Izrada nivoa 2D računalne igre u programu Unity

Žugec, Luka

Sveučilište
Sjever

Završni rad br: 711/MM/2021

Izrada nivoa 2D računalne igre u programu Unity

Luka Žugec, 2914/366

Varaždin, srpanj 2021. godine

Sveučilište Sjever

Odjel za multimediju, oblikovanje i primjenu

Završni rad br: 711/MM/2021

Izrada nivoa 2D računalne igre u programu Unity

Student

Luka Žugec, 2914/366

Mentor

dr.sc. Andrija Bernik, pred.

Varaždin, srpanj 2021. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ preddiplomski stručni studij Multimedija, oblikovanje i primjena

PRISTUPNIK Luka Žugec | MATIČNI BROJ 2914/336

DATUM 21.12..2020 | KOLEGIJ Računalna animacija

NASLOV RADA Izrada nivoa 2D računalne igre u programu Unity

NASLOV RADA NA ENGL. JEZIKU Creating a level for 2D computer game in Unity

MENTOR dr. sc. Andrija Bernik | ZVANJE Docent

ČLANOVI POVJERENSTVA
1. mr.sc. Dragan Matković, v. pred. - predsjednik
2. pred. Nikolina Bolčević Horvatić, dipl.ing. - član
3. doc.dr.sc. Andrija Bernik - mentor
4. doc.art. Robert Geček - zamjenski član
5.

Zadatak završnog rada

BROJ 711/MM/2021

OPIS

2D video igre odnose se na igre u kojima se radnje događaju u 2D ravnini i obično se kreću vodoravno ili vertikalno. U njima se likovi i okruženja obično prikazuju u 2D-u. Njihove karakteristike u odnosu na 3D video igre su što koriste ravnu grafiku, nazvani sprite i nemaju trodimenzionalnu geometriju. Na zaslonu su prikazane kao ravne slike, a kamera koja prati igrača nema perspektivu. Najpoznatija vrsta 2D video igra su "platformer" igre koje karakterizira velika upotreba skakanja i penjanja za navigaciju u igračevom okruženju. Razine i okruženja uglavnom se sastoje od neravnih terena i raznih platforma postavljenih na različite visine što zahtjeva korištenje različitih sposobnosti samog lika igrača za prelazak.
U radu je potrebno:
Izraditi jedan nivo 2D računalne igre ("platformer" igre) koristeći program Unity
Opisati proces razvijanja 2D računalne igre

ZADATAK URUČEN 05.03.2021. | POTPIS MENTORA

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER

Predgovor

Tema ovog završnog rada proizašla je iz interesa i ljubavi u klasične 2D video igre koje svojom jednostavnošću mogu staviti fokus na izgled i osjećaj igre, stoga je želja za stvaranje vlastitog svijeta koji svoj karakter preko grafike i priče prenosi na druge ljude dovela do izrade jednog nivoa 2D računalne igre. Cilj kreiranja ovog završnog rada bio je usavršavanje znanja u stvaranja 2D igra programu Unity te osobno unapređenje i iskustvo u izradi piksel grafike.

Sažetak

2D video igre odnose se na igre u kojima se radnje događaju u 2D ravnini i obično se kreću vodoravno ili vertikalno. U njima se likovi i okruženja obično prikazuju u 2D-u. Njihove karakteristike na odnosu od 3D video igara su što koriste ravnu grafiku, nazvanu sprite, i nemaju trodimenzionalnu geometriju. Na zaslonu su prikazane kao ravne slike, a kamera koja prati igrača nema perspektivu. Najpoznatija vrsta 2D video igra su "platformer" igre koje karakterizira velika upotreba skakanja i penjanja za navigaciju u igračevom okruženju. Razine i okruženja uglavnom se sastoje od neravnih terena i raznih platforma postavljenih na različite visine što zahtjeva korištenje različitih sposobnosti samog lika igrača za prelazak.

Program korišten za izradu nivoa 2D video igre u radu je Unity. Unity je razvijen kako bi razvijanje 2D i 3D video igra bilo dostupno novim i neiskusnim programerima. Njegovo sučelje prilagođeno je kako bih se korisnici mogli jednostavno služiti svim njegovim alatima i funkcijama kod izrade projekata. U radu je opisana njegova povijest i razvoj, korisničko sučelje s fokusom na razvijanje 2D video igara, osnovna potrebna znanja o skriptiranju i glavne razlike odnosno opcije kod Unity 2D-a. Za praktičan dio opisan je proces izrade kompletnog nivoa 2D računalne igre, od izrade spriteova do izrade funkcionalnosti glavnog lika i neprijatelja.

Ključne riječi: Video igra, Unity, skriptiranje, sprite, 2D

Summary

2D video games refer to games in which actions take place in the 2D plane and usually move horizontally or vertically. In them, characters and environments are usually displayed in 2D. Their features compared to 3D video games are that they use flat graphics, called sprites, and do not have three-dimensional geometry. They are displayed on the screen as flat images, and the camera that follows the player has no perspective. The most famous type of 2D video games are platformers which are characterized by a large use of jumping and climbing to navigate the player's environment. Levels and environments mainly consist of uneven terrain and various platforms set at different heights which requires the use of different abilities of the player's own character to cross.

The program used to create one level of a 2D video game in this final work is Unity. Unity was developed to make the development of 2D and 3D video games available to new and inexperienced developers. Its interface has been customized so that users can easily use all its tools and functions when creating projects. The paper describes its history and development, the user interface with a focus on the development of 2D video games, the basic knowledge of scripting and the main differences and options in Unity 2D. For the practical part, the process of creating a complete level of 2D computer game is described, from making sprites to creating the functionality of the main character and the enemy.

Keywords: Video Game, Unity, Scripting, Sprite, 2D

Popis korištenih kratica

2D Two-dimensioal
Dvodimenzionalno

3D Three-dimensional
Trodimenzionalno

UI User Interface
Korisničko sučelje

Sadržaj

1. Uvod

2D „platformer“ video igre, varijacija su video igara na koje su vrlo popularan žanr nastao početkom 1980-ih, a sredinom 1990-ih na „platformer“ žanr počinje se primjenjivati 3D. Isprva su se platformeri odvijali na jednom zaslonu, gdje bi igrač morao prevladati određene prepreke. Pojavom Super Mario Bros-a, 2D platformer igre počele su uvoditi razine pomicanja kamere, gdje su se igrači pomicali slijeva udesno, pri čemu se lik koji se može igrati mogao gledati iz bočnog kuta. Lik se penje gore-dolje ljestvama ili skače s platforme na platformu, boreći se s neprijateljima i često ima sposobnost stjecanja moći ili oružja. Kasnije je taj pojam počeo opisivati igre u kojima je izazovno preskakanje s platforme na platformu, za razliku od pucanja, bilo glavni fokus igranja. Taj tip uključije igre poput Super Mario Bros. i Donkey Kong Country. Međutim, mnoge igre na platformi sadrže projektilsko oružje, uključujući franšize Mario i Castlevania. [1]

Unity je motor za razvijanje i izradu video igra koji pruža izvrsnu ulaznu točku za nove zainteresirane programere. On omogućuje brzo i učinkovito stvaranje objekata, uvoz vanjskih sredstava i povezivanje istih s kodom. Vizualno je vođen i izgrađen na principima s kojima je moguće učiniti sve jednostavnim povlačenje i ispuštanje elemenata. Te je zbog toga odabran kao primarni program za razvijanje video igre u ovom radu. [2]

Rad je podijeljen na dva dijela. U prvom dijelu opisane su sve osnove koje su vezane za rad u motoru Unity, njegova povijest razvijanja, općenite informacije o mogućnostima kod preuzimanja i instalacije programa, opisano je njegovo korisničko sučelje odnosno funkcije svih prozora koji se nalaze u njemu, proces i način skriptiranja te ključne razlike i elementi kod rada na 2D igrama.

Drugi dio rada prikazuje cjelokupni proces izrade jednog nivoa 2D računalne igre. Postavljanje projekta, izrade sprite grafika i setova pločica, postavljanje osnovnih funkcija kretnje glavnog lika, izrada animacija stanja, postavljanje cjelokupnog nivoa, skriptiranje i animiranje neprijateljskih likova, izrada sistema za interakciju s njima, dodavanje zvukova, izrada korisničkog sučelja te dodatnih scena i krajnji izgled nivoa.

