciju kako bismo se mogli kladiti sa svojim pametnim telefonima. [15]

2.3.3 Puzzle igre

Potrebe mobilnih igara mijenjaju se ovisno o zemljama, na primjer u Velikoj Britaniji, najpoželjniji žanr igraćih aplikacija je *puzzle* (62,7%). Samo 1% više od povremenih igara (61,6). Ovaj žanr igara drugi je po reprodukciji na mobilnim uređajima u svijetu s više od 30% ukupnih preuzimanja. To su igre u kojima moramo smisliti strategiju kako riješiti neki problem. Unutar ovog žanra uvrštene su najpopularnije društvene igre, kao što su *Parcheesi, Domine, Scrabble*; kao i kartaške igre i zagonetke.

3 EMPIRIJSKI DIO RADA

U ovome dijelu rada razrađuje se izrada igre u *Unity* alatu. Na početku ću opisati tip igre koji sam izabrao za realizaciju. Nakon toga ću objasniti korištenje sprajtova (engl. *Sprite*) u *Unityju* i postupak njihove obrade. Opisat ću kako sam se odlučio za *sprajtove* koje ću koristiti. Sljedeće što je bilo važno je kako napraviti kretanje lika na željeni način. Kod igara animacije likova su jedan od osnovnih dijelova i u *Unityju* ih je relativno lako napraviti i za to sam upotrijebio *Unity* animator i alate za animaciju. Sljedeće čega ću se dotaknuti u ovom poglavlju je oblikovanje mape igre, kao i različitih razina koristeći *Tilemaps* alat unutar *Unityja*.

3.1 Opis igre

Tip igre koju obrađujem u ovom radu je 2D igra tj. u dvije dimenzije. Dvodimenzionalne igre fokusiraju se na plošna iskustva u kojima upravljanje ne zahtjeva potpunu kontrolu nad kamerom u prostoru. Kao i svaki žanr igara i 2D igre imaju svoje pod žanrove tako sam ja odabrao 2D *top down* igru. Kod *top down* igara važna je kamera koja je postavljena u ptičju perspektivu, pa time igru gledamo s povišene pozicije. Na slici 5 se vidi primjer jedne *top down* igre.

Slika 5 Primjer top down igre

Kako bih još malo produbio igru, odabrao sam *Dungeon Crawler* tip igre jer on odgovara ideji koju sam imao prije početka izrade ovoga rada. Kod tog tipa igre heroj prolazi kroz labirinte, boreći se protiv različitih čudovišta, izbjegavajući zamke i rješavaju zagonetke. Na slici 6 je prikazan izgled ekrana jedne igre tipa *Dungeon Crawler.*

Slika 6 Prikaz igre tipa Dungeon Crawler

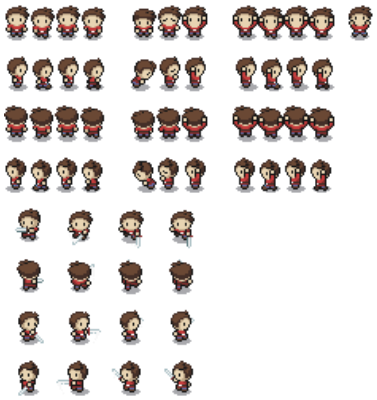
3.2 Sprajtovi, animacije i skripte u *Unityju*

Sprajtovi su 2D grafički objekti. Oni predstavljaju standardne teksture koje su prilagođene razvoju 2D igara. *Unity* ima nekoliko alata za obradu sprajtova od kojih se najčešće koriste što su *Sprite Editor* i *Sprite Renderer*.

3.2.1 *Sprite editor*

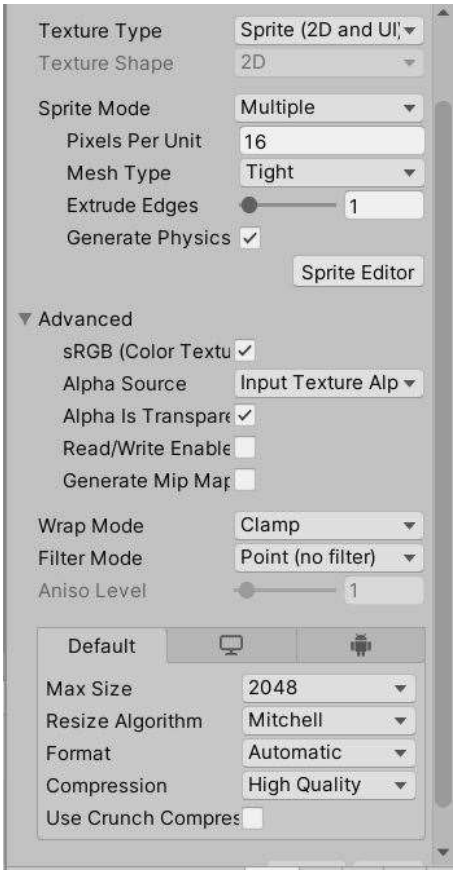
Sprite Editor je alat koji se koristi za rezanje većih slika u manje sprajtove. Editor sam koristio za rezanje elemenata korištenih u igri.

Za početak potrebno je umetnuti željeni izgled igre koji sam preuzeo sa interneta. Naravno izabrao sam element čije korištenje je dopušteno u svrhu izrade ovog rada. Zbog lakše preglednosti i bolje organiziranosti u *Unityju* sam napravio posebnu mapu za spremanje sprajtova. Umetanjem slike sa željenim izgledom lika ili mape trebamo podesiti neke stvari da nam sprajtovi dobro izgledaju i da igra dobro radi. Na slici 8 prikazana je slika lika koji je korišten u igri. Na slici je vidljiv skup manjih slika koje trebamo izrezati *Sprite Editorom* kako bismo dobili potrebne pojedinačne sprajtove.



Slika 7 Grupna slika lika koju trebamo izrezati

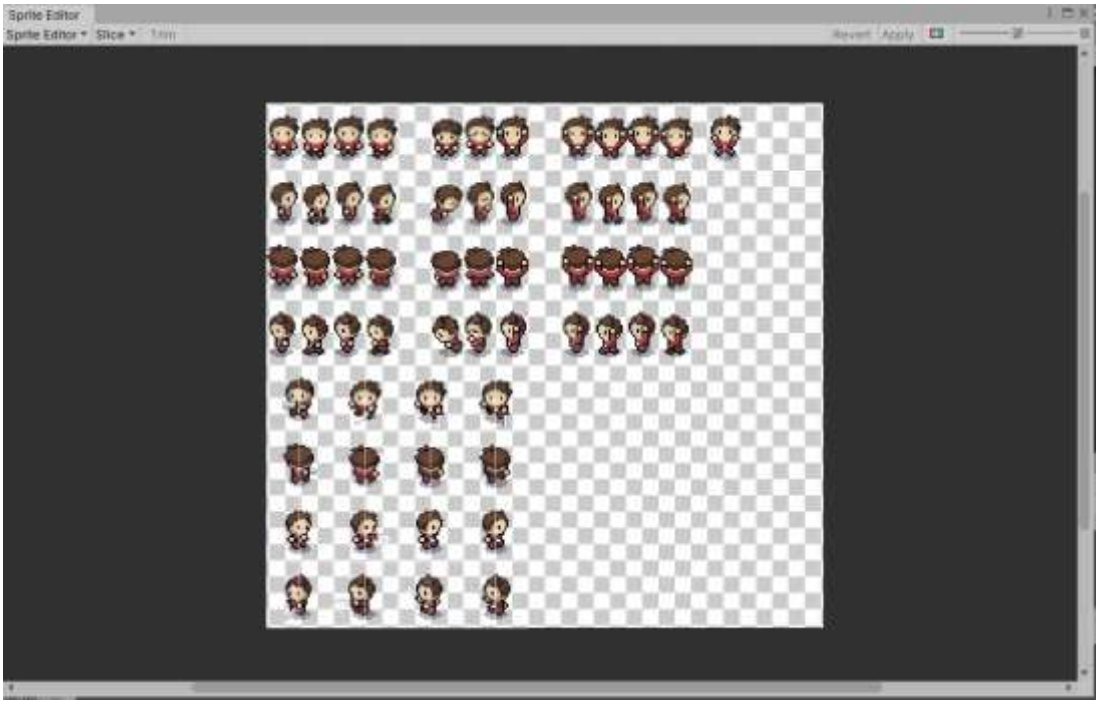
Na slici 9 vidi se izgled sučelja *Unity Inspectora* prije korištenja *Sprite Editora*. *Inspector* nam daje trenutne informacije o *GameObjectu* koji smo odabrali. Zbog toga jer koristimo sliku koja se sastoji od više manjih slika, trebamo postaviti *Sprite mode* na *Multiple*. Veličina svake sličice koju želimo dobiti treba biti 16x16 piksela. To postavimo u *Pixels Per Unit* izborniku. Da bi nam poslije sprajtovi izgledali bolje, trebamo postaviti *Filter Mode* na *Point (no filter)*. Sljedeći korak u radu je otvaranje samog *Sprite Editora* pritiskom na tipku *Sprite Editor*.