2. Počeci i razvoj Unity -a

Unity je motor za izradu video igra koji je osnovana u Kopenhagenu od strane Nicholasa Francisa, Joachima Antea i Davida Helgasona. Njihov prvi susret desio se u svibnju 2002. godine na forumima OpenGL-a, gdje je Francis objavio poziv za suradnju u izradi Shader Compilera otvorenog izvora (grafičkog alata) koji bi koristili manji i novi programeri igara zasnovanih na Macu kao što je bio i on. Ante, koji je u tadašnje vrijeme bio srednjoškolac u Berlinu, pohvalio je Franjin fokus na grafiku i igru uz intuitivnim smislom za pozadinsku arhitekturu motora igra. U to vrijeme oboje su honorarno surađivali na razvijanju Shadera te su istovremeno samostalno radili na svojim vlastitim projektima.

Nakon osobnog sastanka i dijeljenja svojeg znanja odlučili su spojiti snage svojih motora za izradu igra. U žurbi spajanja svojih baza kodova, kampirali su u Helgasonovom stanu dok je on bio izvan grada. Njihov plan je bio pokrenuti studio za igre koji je zasnovan na robusnoj tehnološkoj infrastrukturi koja bi također mogla biti licencirana. Helgason je bio Francisov poznanik iz srednje škole s kojim je radio na raznim pothvatima za web razvoj. Helgason je napustio svoje školovanje na sveučilištu u Kopenhagenu te je počeo raditi kao slobodni web programer. Kroz nekoliko mjeseci Francisu je pružio pomoć gdje je mogao te se je na kraju odlučio pridružiti punim radnim vremenom njihovom projektu. Helgason, pošto je uvijek bio poslovno orijentiran, proglašen je titulom izvršnog direktora. Ante i Francis tek nakon dvije godine proširuju svoju titulu na suosnivača te dobivaju pravednu jednakost, što se tiče pozicija, s Helgasonom. Kroz vrijeme rotirali su ljude koji su besplatno radili i pomogli izrađivali prototipe širokog spektra ideja, što je dovelo do rezultata da motor može riješiti širok spektar slučajeva upotrebe.

Njihov sljedeći plan je bio smišljanje hit igre koja bi u sebi sadržala sve snage motora za igre te kako bi pokazala njegove najbolje prednosti za indie programere igara. Unatoč tome motor je imao svoju slabost, svaka nova ideja igre zahtjeva rekonstrukcijeu. Riješavanje toga problema omogućilo bi mnogo novih i krejativnijih ishoda. Sljedeće tri godine oslanjali su se na svoju uštеđevinu, ulogu oca Ante od 25000 eura i Helgasonovim honorarnim poslom u kafiću, te su se druge godine rada (2004.) imenovali Over The Edge Entertainment (OTEE).

Over The Edge Entertainment objavio je svoju prvu igru, GooBall, 2005. godine. GooBall je bila Mac video igra u kojoj je igrač imao ulogu vanzemaljca pod imenom Goober koji je bio nasukani na Zemlji. U priči vanzemaljac je postao poznat CIA-ju te su ga zaglavili u uređaj za održavanje života napravljen od protoplazme, odnosno loptu. Cilj video igre je bio proči nivo kroz razne prepreke te skupljati gemove. Video igra nije postigla komercionalni uspjek no troje osnivača s znanjem da njihov motor za igre, koji je stvoren da olakšava i pojednostavi razvoj igara, ima veliku vrijednost te su fokus tvrtke usmjerili kako bi stvorili game engine za druge programere. [3]

Slika 1: GooBall, 2005.

Nakon neuspjeha svoje video igre, tvrtka nastoji demokratizirati razvoj igara i učiniti razvoj 2D i 3D interaktivnog sardžaja pristupačnijim drugim programerima. 2006. godine na dodjeli nagrada Apple Design, Unity je dobio nagradu za najbolje korištenje grafike Mac OS X-a. 2007. godine tvtka mijenja ime u Unity Technologies i dolazi nova verzija Unity-a, Unity 2.0, koja je sadržavala otprilike 50 novih značajki za razvijanje video igra. Neke od glavnih promjena i dodataka bile su optimizirani terenski engine za detaljna 3D okruženja, dinamičke sjene u stvarnom vremenu, usmjerena svjetla i reflektori odnosno naprednija svjetla i drugo. Neke od značajki dodale su mogućnosti pomoću kojih bi programeri mogli lakše surađivati u svojem radu te mogućnosti izrade video igara za više igrača. 2008. godine tvrtka još više raste zbog pojave iPhonea, Unity Technologies proizveo jedan od prvih engina koji potpunosti podržava tu platformu. U to vrijeme industrija video igara bila je usredotočena na konzolama stoga je Unity, kada su iPhone i App Store bili objavljeni, bio u savrženoj poziciji da podržava programere koji žele stvarati mobilne igre. Dominacija njihovog motora na iPhoneu bila je neosporena nekoliko godina.

U rujnu 2010. godine izlazi nova verzija Unity-a, Unity 3.0, koja proširuje grafičke značajke engina na stolna računala i konzole za video igre. Unity 3.0 je uz Android podršku, sadržavao mogućnosti integracije alata Beast Lightmap (tvrtke Illuminate Labs), ugrađeni uređivač stabla, izvono prikazivanje fontova, automatsko UV mapiranje i audio filtere. Istraživanje Game Developer magazina u svibnju 2012. godine pokazalo je da je Unity glavni motor za video igre na mobilnim platformama te piše kako je cilj Unity-a biti motor za više platforma. U to vrijeme više od 1.3 milijuna programera koristi njegove alate za stvaranje grafike na njihovim video igrama za iOS, Android, konzole, računala te za igre bazirane za web.

Iste godine u studenomu izlazi Unity 4.0 u kojoj je dodana podrška za DirectX 11 i Adobe Flash, nove alate za animaciju zvane Mecanim te mogućnost pristupa „Linux preview“. Sljedeći ključni dio razvoja Unity-a dogodio se 2013. godine kada je Facebook integrirao komponet za razvoj softvera video igara koje koriste Unity game engine. Taj komponent je sadržavao alate koji su omogućili praćenje reklamnih kampanja i povezivanje kod kojeg su korisnici bili izravno povezani s njihovih Facebook računa na određene dijelove u igrama. Stoga je 2016. godine Facebook, s Unity-om, razvio novu platformu za igranje računalnih igara. Time je Facebook dobivao podršku za svoju platformu, a Unity programeri su bili u mogućnosti puno brže izvoziti i objavljivati svoje nove video igre.

Unity 5.0 bila je nova verzija Unity-a koja je izašla 2015. godine te s njom Unity Technologies dolaze sve bliže svojem cilju da Unity učine univerzalno dostupnim motor za izradu igra. U novoj verziji poboljšano je njegovo osvjetljenje i zvuk, te su putem WebGL-a Unity programeri mogli dodavati svoje igre u kompatibilne web preglednike bez da su njihovi igrači bili potrebni preuzimat ikakve dodatke. Neke od glavnih značajka nove verzije bili su globalno osvjetljenje u stvarnom vremenu, mogućnost predpregleda mapiranja svjetlosti, Unity Cloud (koji je omogućio lako dijeljenje svojih projekata preko drugih računala i ljudi), novi audio sustav i fizički motor Nividia PhysX 3.3. Unatoč velikog unapređenja neki igrači kritizirali su dostupnost i mogućnosti Unity engina zbog velikog broja brzo proizvedenih video igara koje su neiskusni programeri objavljivali na distribucijskoj platformi Steam. Odgovor koji je izvršni direktor John Riccitiello na to dao je da je to nuspojava uspjeha Unity-ja u demokritizaciji razvoja igara. [4] [5]

"Da imam svoj put, volio bih vidjeti 50 milijuna ljudi koji koriste Unity - iako ne mislim doći ćemo tamo uskoro. Volio bih vidjeti kako ih koriste srednjoškolci i studenti, ljudi izvan osnovne industrije. Mislim da je tužno što je većina ljudi potrošači tehnologije, a ne kreatori. Svijet je bolji mjesto kada ljudi znaju kako stvarati, a ne samo trošiti, i to je ono što pokušavamo promovirati. " [6]

2016. godine Unity Technologies najavljuju da će promjeniti sustan numeriranjea za verzije Unity-a na godinu izdanja verzije, stoga je Unity 5.6 sljedio Unity 2017. Alatima u novoj verziji dodan je mehanizam za grafičko prikazivanje u stvarnom vremenu prilikom radnje s njima, dodan je „color grading“ i „worldbuilding“ te uživa analitika operacija i izvješća o izvedbi video igra. Dodani su novi alati poput Timelinea, koji je omogućio povlačenje i ispuštanje animacija u igre, i Cinemachine odnosno pametnog sustava kamera za video igre. U Unity 2017.2 bili su integrirani Autodeskovi 3DS Max i Maya alati za pojednostavljeni postupak iterakcije s elementima u procesu izrade video igra.

Unity 2018 sadržavao je „Scriptable Render Pipeline“ koji je omogućio programerima stvaranje vrhunske grafike. To je uključivalo „High-Definition Rendering Pipeline“ za konzole i računala te istovremeno „Lightweight Rendering Pipeline“ za mobilne uređaje, virtualnu stvarnost, proširenu stvarnost i mješovitu stvarnost. U Unity 2018 su bili također dodani neki alati koji su omogućavali strojno učenje poput „Imitation Learning“, podršku za Magic Leap i

predloške koji su bili namjenjeni za nove programere. U lipnju 2020., Unity je predstavio Mixed and Augmented Reality Studio, koji programerima pruža dodatne funkcije za generiranje aplikacija proširene stvarnosti (AR). [4] [5]

3. Uvod u Unity-u

Unity je motor za razvijanje i izradu video igra, brzo i učinkovito omogućuje stvaranje objekata, uvoz vanjskih sredstava i povezivanje istih s kodom. Vizualno je vođen i izgrađen na principima s kojima je moguće učiniti sve jednostavnim povlačenje i ispuštanje elemenata. Unity također sadrži sposobnosti kao što su integrirano okruženje skriptiranja, ugrađene mrežne mogućnosti i sposobnost izgradnje i implementacije video igra za više platformi.

Besplatna verzija Unity omogućuje ljudima eksperimentiranje, učenje, razvoj i prodaju igre prije nego što uplate bilo koji njihov novac. Svojom ogromnom korisničkom zajednicom, Unity uklanja elitističko razdvajanje između programera, umjetnika i dizajnera igara što je tipično za skupocjene motore video igara. Omogućuje svima pružati prvi korak u oživljavanju njihovih ideja.

Unity ima dvije vrste licenci: besplatnu i Pro. Kod prvog preuzimanja i instalacije Unity-a, korisnici dobivaju mogućnost korištenja besplatne Pro verzije na 30 dana. Uz nekoliko iznimaka, većina Pro značajki usmjerena je na optimizaciju samog projekata stoga nije nimalo nužna za nove korisnike.

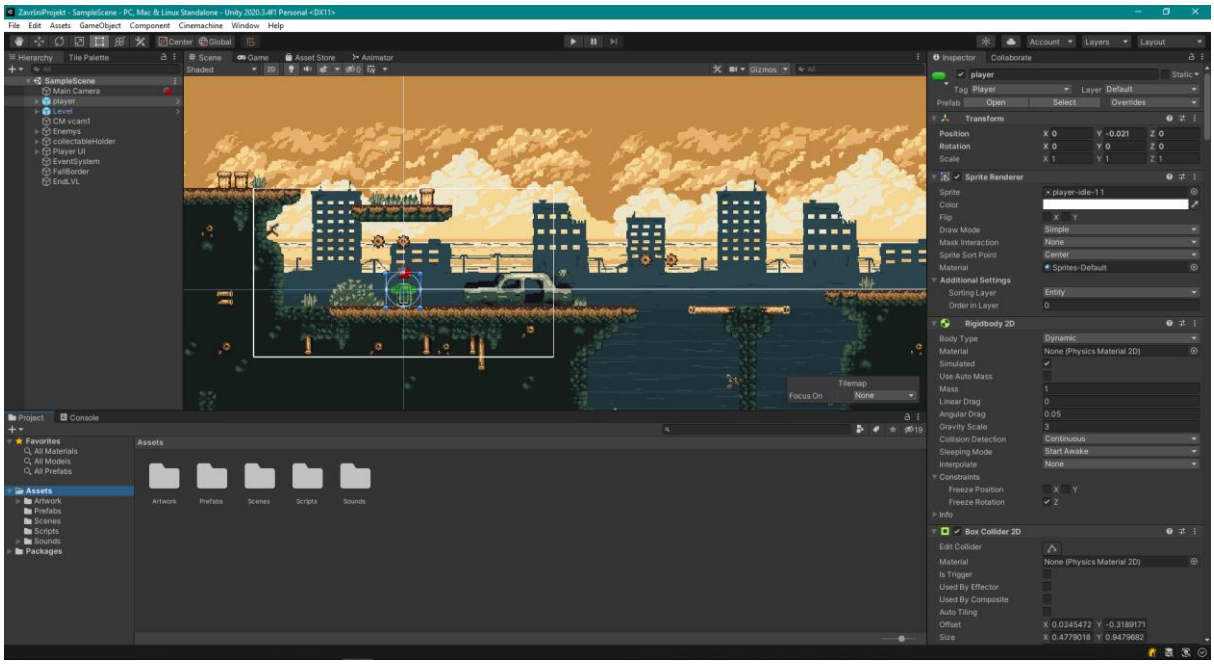
Tijekom instalacije potrebno je izraditi Unity korisnički račun koji omogućuje korisnicima pristup Unity Forumu, Unity Answers-ima i Unity Asset Store-u. Forum pruža mjesto za rasprava o svim stvarima Unity-a. Answers je mjesto na kojem je moguće potražiti brza rješenja za određene upite, i Unity Asset Store je mjesto gdje se nalaze sredstva svih vrsta (3D modeli, animirani likovi, teksture, skripte, cjelovita okruženja igara itd.) koja korisnicima mogu pomoći u izradi projekta. [2] [7] [8]

Slika 2: Unity logotip

4. Korisničko sučelje [9] [10]

Dizajn korisničkog sučelja najnovije verzije Unity-a, Unity 2020, modernog je izgleda te je univerzalnog dizajna kako bi imao što ugodniji pristup svojim korisnicima i ljudima koji žele početi s programiranjem i stvaranjem svojih video igara.

Samo sučelje sastoji se on nekoliko prozora od kojih svaki ima svoju svrhu prilikom izrade video igara. Korisnici programa imaju mogućnost mijenjanja pozicija, veličine i postave tih prozora no prilikom prvog otvaranja Unity-a oni su postavljeni na najoptimalniji način. Na gornjem lijevom vrhu nalaze se klasični izbornici koji sadrže razne opcije programa. [11] [12]



Slika 3: Korisničko sučelje Unity-a

4.1. Alatna traka (engl. The Toolbar)

Alatna traka omogućuje brzi pristup svim najvažnijim značajkama za rad unutar programa. Na lijevoj strani alatne trake nalaze se osnovni alati za manipulaciju prozora scene i objekta (engl. GameObjects) unutar njega. Prvi alat u toj skupini je Hand Tool koji služi za jednostavno micanje po samoj sceni, dok alati za premještanje (engl. Move Tool), rotiranje (engl. Rotate Tool), skaliranje (engl. Scale Tool), preusmjeravanje (engl. Rect Transform Tool) i pretvaranje (engl. Transform Tool) omogućuju uređivanje pojedinih objekta unutar scene.