Slika 8 Izgled Inspector-a prije uporabe Sprite editora

Nakon otvaranja *Sprite Editora* otvara se ekran prikazan na slici 10. Da bismo dobili željene sprajtove treba izrezati sliku na komadiće. To činimo tako da izaberemo naredbu *Slice*, iz padajućeg izbornika *Type* izaberemo *Grid by Cell Size*. Nakon toga biramo

željenu veličinu, a naša je 16x16 piksela. Nakon toga možemo vidjeti da je naša slika izrezana i ako smo zadovoljni rezultatom potvrđujemo akciju pritiskom na *Apply* u desnom gornjem kutu ekrana.



Slika 9 Prikaz Sprite Editor-a

Ovo je je opis jednog primjera u kojem sam koristio *Sprite Editor*. Da bih dobio i ostale elemente igre ponovio sam isti postupak. Veće slike koje se sastoje od više elemenata izrezao sam u manje sprajtove odgovarajuće veličine. Na slici 11 prikazan je primjer izrezivanja dijelova mape koja se koristiti u igri.

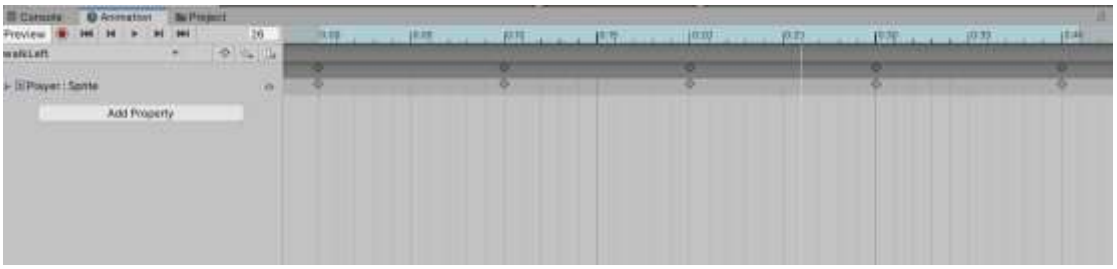
Slika 10 Primjer rezanja dijelova mape u Sprite Editoru

3.3 Glavni lik

U prethodnom potpoglavlju opisan je postupak rezanja sprajtova za glavnog lika i konačan rezultat prikazan je na slici 10. U ovom potpoglavlju opisat ću postupak izrade animacija, kretanja i napada glavnog lika.

3.3.1 Animacije

Za animaciju kretanja glavnog lika koriste se prethodno izrezani sprajtovi. Za animiranje lika koristio sam *Unity* alat *Animation*. Prvo što sam naravio je da sam dodao animacije lika koji stoji na mjestu. Ta animacija se sastoji samo od jedne slike. Animirao sam kretanje lika u svim smjerovima u kojima se može kretati (gore, dolje, lijevo i desno). Nakon toga sam animirao kretnje lika. Na slici 12 prikazan je *Animation* prozor.



Slika 11 Prikaz Aniation-a za kretanje lika u lijevo

Animation prozor se sastoji od većeg dijela na koji umećemo slike koje želimo animirati. Te slike umećemo na vremensku traku. Traka odgovara milisekundama animacije. Da bi lik bio dobro animiran, treba paziti na vremenski razmak. U prikazanom primjeru koristim četiri različite slike koje trebam animirati. Prva slika postavljena je na 0:00, a svaka sljedeća je pomaknuta za daljnjih 10.00 milisekundi. Zadnja slika je stavljena dva puta za redom kako bi animacija bila bolja. Da na raspolaganju imamo više slika, vremenski raspored bi trebao biti drugačiji. Pravilo je da što više slika imamo, to je animacija bolja.

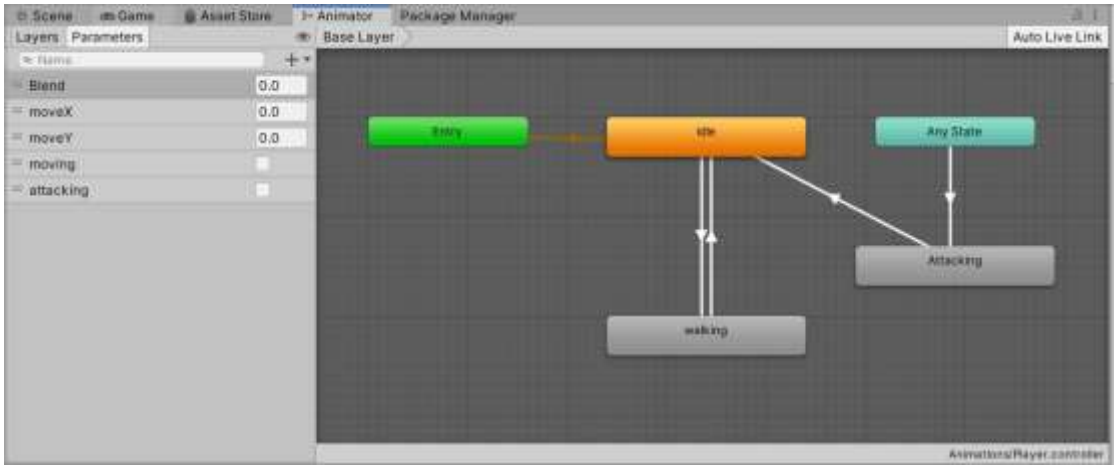
Opisani postupak ponovljen je za svaki smjer kretanja, samo sa slikama koje odgovaraju smjeru kretanja koji se u konkretnom slučaju animira.

Sljedeća stvar koju sam animirao za glavnog lika je napad. Prvo sam trebao napraviti sprajtove za napad. Proces pravljenja sprajtova je prethodno opisa, a na slici 13 prikazani su sprajtovi korišteni za prikaz napada glavnoga lika.

Slika 12 Sprite-ovi za napad glavnog lika

Animiranje je provedeno na isti način kao i animiranje kretanja glavnog lika. Kao što se vidi na slici 13. za svako kretanje tijekom napada postoji odgovarajući sprajt. Postupak animacije napada identičan je postupku animacije kretanja te je i napad trebalo animirati za svaki smjer u kojem se napad može izvršiti..

Nakon što su pripremljene sve animacije, treba ih i iskoristiti odnosno povezati. To je napravljeno u *Animatoru*. *Animator* je dio *Unityja* koji služi za povezivanje animacija tj. za prelaze između animacija, kao i postavljanje uvjeta koji određuju kada se koja animacija izvršava. Na slici 14 prikazan je primjer prozora *Animatora*.