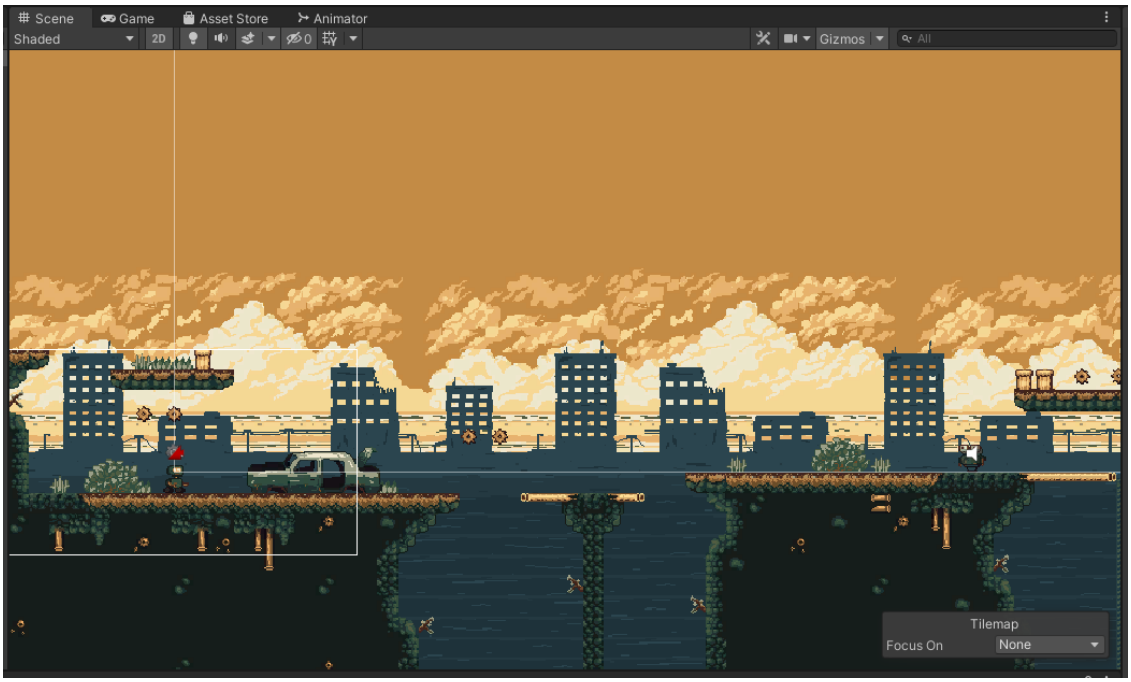
U središtu trake nalaze se kontrole za pokretanje, pauzu koje služe za pregled odnosno prozor video igre (engl. Game view). Gumbi s desne strane omogućuju pristup Unity Collavorate i Unity Cloud servisima te vlastitom Unity računu. Nakon njih slijede izbornik

vidljivosti slojeva u sceni i na kraju izbornik izgleda uređivača koji već pruža nekoliko različitih izgleda za prozore uređivača te omogućuje korisnicima spremanje vlastitih prilagođenih izgleda.

Slika 4: Alatna traka

4.2. Prozor Scene (engl. The Scene view)

Prozor scene je interaktivni pogled na svijet koji video igre koju neko stvara unutar Unity-a. U prozoru scene korisnik može odabirati i postavljati sve elemente video igre, poput likova, pozadinskih elemenata, kamera, svijetla te ostalih vrsta objekta. Odabir, manipulacija i modifikacija objekta u prozoru Scene su jedna od prvih vještina s kojima se novi korisnici Unity-a susreću. [7]



Slika 5: Prozor Scene

4.3. Prozor Igre (engl. The Game view)

Prozor igre prikazuje aplikaciju odnosno video igru s mjesta na kojemu se nalazi kamera postavljena u sceni. On pokazuje konačan izgled neke izrađene video igre unutar Unity-a te se uglavnom koristi za testiranje tokom rada. Kako bi se na prozoru igre bilo što prikazalo u sceni je potrebno koristiti jednu ili više kamera za kontrolu onog što igrač vidi. Testiranje odnosno pokretanje video igre vrši se pomoću središnjih gumba koji se nalaze na alatnoj traci. Prilikom pokretanja igre korisnik ulazi u Play Mode te sve promjene koje se naprave, u tome stanju, su privremene i resetiraju se kod izlaska. To je dobar način za direktno testiranje postavki nekih parametra na objektima. Korisničko sučelje je u tome stanju potamnjeno kako bi korisnike podsjetilo da će sve što rade biti resetirano. [7]

Slika 6: Prozor igre

4.3.1. Kontrolna traka prozora igre

Kontrolna traka sadrži nekoliko opcija za postavljanje izgleda i funkcije prozora igre. S lijeve strane nalazi se Display gumb odnosno izbornik koji služi na mijenjanje kamere koju prozor igre prikazuje. Ova funkcija je korisna i nužna ako se u sceni koristi više od jedne kamere, u tome slučaju sve kamere u sceni nalaze se u Display izborniku.

Desno od Display izbornika nalazi se izbornika za određivanje aspekta. On služi kako bi mogli testirati izgled video igre na ekranima različitih omjera. Vrlo je koristan kod točnog postavljanja korisničkih sučelja u video igrama.

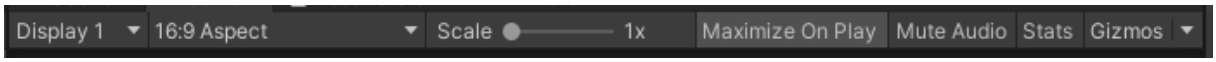
Klizač za skaliranje (engl. Scale slider) uglavnom služi za povećanje i detaljniji pregled nekih područja zaslona igre te obratno.

Desno od klizaća nalazi se gumb „Maximize on Play“ koji služi da, prilikom testiranja, prozor igre bude maksimiziran.

Gumb „Mute Audio“ služi isključivanje zvuka prilikom Play Mode-a.

Gumb „Stats“ služi za prikaz prozora statistike koji sadrži podatke o prikazivanju grafike i zvuku video igre. Ova funkcija je vrlo korisna za praćenje izvedbe video igre unutar Play Mode-a.

Izbornik „Gizmos“ služi za kontrolu vidljivosti Gizmosa. Gizmosi su grafika koja okružuje svaki objekt u sceni te oni mogu određivati neke od njihovih parametara. U izborniku moguće je odabrati vrste Gizmosa koje će biti vidljive u Play Mode-u.



Slika 7: Kontrolna traka prozora igre

4.4. Prozor projekta (engl. The Poject window)

Prozor Project prikazuje sve datoteke povezane s projektom i glavni je način na kojim se mogu pronaći i navigirati svi radni materijali i druge datoteke projekta. Kod pokretanja novog projekta prema zadanim postavkama, ovaj je prozor otvoren. Kao sve ostale prozore moguće ga je micati i prilagođavati na bilo koji način te je moguće mijenjat sami izgled njega i sadržaja u njemu.

Lijevi dio preglednika prikazuje strukturu mapa Projekta kao hijerarhijski popis. Kod odabira mape s tog popisa, Unity prikazuje njezin sadržaj u prostoru s desne strane. Moguće je kliknuti mali trokut da bi se mapa proširila ili sažela, prikazujući sve ugniježđene mape koje ona sadrži.

Pojedinačni radni materijali prikazana su na desnoj strani prozora kao ikone koje označavaju njihov tip (na primjer, skripta, materijal, podmapa). Podešavanje veličine ikona moguće je korištenjem klizača na dnu prozora. Prostor s lijeve strane klizača prikazuje trenutno odabranu stavku, uključujući puni put do stavke.

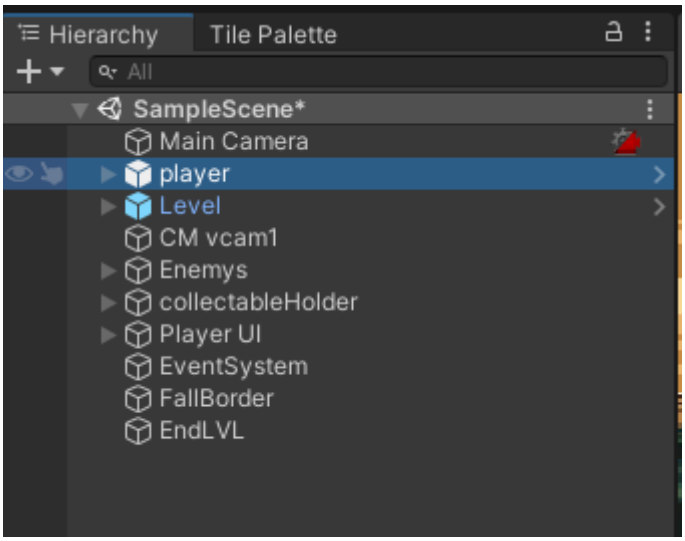
Slika 8: Prozor projekta

4.5. Prozor Hijerarhije (engl. The Hierarchy window)

U prozoru Hijerarhije prikazan je svaki objekt koji se nalazi u sceni, to mogu biti modeli, kamere ili prijašnje definirani objekti (engl. Prefabs). Koristi se za sortiranje i grupiranje objekata koji su korišteni u Sceni. Dodavanjem ili uklanjanjem objekta u sceni, također dodaje ili uklanja te objekte u prozoru Hijerarhije.

Prozor Hijerarhije može sadržavati i druge scene pri čemu svaka scena sadrži svoje objekte igre.

Prilikom rada u Unity-u, u prozoru Hijerarhije, također je moguće upravljati vidljivosti i dohvatljivosti elemenata koje se nalaze na sceni. Te opcije nalaze se sa lijeve strane elemenata u prozoru Hijerarhije.