Slika 13 Prikaz Animatora za glavnog lika

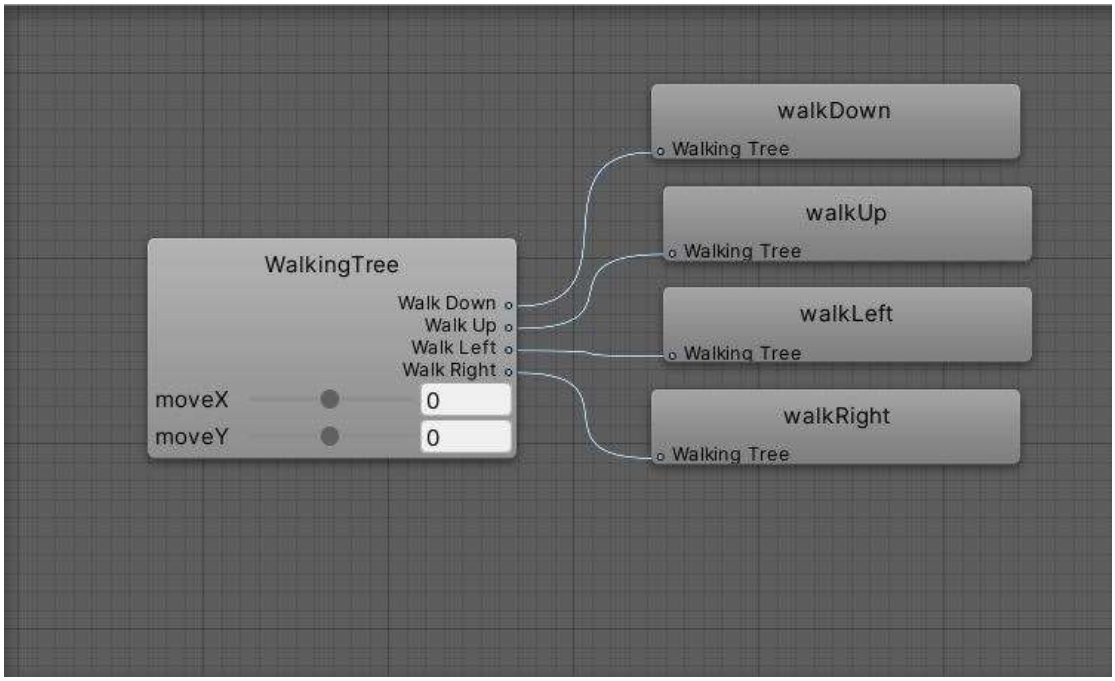
U desnom dijelu prozora prikazane su skupine različitih animacija koje sam kod glavnog lika nazvao *idle* (početna animacija), *walking* (animacija za kretanje) i *attacking* (animacija za napad). Svaka animacija prikazan je jednim pravokutnikom i svaki prikazani pravokutnik sadrži skupinu animacije za dio koji je animiran.

Kao sto se vidi na slici 14, pravokutnici su povezani i te veze se koriste za prijelaze između animacija. Na lijevoj stari ekrana dodajemo parametre čija promjena će utjecati na to koja će se od animacija izvesti. Parametri koje sam dodao su *moveX* (promjena x koordinate), *moveY* (promjena y koordinate), logički parametar *moving* koji je istina ako se lik kreće i logički parametar *attacking* koja ima vrijednost istinu ako lik napada.

Imamo nekolika smjerova izvršenja animacija. Dvi su vrste veze između *idle walking* animacija. Ako je parametar *moving* istina, onda prelazimo na animaciju *walking*. U suprotnom, ako je vrijednost parametra *moving* laž, prelazimo na *idle* animaciju.

Animaciji napada možemo pristupiti iz bilo koje druge animacije ako je parametar *attacking* istina. Zbog toga je pravokutnik *Any State* povezan s pravokutnikom *Attacking*. *Attacking* je također povezan s *idle* pravokutnikom te se *idle* animacija izvršava nakon što prestane animacija *attacking*.

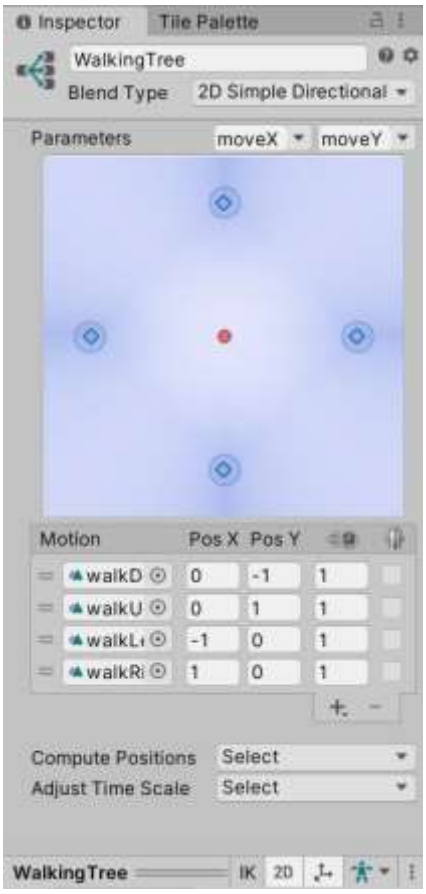
Svaki pravokutnik predstavlja skupinu animacija. Njihov sadržaj možemo vidjeti ako dvaput kliknemo na neku od njih. Na slici 15 se nalazi prikaz stabla animacija. Kao što sam prethodno napisao, dodani su parametri za promjene x i y osi tj. *moveX* i *moveY*.



Slika 14 Prikaz stabla aminacija

Na *WalkingTree* su dodane različite animacije za promjene x ili *y* osi. Npr. ako je x = 0, a y = -1, s tim znamo da se lik kreće prema dolje i da treba pokrenuti animaciju za kretanje prema dolje, Tako isto vrijedi za sve ostale kretnje.

Također za napad je napravljeno slično stablo kako bi se pokrenula animacija za napad u točno onom smjeru u kojem se lik kreće. Na slici 16 se vidi kako je postavljeno stablo za izvršavanje točne animacije. Parametri su postavljeni na *moveX* i *moveY* što znači da pratimo promjene tih parametara. Ispod se nalazi prikaz izvršenja animacija u smjeru parametara koje smo zadali. Pritiskom na tipku + dodajemo nove *Motione* za sve različite parametre kako što se vidi na slici 16.



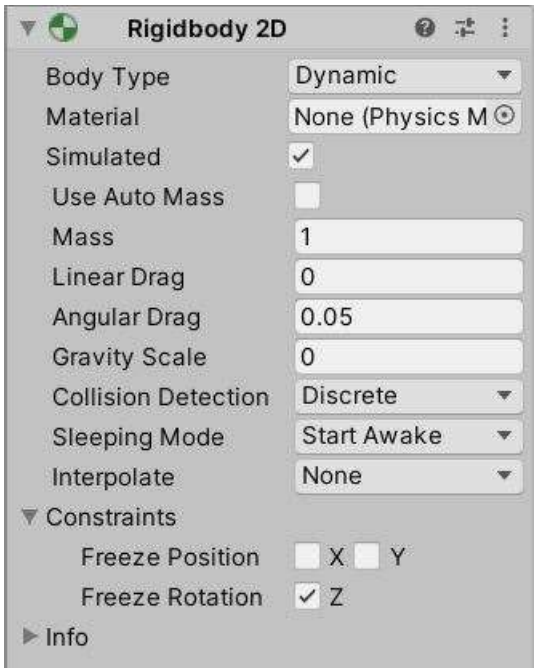
Slika 15 Prikaz Inspector-a za stablo kretanja

3.4 *Rigidbody* 2D

Rigidbody 2D je komponenta koju stavljamo na objekt kada želimo da s njim upravlja fizički *engine Unityja*. To znači da objekt postaje „živ“ odnosno možemo pomoću skripti upravljati objektom u prostoru. U *Unityju* postoje dvije komponente: *Rigidbody* i *Rigidbody* 2D. Razlika među njima je ta da se objekti na koje je stavljen *Rigidbody* 2D mogu se kretati samo po osima X i Y i rotirati na jednoj osi [9].

Rigidbody 2D komponenta ima različite opcije. Jedna od najvažnijih je *Body Type*. Postoje tri vrste *BodyTypea*, a to su *Dynamic*, *Kinematic* i *Static*. *Objektima tipa Dynamic.* možemo mijenjati masu (*mass*), opterećenje (*drag*), gravitaciju (*gravity*) i silu (*force*) koje utječu na objekt. Objekti tipa *Kinematic* su brzi za izvođenje i koriste manje resursa. Dizajnirani su tako da koriste fizikalne osobine iz stvarnoga svijeta putem skripti. To znači da imamo više slobode i mijenjamo odnosno koristimo samo ona osobine koja želimo. Objekti tipa *Static* su statični tj. na njih ne utječe nikakve fizikalne osobine. Koristimo ih za objekte koji su statični i ne miču se.

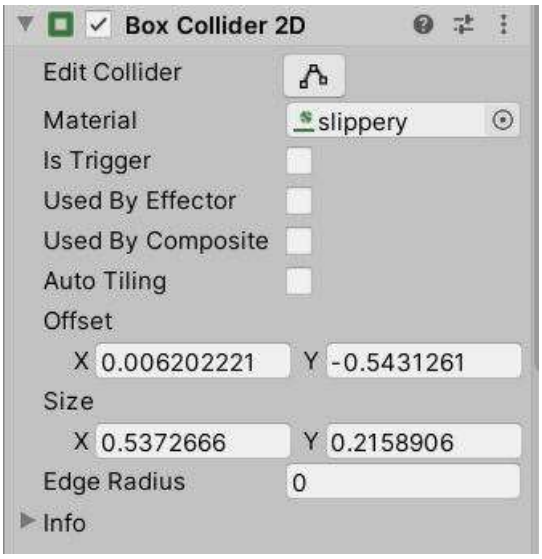
Na slici ispod 17 prikazana je *Rigidbody* 2D komponenta s *Player* objektom koji predstavlja glavni lik koji u projektu zovem *Player*. *Body Type* je postavljen na *Dynamic*, *Gravity* je postavljen na 0 jer ne želimo da na objekt utječe gravitacija. *Freeze Rotation* je postavljen na *true* jer ne želimo da se objekt može okretati po osi z.



Slika 16 Prikaz Rigidboy 2D komponente na Playeru

3.5 *Box Collider* 2D

Box Collider je nevidljivi oblik koji se koristi za određivanje kolizije objekta na koji je postavljen. *Colliser* za 2D objekte je pravokutnog oblika koji otprilike predstavlja oblik sprajta objekta. Na slici 18 prikazan je *Box Collider* 2D postavljen na objekt *Player*.



Slika 17 Box Collider Player

Na slici 19 prikazan je izgled *Box Collidera* objekta Player u *Scene* prozoru. Mali zeleni pravokutnik na dnu sprajta lika predstavlja *Box Collider*. On je vidljiv samo u *Scene* prozoru, a kada pokrenemo igru on se ne vidi. Kolizija je mala jer želimo da objekt može proći oko malih objekata. Da je kolizija veća, izgledalo bi da objekt ne može proći pokraj drugog manjeg objekta.

Slika 18 Prikaz Box Collidera na Player objektu

3.6 Kretanje glavnog lika

Kretanje lika obradio sam u skripti. *Unity* za pisanje skripti koristi C#, objektno orijentirani programski jezik. Kako sam imao opciju iskoristiti sve pogodnosti objektnog pristupa, iskoristio sam ih u nekim od narednih skripti koje su korištene u ovom radu. Kako imam samo jednog glavnog lika, nisam trebao raditi nadređenu klasu klasi *Player*.

U nastavku je objašnjena skripta za kretnje lika. Na slici 20 prikazan je jedan dio programskog koda iz skripte *CharacterMovment*. Na samom početku skripte definirao sam *public enum*. *Enum* je tip nabrajanja koji je definiran skupom konstanti koje se definiraju unutar vitičastih zagrada. Unutar *enuma* sam dodao različita stanja u kojima se može naći lik. Klasa *CharacterMovment* je pod klasa *MonoBehaviour*-a.

Samim otvaranjem skripte ponuđene su dvije metode, *Start()* i *Update()*. U *Start* metodi definirano je početno stanje lika. Postavlja se početno stanje lika, pozicija, opisuje način hodanja lika, pokreću se animacije i postavljaju *moveX* i *moveY* na željene vrijednosti. Metoda *Update* se izvodi na svakom *frameu* što znači da prati promjene varijabli. U metodi su postavljena dva *if* grananja kako bi se postavili uvjeti za izvođenje metoda u kojima se obrađuju animacije lika ako hoda i da izvodi napade. Metodu *UpdateAnimationAndMove()* i korutinu *AttackCo()* ću objasniti na sljedećem dijelu programskog koda.

```
public enum PlayerState
    {
    walk,
    attack,
    interact,
    stagger,
    idle
    }

public class CharacterMovment : MonoBehaviour
{
    public PlayerState currentState;
    public float speed;
    private Rigidbody2D myRigidbody;
    private Vector3 change;
    private Animator animator;

    void Start()
    {
        currentState = PlayerState.walk;
        animator = GetComponent<Animator>();
        myRigidbody = GetComponent<Rigidbody2D>();
        animator.SetFloat("moveX", 0);
        animator.SetFloat("moveY", -1);
    }
    // Update is called once per frame
    void Update()
    {
        change = Vector3.zero;
        change.x = Input.GetAxisRaw("Horizontal");
        change.y = Input.GetAxisRaw("Vertical");
        if (Input.GetButtonDown("attack") && currentState != PlayerState.attack
            && currentState != PlayerState.stagger)
            {
            StartCoroutine(AttackCo());
            }
        else if (currentState == PlayerState.walk || currentState == PlayerState.idle)
            {
            UpdateAnimationAndMove();
            }
     }
```

Slika 19 Primjer koda skripte CharacterMovment

IEnumerator je sučelje korišteno za korutinu *AttackCo()*. U *AttackCo* postavlja se animator na *attacking true*, i nakon što završi napad postavlja se *attacking* na *false* i čeka 3 milisekunde. Nakon toga se vraća se na animaciju hodanja. Metoda *UpdateAnimationAndMove()* koristi se za pokretanje animacije kretanja lika. Postavljena su dva *if* izraza da bi se moglo vidjeti da li se lik kreće ili ne. Ako se kreće poziva se metoda *MoveCharacter()* i postavljaju vrijednosti *moveX* i *moveY* kako bismo znali u kojem se smjeru lik kreće te da bismo pokrenuli prave animacije. Metoda *MoveCharacter* računa u kojem smjeru se lik kreće. Smjer pomnožimo s brzinom koju samo određujemo i s *Time.deltaTime* kako bi se igra dobro izvodila na svakom računalu. Metode *Knock()* i korutina *KnockCo()* ću objasniti poslije jer su usko povezane sa skriptom kretanja protivnika.

```
private IEnumerator AttackCo()
        {
            animator.SetBool("attacking", true);
            currentState = PlayerState.attack;
            yield return null;
            animator.SetBool("attacking", false);
            yield return new WaitForSeconds(.3f);
            currentState = PlayerState.walk;
        }

    void UpdateAnimationAndMove()
        {
        if (change != Vector3.zero)
            {
            MoveCharactert();
            animator.SetFloat("moveX", change.x);
            animator.SetFloat("moveY", change.y);
            animator.SetBool("moving", true);
            }
        else
            {
            animator.SetBool("moving", false);
            }
        }

    void MoveCharactert()
        {
        change.Normalize();
        myRigidbody.MovePosition(transform.position + change * speed * Time.deltaTime);

        }

    public void Knock(float knockTime)
        {
        StartCoroutine(KnockCo(knockTime));

        }
    private IEnumerator KnockCo(float knockTime)
        {
        if (myRigidbody != null)
            {
            yield return new WaitForSeconds(knockTime);
            myRigidbody.velocity = Vector2.zero;
            currentState = PlayerState.idle;
            myRigidbody.velocity = Vector2.zero;
            }
        }
    }
```

Slika 20 Nastavak skripte CharacterMovment()

3.7 Neprijatelji

Uz glavnog lika neprijatelji su glavni dio igre na kojoj sam radio. Oni služe da bi igra bila što zanimljivija i da bi samom igraču dali bolji doživljaj igre kao kako igra ne bi bila previše dosadna.