Slika 9: Prozor Hijerarhije

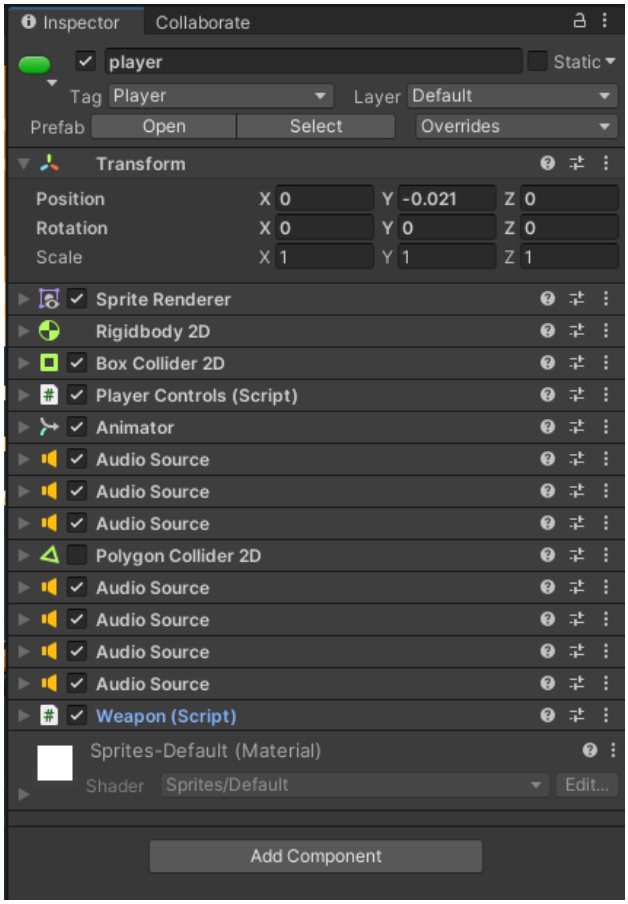
4.5.1. Roditeljstvo (engl. Parenting) unutar Prozora Hijerarhije

Unity koristi koncept hijerarhije roditelj-dijete ili roditeljstvo za grupiranje objekta. To znači da neki objekt može sadržavati druge objekte koji tada poprimaju odnosno nasljeđuju njegova svojstva. objekti se mogu povezati kako bi lakše mogli premještati, skalirati ili transformirati skupine objekata. To znači da kod premještanja objekta najviše razine, ili objekta koji je „roditelj“, premještaju se i svi objekti koji su njihova „djeca“ odnosno niže razine. [7]

Slika 10: Prikaz primjenjene hijerarhije roditelj-dijete

4.6. Prozor Inspektor (engl. The Inspector window)

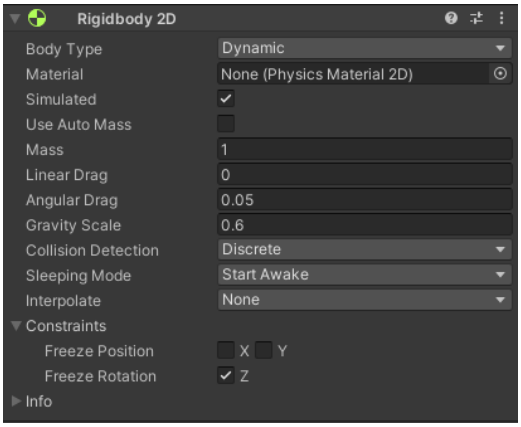
Prozor inspektora omogućuje pregled i uređivanje svih svojstava trenutno odabranog objekta. Budući da različite vrste objekta imaju različite skupove svojstava, izgled i sadržaj Inspektora prozor se mijenja svaki put kad odaberete drugi objekt. Osim objekta u inspektoru moguće je uređivati svojstva gotovo svega u Unity-u, Unity komponenta, sredstva iz prozora projekta, materijala te postavka unutar samog editora. [7]



Slika 11: Prozor Inspektor

4.6.1. Pregled GameObjekta

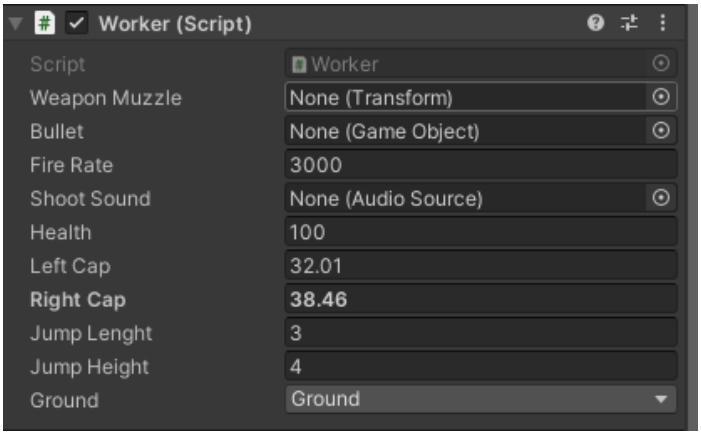
Kod odabira objekta, u prozoru Hijerarhije ili Scene, Inspektor prikazuje svojstva svih njegovih komponenta i materijala. U prozoru Inspektor moguće je urediti sva njegova svojstva komponenta te dodavati nove komponente.



Slika 12: Prikaz svojstva jednog komponenta objekta igre

4.6.2. Pregled prilagođenih komponenta skripte

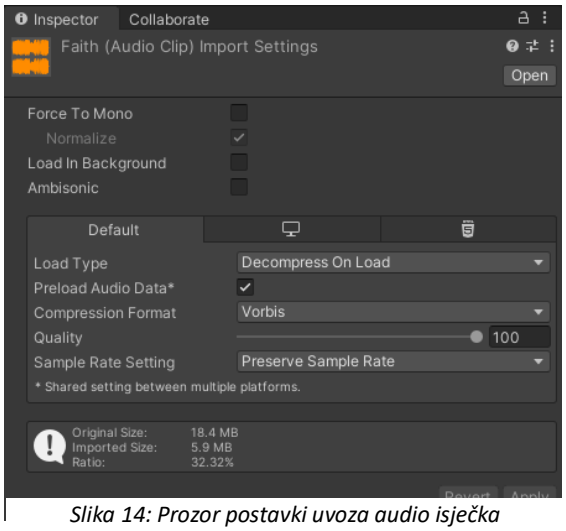
U slučaju kada objekti sadrže prilagođene komponente skripte, Inspektor prikazuje javne varijable koje se nalaze u skripti. Varijable skripte moguće je uređivati na isti način kao i bilo koja druga svojstva, to znači da je u skriptama moguće postaviti parametre i zadane vrijednosti bez mijenjanja koda. Ti parametri i vrijednosti i dalje mogu biti predefinirani unutar koda te su u tome slučaju te vrijednosti također ispisane u Inspektoru.



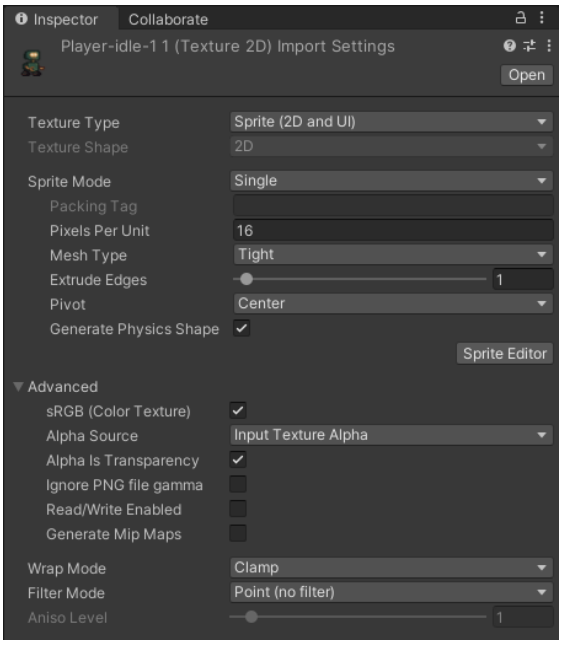
Slika 13: Prikaz prilagođene komponente skripte koja sadrži javne varijable

4.6.3. Pregled sredstva iz prozora projekta (engl. Assets)

Kada odaberete sredstvo (na primjer, iz prozora Project), Inspektor prikazuje postavke koje kontroliraju način na koji Unity uvozi i koristi sredstvo u vrijeme izvođenja. Svaka vrsta sredstva ima svoje postavke. Primjeri postavki uvoza sredstva koja se uređuju u prozoru Inspektor uključuju prozor postavki uvoza modela, prozor postavki uvoza audio isječaka i prozor postavke uvoza teksture.