3.7.1 Animacije

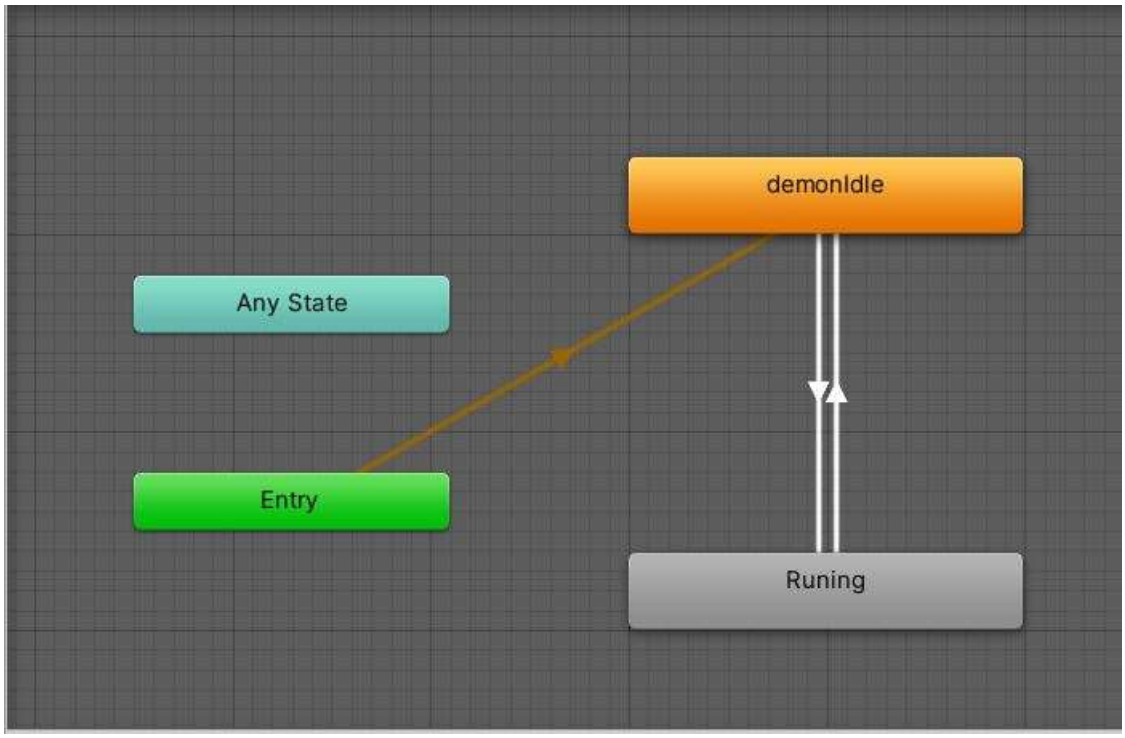
Postupak animiranja neprijatelja je isto kao i animiranje glavnog lika. Na slici 22 vidi se izgled neprijatelja.

Slika 21 Prikaz izgleda neprijatelja

Kao i glavni lik, i neprijatelj je izrezan u *Sprite Editoru* iz veće slike te su dobivene manje slike koje se koriste za animaciju neprijatelja . Na slici 23 prikazane se slike koje sam upotrijebio za animiranje neprijatelja.

Slika 22 Prikaz slika prije animacije

Na slici 24 prikazan je izgled *Animator* prozora. Kao što se vidi postavljeni su prelazi iz početne animacije u animaciju napada. Za odluke koja se animacija izvršava dodani su isti parametri kao i kod animiranja glavnog lika. Promjenom parametara animacije se mijenjaju. Samim parametrima upravljamo pomoću skripte.



Slika 23 Prikaz Animator prozora za neprijatelja

3.7.2 Kretanje neprijatelja

Kretanje neprijatelja je napravljeno pomoću skripte kao i kod glavnog lika, samo s nekim dodatcima. Neprijateljem ne upravlja igrač već samo računalo, pa je zbog toga trebalo dodati dodatne varijable koje se koriste da bi da neprijatelj reagirao samo u određenim situacijama kada mu se glavni lik približi.

Na slici 25 prikazan je programski kod klase *Enemy*. Pošto skripte u *Unityju* radimo koristeći jezik koji nam daje mogućnost objektnog pristupa, iskorištene su objektne mogućnosti. Klasa *Enemy* je postavljena kao glavna klasa, nadređena svim drugim klasama neprijatelja. Klase koje predstavljaju razne neprijatelje bit će podređene klasi *Enemy* i nasljeđivat će varijable i metode klase *Enemy*. U klasi *Enemy* sam postavio varijable koje sam koristio. U *Awake()* metodi postavlja se varijabla koja predstavlja razinu zdravlja neprijatelja na željenu vrijednost. U ovom slučaju postavljena je na 2. To znači da kada neprijatelja udarimo dva puta on „umire“. Metoda *TakeDamage()* provjerava koliko je zdravlje i ako je manje od 0, neprijatelja označavamo i prikazujemo neaktivnim.

Metoda *Knock()* i korutina *KnockCo()* su zanimljive jer daju dodatni element igri, a to je da kada igrač napravi štetu neprijatelju, neprijatelja odgurnemo od igrača i određeno kratko vrijeme (vrijednost *knockTime* varijable) neprijatelj postaje neaktivan. Metoda *Knock()* na glavnom liku radi isto kao i na neprijatelju, tj. ako neprijatelj dodirne glavnog lika on ga odgurne za postavljenu vrijednost.

```
public enum EnemyState
    {
    idle,
    walk,
    attack,
    stagger
    }

public class Enemy : MonoBehaviour
{
    public EnemyState currentState;
    public FloatValue maxHealth;
    public float health;
    public string enemyName;
    public int baseAttack;
    public float moveSpeed;

    private void Awake()
        {
        health = maxHealth.initialValue;
        }

    private void TakeDamage(float damage)
        {
        health -= damage;
        if(health <= 0)
            {
            this.gameObject.SetActive(false);
            }
        }

    public void Knock(Rigidbody2D myRigidbody, float knockTime, float damage)
        {
        StartCoroutine(KnockCo(myRigidbody, knockTime));
        TakeDamage(damage);
        }

    private IEnumerator KnockCo(Rigidbody2D myRigidbody, float knockTime)
        {
        if (myRigidbody != null)
            {
            yield return new WaitForSeconds(knockTime);
            myRigidbody.velocity = Vector2.zero;
            currentState = EnemyState.idle;
            myRigidbody.velocity = Vector2.zero;
            }
        }

    }
```

Slika 24 Prikaz klase Enemy

Klasa koja opisuje kretanje neprijatelja se zove *Demon* i nasljeđuje klasu *Enemy*. Programski od klase je prikazan na slici 27. Sama klasa ima nekoliko dijelova koje sam trebao sam isprogramirati kako bi neprijatelj izgledao i ponašao se onako kako želim. Jedno od rješenja je okretanje lika po x osi. Na početku nisam imao sliku lika koji gleda u lijevu stranu pa sam u *FixedUpdate()* metodi to popravio, tako da provjeravam horizontalnu os i vrijednosti x.. Ako je x manji od nule okrećemo ga, a ako nije, ostaje isti.