Slika 14: Prozor postavki uvoza audio isječka



Slika 15: Prozor postavki uvoza sprite grafike (modela)

4.7. Traka stanja (engl. The status bar)

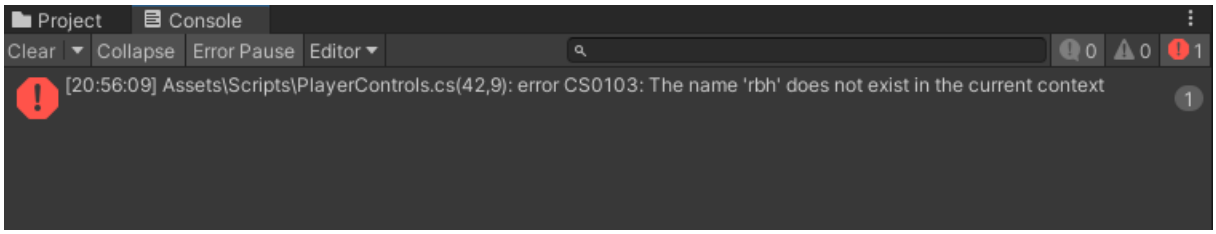
Traka stanja pruža obavijesti o raznim Unity procesima i pruža brzi pristup alatima i postavkama povezanim s njima. Nalazi na dnu cjelokupnog prozora te se ne može premjestiti ili preurediti. Traka stanja redom prikazuje, najnoviju poruku zabilježenu u prozoru konzole, globalnu traku napretka (engl. A global progress bar) za razne asinkrone zadatke, trenutni način optimizacije koda (klikom ikone moguće je prebaciti način sa debug-a na release), status poslužitelja pred memorije, status automatskog generiranja osvjetljenja za globalno osvjetljenje i pokazatelj aktivnosti (spinner) koji pokazuje kada Unity sastavlja C # skripte ili izvodi asinkrone zadatke.

Slika 16: Traka stanja (engl. The status bar)

4.8. Prozor konzole (engl. The Console Window)

Prozor konzole prikazuje pogreške, upozorenja i druge poruke koje generira Unity. Također je moguće prikazati vlastite poruke pomoću klase Debug kroz sami kod neke skripte. Sve što je ispisano u prozoru konzole također je zapisano u dnevniku datoteka (engl. Log File). Prozor konzole moguće je otvoriti kroz glavni izbornik Unity-a, Prozor > Općenito > Konzola

(engl. Window > General > Console). Alatnom trakom konzole, koja se nalazi na gornjem dijelu prozora odnosno iznad poruka, moguće je upravljanje načinom prikazivanja poruka te pretraživanje i filtriranje poruka.

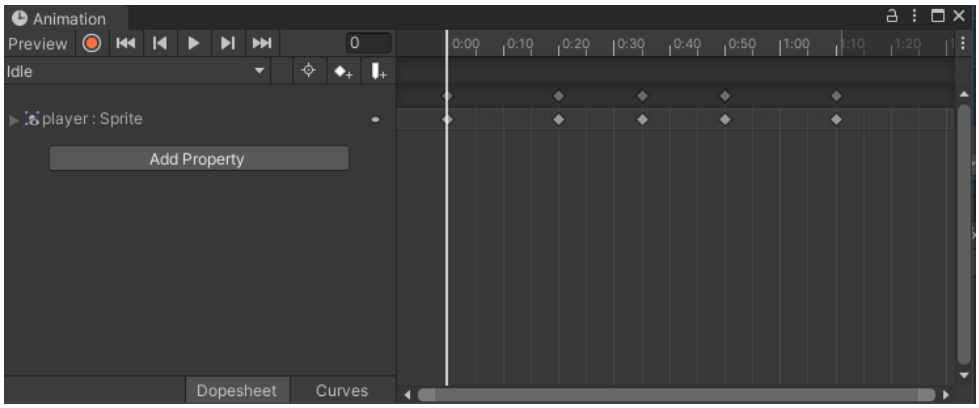


Slika 17: : Prozor konzole (engl. The Console Window)

4.9. Prozor za Animaciju (engl. The Animation Window)

Prozor za animaciju omogućuje stvaranje i modificiranje isječka za animaciju unutar samog Unity-a. Dizajniran je da djeluje kao moćna i izravna alternativa vanjskim programima za 3D animaciju dok za 2D djeluje na isti način kao svaki tip kadar po kadar (engl. Frame by frame) animacije u nekoj vremenskoj traci . Uz animiranje pokreta ili sprite grafike, moguća je i animacija varijabli materijala i komponenta te dodavanje animacijskih eventa na animirane isječke, odnosno funkcija koje su pozvane na određenim točkama na vremenskoj traci.

Prozor za animaciju sastoji se od 3 glavna dijela, na lijevoj strani nalazi se popis animiranih svojstva za i kontrole za reprodukciju i navigaciju kadrova, a desno vremenska traka.



Slika 18: Prozor za Animaciju (engl. The Animation Window)

Popis animiranih svojstva samostalno se ispunjavaju tokom izrade animacija, kod kompleksnijih 3D animacija, za razliku od 2D animacija, ta lista postane velika. Ako animacija kontrolira više podređenih objekata, popis će također sadržavati hijerarhijske pod liste animiranih svojstava svakog podređenog objekta.

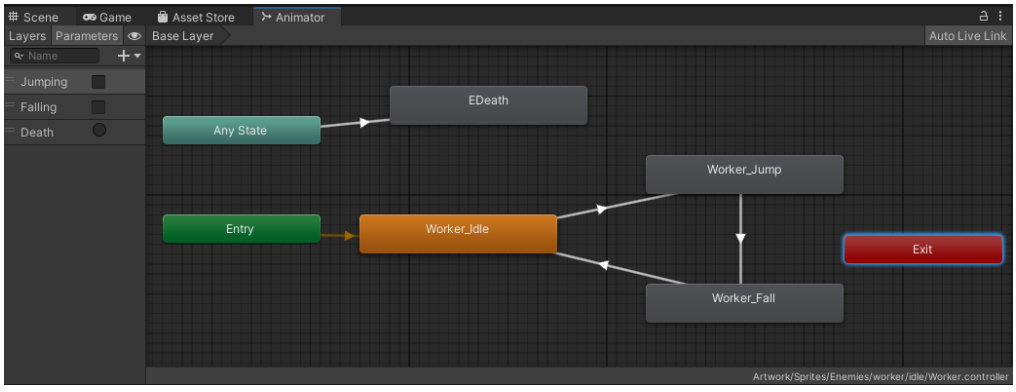
Na vremenskoj traci nalaze se ključni okviri za svako animirano svojstvo. Ona se može prikazati na dva načina, Dopesheet i krivulje (engl. Curves). Izmjena između ta dva načina prikazivanja moguća je klikom na isto nazvane gumbe na dnu područja popisa animiranih svojstva. Dopesheet prikaz omogućuje prikaz redoslijeda ključnih okvira svakog svojstva u pojedinačnom vodoravnom zapisu, što omogućuje jednostavniji pregled razmaka vremena između ključnih okvira za više svojstva ili objekta. Način prikaza krivulja prikazuje graf koji se može mijenjati te sadrži prikaz kako se vrijednosti svakog animiranog svojstva mijenjaju tijekom vremena animacije. Sva odabrana svojstva međusobno su prekrivena unutar istog prikaza grafikona što omogućuje veliku kontrolu nad pregledavanjem i uređivanjem vrijednosti i načinom njihove interpolacije.

4.10. Prozor Animator (engl. The Animator Window)

Prozor Animatora omogućuje stvaranje, prikaz i izmjenu sredstava za Kontrolor animatora (engl. The Animator Controller).

Kontrolor animatora omogućuje organiziranje i održavanje niza animacija za lika ili neki drugo animirani objekt igre. Kontroler ima reference na animacijske isječke koji se koriste unutar njega te upravlja raznim animacijskim stanjima i prijelazima između njih, pomoću mašine stanja (engl. State Machine). Mašina stanja je neka vrsta dijagrama toka odnosno jednostavno napisan program s vizualnim programskim jezikom unutar Unity-a.