U metodi *CheckDistance()* obrađujemo kretanje neprijatelja. Varijable *chaseRadius* i *AttackRadius* služe za određivanje da li je igrač u zadanom radijusu. Ako jest, neprijatelj se pokreće i počinje slijediti glavnog lika. Sve dok je lik u tom radijusu neprijatelj će ga pratiti. Druga varijabla je *attackRadius*. Ona je također zadana i govori nam na kojoj udaljenosti glavni lik može napasti neprijatelja i obratno.

Na slici 26 vidimo prikaz *Inspectora* u *Unityju* . U *Inspectoru* su vidljive sve varijable u skripti imaju javnu razinu pristupa te ih možemo ručno postaviti na željenu vrijednost. Dobra stvar ručnog postavljanja je da možemo testirati rad igre za različite vrijednosti varijabla i samim time igru prilagoditi kako bi bila što fluidnija i bolja za igranje.

Slika 25 Prikaz Inspectora za skriptu Demon

```
public class Demon : Enemy
{
    private Rigidbody2D myRigidbody;
    public Transform target;
    public float chaseRadius;
    public float attackRadius;
    public Transform homePosition;
    public Animator anim;
    // Start is called before the first frame update
    void Start()
    {
        currentState = EnemyState.idle;
        myRigidbody = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        target = GameObject.FindWithTag("Player").transform; //transform = scale,
rotation, position

    }

    // Update is called once per frame
    void FixedUpdate()
    {
        CheckDistance();
        Vector3 characterScale = transform.localScale;
        if (Input.GetAxis("Horizontal") < 0)
            {
            characterScale.x = -1;
            }
        if (Input.GetAxis("Horizontal") > 0)
            {
            characterScale.x = 1;
            }
        transform.localScale = characterScale;

        }

    void CheckDistance()
        {
        if (Vector3.Distance(target.position, transform.position) <= chaseRadius
            && Vector3.Distance(target.position, transform.position) > attackRadius)
            {
            if (currentState == EnemyState.idle || currentState == EnemyState.walk
                && currentState != EnemyState.stagger)
                {
                Vector3 temp = Vector3.MoveTowards(transform.position,
target.position, moveSpeed * Time.deltaTime);
                myRigidbody.MovePosition(temp);
                changeAnim(temp - transform.position);
                ChangeState(EnemyState.walk);
                anim.SetBool("moving", true);
                }
            }
        else
            {
            anim.SetBool("moving", false);
            }
        }
```

Slika 26 Prikaz dijela klase Demon

```
private void ChangeState(EnemyState newState)
        {
        if(currentState != newState)
            {
            currentState = newState;
            }
        }
    private void changeAnim(Vector2 direction)
        {
        direction = direction.normalized;
        anim.SetFloat("moveX", direction.x);
        anim.SetFloat("moveY", direction.y);
        }
```

Slika 27 Primjer klase Demon

Metoda *ChangeState()* mijenja *EnemyState* tj. mijenja stanja neprijatelja ovisno o tome u kojem stanju se nalazi i u koje treba doći (npr. iz kretanja u stanje mirovanja).

Metoda *changeAnim()* mijenja animacije neprijatelja ovisno o smjeru u kojem se nalazi. Promjenom parametara *moveX* i *moveY* mijenja se i animacija.

3.8 Izrada mape igre

Bez izgleda mape igre tj. bez okolnih elemenata okruženja s kojima će lik ili neprijatelj dolaziti u interakciju, igra ne bi ni postojala. Stil za koji sam se odlučio je tipičan za vrstu igre koju radim a to je mračni stil sa izgledom podzemnih zatvora iz srednjeg vijeka.

Elemente izgled mape sam izrezao s veće slike pomoću *Sprite Editora* i prilagodio ih veličini koja odgovara igri. Na slici 29 prikazan je izgled slika nakon rezanja i pripreme u *Sprite editoru*.

Slika 28 Prikaz izgleda pojedinih slika prije izrade mape

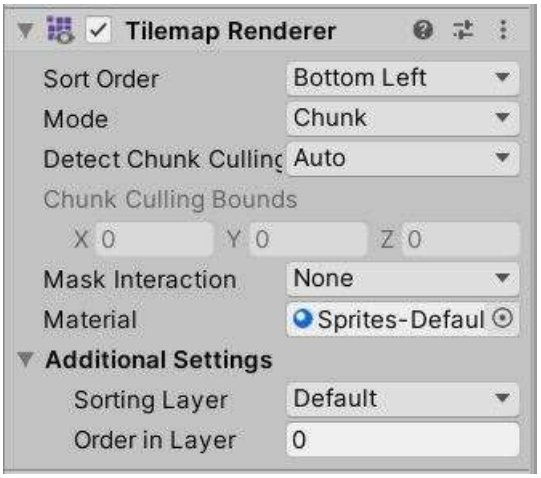
3.8.1 Tilemap u Unityju

Tilemap je komponenta koja sadržava i upravlja *Tile assete* za kreiranje 2D mapa. Da bi koristili *Tilemap* prvo da treba dodati objekt *Grid* koji nam omogućuje stavljanje *Tileova* na grid koji se vidi u *Scene* prozoru. Nakon toga kao objekte podređene objektu Grid dodajemo objekte tipa *Tilemap*. *Tilemap* se nalazi u izborniku pod 2D objektima.

U svome projektu dodao sam dva objekta, jedan za pod po kojem će se kretati glavni lik i neprijatelji i drugi za kolizije da napravim ograničenja da se lik ne može kretati izvan mape. Na slici 30 prikazan je raspored tih objekata.