Sastoji se od dva glavna dijela, glavno mrežno područje rasporeda stanja i s lijeve strane dio sa slojevima i parametrima. Prikaz parametara omogućuje vam stvaranje, prikaz i uređivanje parametara Kontrolera animatora. To su varijable koje su definirane od strane korisnika i koje određuju kada će koja animacija biti učinjena, odnosno tu informaciju šalje u mašinu stanja koja je prema tome definirana. Glavni dio s desne strane je područje u kojemu se raspoređuju, stvaraju, uređivanju i povezuju stanja.



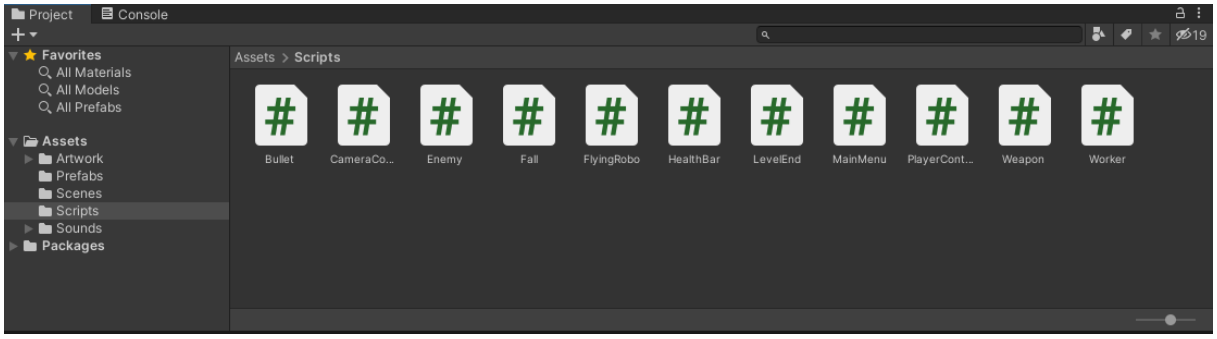
Slika 19: Prozor Animatora (engl. The Animator Window)

5. Kodiranje u Unity-u

Kodiranje, odnosno skriptiranje, je važan dio svih aplikacija koje se izrađuju u programu Unity. Većina aplikacija odnosno video igra trebaju imati skripte koje odgovaraju na inpute igrača i razne događaje koji se trebaju dogoditi u nekom zadanom vremenu. Skriptiranje u Unity-u razlikuje se od tipičnog kodiranja nekog programa, za razliku od kodiranja gdje je potrebno izraditi kod koji pokreće samu aplikaciju, jer Unity-u nije potreban kod koji bi pokretao video igru. Fokus skriptiranja stavlja se na igranje igre i njenu interakciju s igračem. Neki primjeri naziva i funkcija skripta koje se nalaze u većini video igra su PlayerController (omogućuje igraću kretanje glavnog lika), EnemyAI (Definira kretanje neprijateljskih likova u igri) i LevelEnd (definira mjesto na kojemu završava neki nivo igre). Svaka od izrađenih skripti namijenjena je kako bi učinila jednu stvar, takav tip skriptiranja najbolja je praksa za arhitekturu projekata u Unity-u.

Kako bi neka skripta bila pozvana kod pokretanja video igre, mora biti priložena objektu koji se nalazi na sceni. Način na koji se skripte dodaju objektima je isti kao i kod dodavanja bilo kojeg drugog komponenta kroz prozor Inspektora. Skripte su napisane na posebnom jeziku koji Unity može razumjeti. I putem ovog jezika programeri mogu razgovarati s motorom i davati mu svoje upute. Sve izrađene skripte nalaze se u prozoru Projekta te se kroz njega ili kroz Inspektor (na dodijeljenom objektu) otvaraju dvostrukim klikom.

Kroz duže vrijeme primarni tekstualni editor koji je bio korišten za skriptiranje u Uniy-u bio je MonoDevelop, a nakon verzije 2018.1 Unity dodaje mogućnost korištenja Visual Studia i nekih ostalih tekstualnih editora. Korištenje njih može nam pomoći da dovršimo pisanje koda te obavještava programere ako se u kodu nalaze greške. [9] [13] [14]

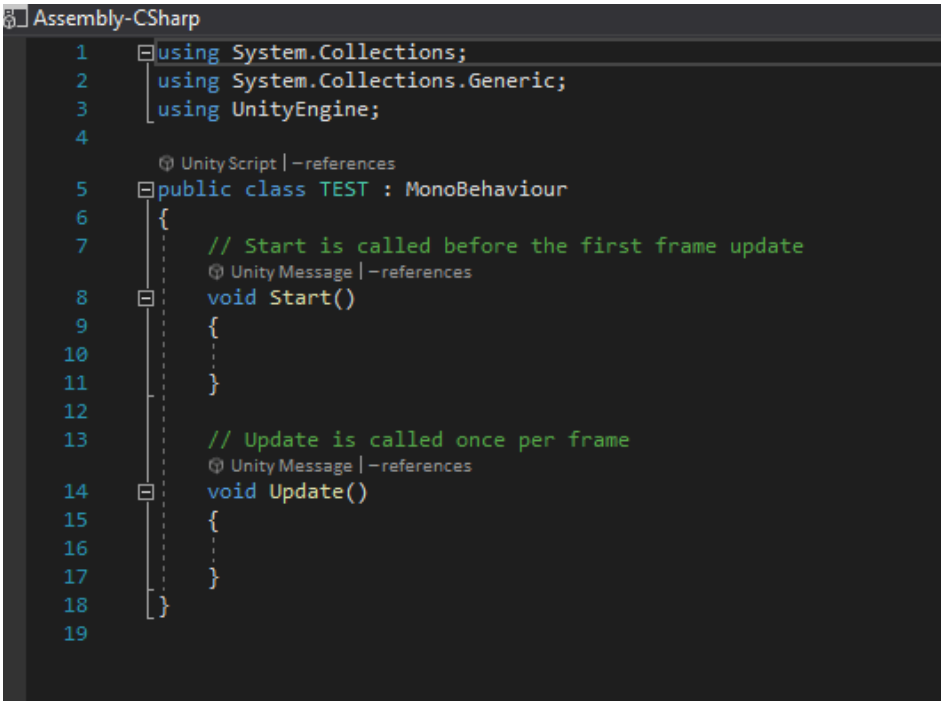


Slika 20: Prikaz svih korištenih skripta u jednom projektu.

5.1. Programski jezik C#

C# je objektno orijentirani i tipski siguran programski jezik koji omogućuje programerima izgradnju mnogih vrsta sigurnih robusnih aplikacija koje se izvode u .NET ekosustavu. C# vuče korijene iz obitelji C jezika i te je lagan za korištenje programerima poznatima s C, C++, Java i JavaScript jezicima. Razvijen je 2002. godine od strane Microsofta te je glavni jezik za Unity od 2005. godine.

Svi jezici s kojima Unity operira su objektno orijentirani skriptni jezici. Kao i svaki jezik, i skriptni jezici imaju sintaksu ili dijelove govora, a primarni dijelovi nazivaju se varijablama, funkcijama i klasama. [13]



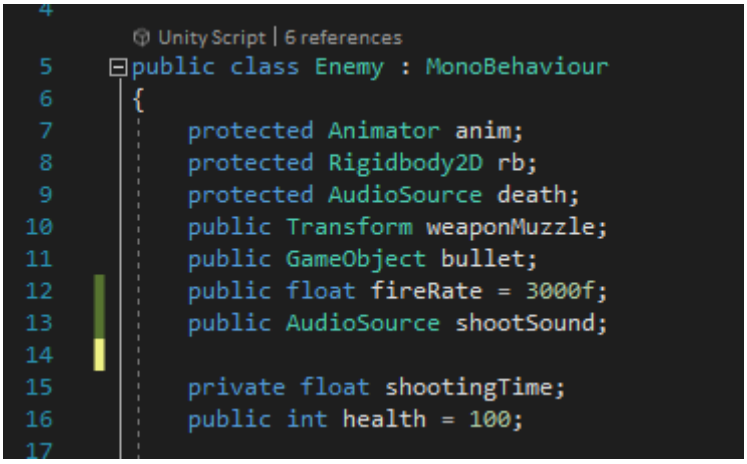
Slika 21: Prikaz novo izrađene skripte u editoru Visual Studio

5.1.1. Varijable (engl. Variables)

U Unity-u sve skripte počinju s postavljanjem elemenata koji će u skripti biti potrebni i s kojima će se manipulirati u njima, a to je obično deklariranjem varijabli. Kod deklariranja varijabli postoje nekoliko vrsti koje ovise o njihovoj vidljivosti, no dvije najvažnije vrste su javne (engl. Public) i privatne (engl. Private). U slučaju deklariranja javne varijable, ona će biti vidljiva u Inspektoru kod komponenta te skripte. Njezina vrijednost tada je dostupna drugim skriptama i drugim klasama, a može se mijenjati u Inspektoru iz Unity-a, to znači da drugi ljudi mogu pristupiti i promijeniti njezinu vrijednost. Dok privatna varijabla nije vidljiva u inspektoru te joj se jedino može pristupiti kroz određenu skriptu u kojoj se nalazi.