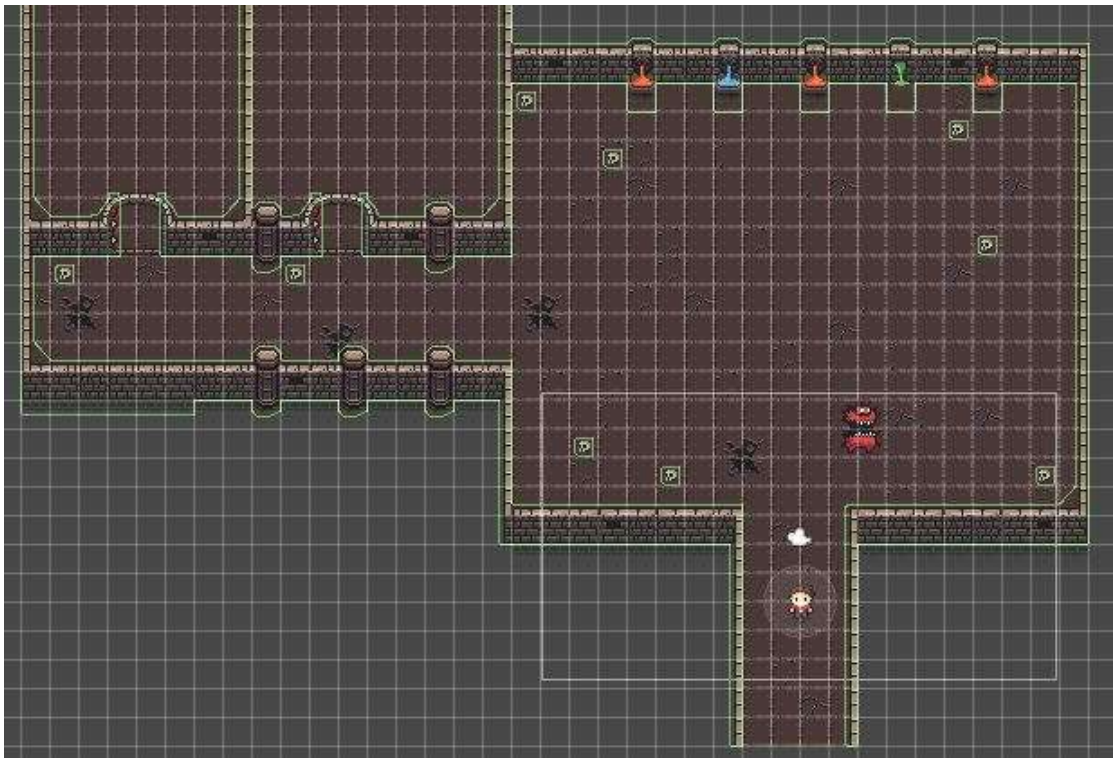
Slika 29 Raspored objekata

Na svaki objekt *Ground* i *Collision* u *Inspektoru* pritiskom na gumb *Add Component* dodajemo *Tilemap Renderer* koji služi za renderiranje mape. To služi da bismo vidjeli one sprajtove koje smo nalijepili na grid u prozoru *Scene*. Na slici 31 prikazan je izgled *Tilemap Renderera*.



Slika 30 izgled Tilemap Renderera

Da bi igra imala opciju kolizije glavnog lika ili neprijatelja sa zidovima i preprekama na objekt *Collision* dodane su komponente *Tilemap Collider* 2D, *Rigidbody* 2D i *Composite Collider* 2D. U *Tilemap Collider* 2D podešena je opcija *Use By Composite* na *true*. Korištenje *Composite Collidera* omogućava da ne moramo za svaki sprajt izrađivati posebnu koliziju nego se ona sama generira. Na slici 32 prikazan je generirana kolizija sa zelenom crtom uokolo zidova na mapi.



Slika 31 Prikaz kolizija na mapi

3.8.2 Tile Palette

Da bi započeli stvarati mapu za igru trebamo sprajtove dodati u *Tile Paletteu*. Otvaramo ju tako da idemo na *Window > 2D > Tile Palette*. Nakon toga će se otvoriti novi prozor. Da bi stvorili novu paletu pritisnemo na *Create New Palette*, odaberemo ime palete i pritisnemo tipku *Create*. Nakon toga imamo paletu ali ona je prazna. Nakon toga pozicioniramo se u mapu gdje se nalaze sprajtovi za mapu, označimo ih sve i prebacimo u prozor palete. Na slici 33 prikazan je paleta koju sam koristio kod izrade mape za igru.

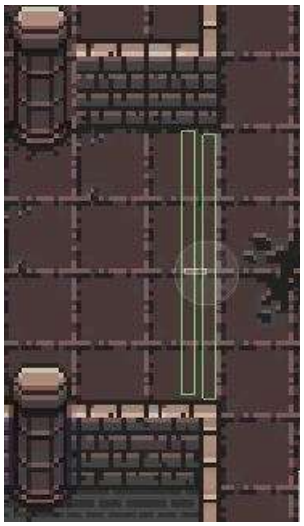
Slika 32 Prikaz palete

U padajućem izborniku *Active Tilemap* odabiremo dio mape koji želimo slikati. Ja imam dvije *Tilemape, Ground* i *Collizion* i trebao sam paziti koju koristim, jer za dio po

kojemu se mogu kretati lik i neprijatelji nikako ne smije biti *Collision Mapa* jer se ne bi mogli kretati zbog kolizija. Prvo sam napravio okvirni izgled prostora po kojem se kreću lik i neprijatelji pa sam zatim na mjestima gdje želim da lik i neprijatelj ne mogu proći ili želim da bude neka prepreka koristio mapu za kolizije.

3.9 Korištenje kolizija za prelazak iz jednog u drugi dio mape

Jedan od dijelova igre koja korisnika čini zadovoljnijim je kretanje kroz mapu tj. ugodniji prelazak iz jednog dijela u drugi. To sam napravio upotrebom *Box Collidera* 2D i jednostavne skripte. Jedan *Box Collider* 2D sam postavio na ulazak u drugi dio mape. Taj objekt je radio samo u jednom smjeru i kada bi lik prešao u drugi dio mape kada bi se želio vratiti u dio mape odakle je došao ne bi se dogodilo ništa. Zbog toga sam dodao i drugi *Box Collider* 2D koji radi istu stvar samo u drugom smjeru i tako sam dobio prilično lako prelazak iz jednog dijela u drugi dio mape. Ova tehnika se može koristiti na više načina i više puta tijekom izrade igre. Na slici 34 vide se dva objekta koja sam koristio.



Slika 33 Prikaz Box Collider-a

Skriptu koja se koristi za prelazak iz jednog djela mape u drugi sam nazvao *RoomMove*. Skripta je jednostavna, U metodi *OnTrrigerEnter2D* se prima parametar *Collider2D* i , provjerava se je li lik dodirnuo kolider, te se odrađuje ostatak metode. Da bi prelazak bio dobro odrađen trebamo i lika i kameru pomaknuti u smjeru gdje se nalazi drugi dio mape, i to se odrađuje u toj metodi.

Na slici 35 prikazan je programski kod klasa *RoomMove* s pripadajućim metodama.

```
public class RoomMove : MonoBehaviour
{
    public Vector2 cameraChange;
    public Vector3 playerChange;
    private CameraMovment cam;

    // Start is called before the first frame update
    void Start()
    {
        cam = Camera.main.GetComponent<CameraMovment>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    private void OnTriggerEnter2D(Collider2D other)
        {
        if (other.CompareTag("Player") && !other.isTrigger)
            {
            cam.minPosition += cameraChange;
            cam.maxPosition += cameraChange;
            other.transform.position += playerChange;
            }
        }
}
```

Slika 34 Prikaz klase RoomMove

3.10 2D svjetlo u *Unityju*

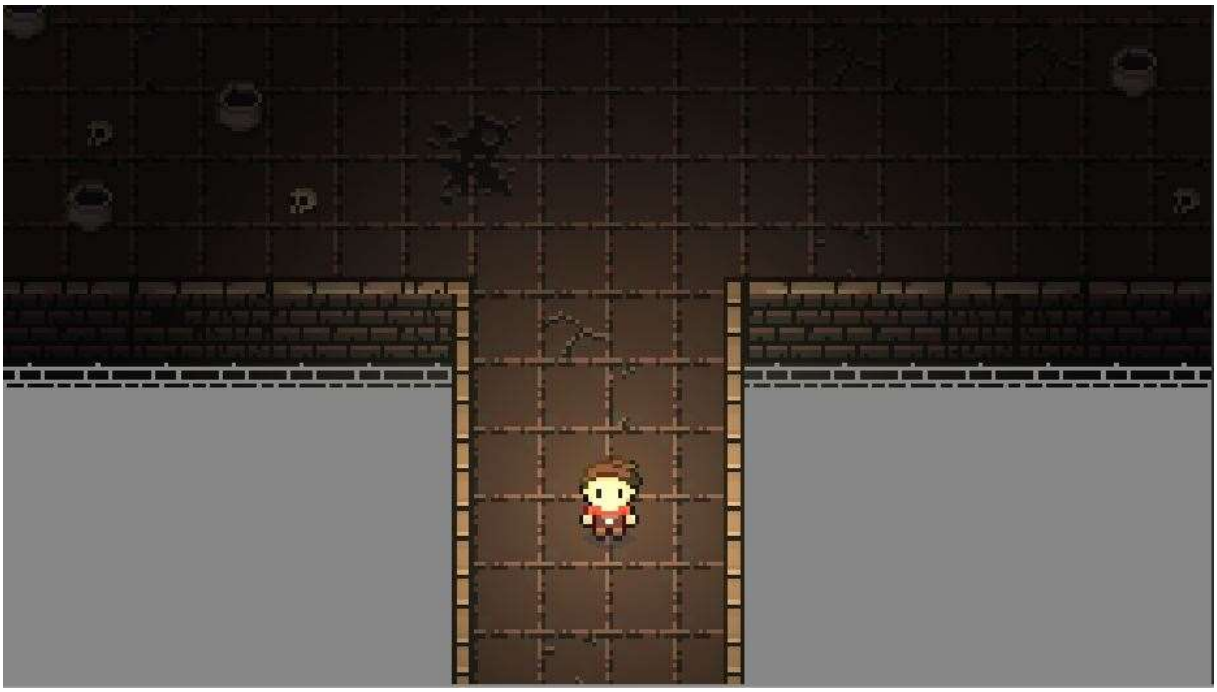
Jedna od zanimljivih noviteta u novijim verzijama *Unityja* je 2D osvjetljenje. Prije dodavanja novih opcija za 2D svjetlo programeri su morali koristiti svijetlo prilagođeno razvoju 3D igara. 3D svijetlo nepotrebno oduzima resurse jer renderira svjetlo u jednoj dodatnoj dimenziji koja se u 2D igri uopće ne koristi. *Unity* počevši od verzije 2019 uvodi mogućnost dodavanja 2D osvjetljenja.

Da bi dodali 2D svjetlo treba otvoriti izbornu traku i pritisnuti *GameObject* > *Light* > 2D te odabrati neki od ponuđenih tipova svjetla. Ponuđeni tipovi su:

- *FreeForm*: Možemo sami prilagoditi oblik svijetla našim željama,
- *Sprite*: Možemo prilagoditi oblik svijetla nekom sprite-u po želji,
- *Parametric*: Možemo oblikovati svjetlo u obliku n-stranom poligonu,
- *Point*: Možemo kontrolirati unutarnji i vanjski krug, smjer i kut svijetla,
- *Global*: Ovaj tip utječe na sve objekte na sceni.

U ovom projektu korišten je *Point* tip svjetla i dodan je na glavnog lika. S time je dobiven efekt da kada se lik kreće, osvjetljuje dio mape u radijusu koji je postavljen.

Na slici 36 prikazan je izgled igre nakon dodavanja *Point* svjetla.



Slika 35 Prikaz igre nakon dodavanja Point svjetla

4 ZAKLJUČAK

U ovom radu opisan je postupak izrade 2D igre tipa *Dungeon Crawler* koristeći *Unity* razvojno okruženje. Igra se može bez većih izmjena koristiti na različitim operacijskim sustavima.

Tijekom izrade rada stekao sam jako puno korisnog znanja koje će mi puno koristiti u budućnosti. Sama izrada rada je bila prilično izazovna jer se prije nisam susreo sa alatom *Unity* te sam trebao dosta učiti tijekom izrade rada.

U radu sam pokazao kako izrada jednostavnih 2D igara nije previše teška nakon što se usvoje osnovne radnje u *Unityju*, ali kako sam ja počeo bez opširnijeg znanja o tom razvojnom okruženju i postupku izrade igara u njemu trebao sam se potruditi kako bih savladao osnove. Unatoč tome izrada je bila je jako zabavna i svaki trenutak proveden u izradi projekta mi nije bio težak.

5 LITERATURA

[1] »phoneArena,« [Mrežno]. Dostupno na: https://web.archive.org/web/20190522004814/https://www.phonearena.com/news/This-was-the-worlds-first-cell-phone-with-a-game-loaded-on-it_id62920. Pristupljeno 28.9.2020.

[2] N. Connects. [Mrežno]. Dostupno na: http://nokiaconnects.com/2011/02/15/7-nokia-world-records-that-will-blow-your-mind/. Pristupljeno 28.9.2020.

[3] N. M. J. B. M. D. R. E. P. T. Behrmann M, »State of the Art of the European Mobile Games Industry,« 2017.

[4] »Mobile games sparked 60% of 2019 global game revenue, study finds,« [Mrežno]. Dostupno na: https://www.mobilemarketer.com/news/mobile-games-sparked-60-of-2019-global-game-revenue-study-finds/569658/. Pristupljeno 28.9.2020.

[5] »Mobile Revenues Account for More Than 50% of the Global Games Market as It Reaches $137.9 Billion in 2018,« [Mrežno]. Dostupno na: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/. Pristupljeno 28.9.2020.

[6] »Unity at 10: For better—or worse—game development has never been easier,« [Mrežno]. Dostupno na: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/. Pristupljeno 28.9.2020.

[7] »DeepMind partners with gaming company for AI research,« [Mrežno]. Dostupno na: https://www.dailydot.com/debug/unity-deempind-ai/. Pristupljeno 28.9.2020.

[8] »Unity’s asset store boss has big plans to fight Epic’s Unreal,« [Mrežno]. Dostupno na: https://venturebeat.com/2018/07/18/unitys-asset-store-boss-has-big-plans-to-fight-epics-unreal/. Pristupljeno 28.9.2020.

[9] U. Documentation, »Rigidbody 2D,« [Mrežno]. Dostupno na: https://docs.unity3d.com/Manual/class-Rigidbody2D.html. Pristupljeno 28.9.2020.

6 PRILOZI

6.1 Popis slika

Slika 1 Nokia Snake 2
Slika 2 Prikaz rasta prodaje mobilnih igara tijekom godina [5] 4
Slika 3 Prikaz izgleda prozora alata Unity 5
Slika 4 Prikaz izgleda Unity Asset Store-a 6
Slika 5 Primjer top down igre 8
Slika 6 Prikaz igre tipa Dungeon Crawler 9
Slika 8 Grupna slika lika koju trebamo izrezati 10
Slika 9 Izgled Inspector-a prije uporabe Sprite editora 10
Slika 10 Prikaz Sprite Editor-a 11
Slika 11 Primjer rezanja dijelova mape u Sprite Editoru 11
Slika 12 Prikaz Aniation-a za kretanje lika u lijevo 12
Slika 13 Sprite-ovi za napad glavnog lika 13
Slika 14 Prikaz Animatora za glavnog lika 13
Slika 15 Prikaz stabla aminacija 14
Slika 16 Prikaz Inspector-a za stablo kretanja 15
Slika 17 Prikaz Rigidboy 2D komponente na Playeru 16
Slika 18 Box Collider Player 16
Slika 19 Prikaz Box Collidera na Player objektu 17
Slika 20 Primjer koda skripte CharacterMovment 18
Slika 21 Nastavak skripte CharacterMovment() 19
Slika 22 Prikaz izgleda neprijatelja 20
Slika 23 Prikaz slika prije animacije 20
Slika 24 Prikaz Animator prozora za neprijatelja 21
Slika 25 Prikaz klase Enemy 22
Slika 26 Prikaz Inspectora za skriptu Demon 23
Slika 27 Prikaz dijela klase Demon 24
Slika 28 Primjer klase Demon 25
Slika 29 Prikaz izgleda pojedinih slika prije izrade mape 25
Slika 30 Raspored objekata 26
Slika 31 izgled Tilemap Renderera 26
Slika 32 Prikaz kolizija na mapi 27
Slika 33 Prikaz palete 27
Slika 34 Prikaz Box Collider-a 28
Slika 35 Prikaz klase RoomMove 29
Slika 36 Prikaz igre nakon dodavanja Point svjetla 30

IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam završni rad izradio samostalno, isključivo znanjem stečenim na studijima Sveučilišta u Dubrovniku, Služeći se navedenim izvorima podataka i uz stručno vodstvo mentora doc. dr.sc Krunoslava Žubrinića, kome se još jednom srdačno zahvaljujem.

Nevenko Prce