Drugi bitan dio varijabli je tip. Tip definira kakvu vrijednost ima varijabla koja se nalazi u memoriji, na primjer to mogu biti broj, tekst ili složenije vrste poput: Transform, Light, AudioSource, GameObject, Rigidbody2D itd. Unity mora znati vrstu predmeta koju referiramo kako bi znao rukovati s njim, stoga tipovi varijabli mogu biti komponenti.

Kod imenovanja varijabla bitno je držati se pravila kako bi se izbjegle bilo kakva nepotrebne poteškoće. Glavna stvar kod imenovanja varijabla je da ne smiju započeti brojem te ne smiju sadržavati znakove. Definirani stil pisanja varijabla u C# je započinjanje malim slovom te dodavanje riječi, bez razmaka, počevši s velikim slovom, kao na primjer „weaponMuzzle“. [15]



Slika 22: Prikaz deklaracije varijabla skripte Enemy

5.1.2. Funkcije (engl. Functions)

Funkcije su dijelovi koda pomoću kojega se manipuliraju varijable. U Unity-u postoji nekoliko funkcija koje se automatski pokreću, bez da ih je potrebno pozvati. U te funkcije spadaju Awake(), Start(), Update(), FixedUpdate(), LateUpdate.

Funkcije Awake() i Start() funkcionalno su vrlo slične, prilikom pokretanja igre one su pozvane samo jednom ukoliko je objekt na kojem se njihova skripta nalazi aktivan. No njihova razlika je da je Awake() funkcija pozvana i u slučaju kada je skripta na objektu deaktivirana s potvrdnim okvirom.

Update() se pozva jednom po okviru (engl. Frame) te se u nju stavlja kod koji definira logiku koja se konstantno izvodi, poput animacija, AI-a i ostalih dijelova igre koji se moraju stalno ažurirati. LateUpdate() je funkcije identična funkciji Update(), no ona se ažurira na kraju svakog okvira (engl. Frame).

Kod skriptiranja u Unity-u bitna je preglednost, stoga je vlastite funkcije, koje definiraju neku radnju ili proces za objekt, bitno pisati posebno. Te funkcije se tada pozivaju u glavne predefinirane funkcije Unity-a, odnosno najčešće u Start() i Update() funkcije. Njih je također

moguće pozivati na duge načine kroz sami Unity i skripte. Pravilno napisana funkcija sastoji se na početku od vrste funkcije, nakon čega slijedi ime funkcije, a zatim parametar u zagradama (ako je potreban). Naziv funkcije uvijek počinje velikim slovom, a tijelo u kojemu se nalazi sama funkcija funkcije ide između vitičastih zagrada.

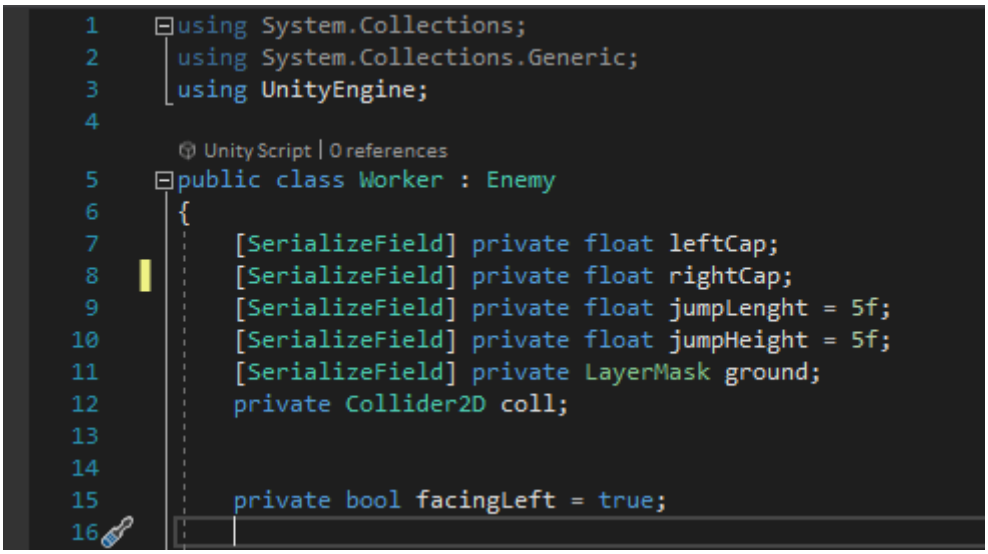
Funkcije mogu izvršiti neke izračune, a zatim vratiti vrijednosti. Od njih se također može tražiti da nešto učine, obradi informacije te zatim vrati odgovor. U slučaju kada ne želimo da one vrate neki dogovor, nego da čine neku radnju, koristi se tip „void“. Kod izrade funkcija za video igru „void“ je najčešći tip koji se koristi, pošto je većinom od objekta traženo da rade neke konstantne radnje, bez vraćanja ikakvih informacija. [15]

```
1 reference
126 private void Movement()
127 {
128     float hDirection = Input.GetAxis("Horizontal");
129     //left
130     if (hDirection < 0)
131     {
132         rb.velocity = new Vector2(-speed, rb.velocity.y);
133
134
135     }
136     //right
137     else if (hDirection > 0)
138     {
139         rb.velocity = new Vector2(speed, rb.velocity.y);
140
141
142     }
143     if ((rb.velocity.x > 0.1 && !facingRight) || (rb.velocity.x < -0.1 && facingRight))
144     {
145         flip();
146     }
147     //jump
148     if (Input.GetButtonDown("Jump") && Mathf.Abs(rb.velocity.y) < .001f)
149     {
150         Jump();
151     }
152 }
    1 reference
153 void flip()...
158
    2 references
159 private void Jump()...
164
```

Slika 23: Primjer funkcije zaslužene za kontrolu glavnog lika

5.1.3. Klase (engl. Classes)

Klase su kolekcije svih varijabla i funkcija, odnosno jedna skripta se može smatrati kao jedna klasa. Bitno je da je ime klase isto kao i ime C# skripte jer suprotno skripta ne bi radila. A zatim da bi klasa bila spojena na objekt, mora proizaći iz druge klase zvane MonoBehaviour koja se automatski zapiše u kod, kada prvi put kreirate skriptu. Klase također mogu biti javne ili privatne te se mogu spajati na vlastito kreirane klase (odnosno skripte) koje su tada spojene s MonoBehaviour klasom. [15]



Slika 24: Primjer klase "Worker" koja proizlazi iz klase "Enemy"

6. Unity 2D

Iako je Unity većinom fokusiran na izradu i razvijanje mogućnosti kod 3D video igara, jednako je pogodan za stvaranje igra u 2D prostoru. Kod izrade novog projekta u Unity-u postoje dvije mogućnosti, stvaranje projekta u 2D ili 3D načinu. Kod početka većine projekata unaprijed je poznato na koji način će video igra odnosno projekt biti postavljen. Izbor između pokretanja u 2D ili 3D načinu određuje neke postavke za Unity uređivač. Neke od glavnih razlika su hoće li se slike u projekt uvesti kao teksture ili spriteovi. Mijenjanje između 2D ili 3D načina moguće je u bilo kojem trenutku, bez obzira na način postavki kod kreiranja projekta.

Najuočljivija značajka, u projektu otvorenom u 2D načinu, je gumb 2D načina prikaza na alatnoj traci prozora scene. Kad je omogućen 2D način rada, ortografski pogled odnosno pogled bez perspektive je postavljen u sceni. Kamera je postavljena da gleda duž Z osi dok je Y os ona koja se povećava prema gore. To omogućuje vizualizaciju scene i lako postavljanje 2D objekata.

Tip igara u kojima se Unity projekt otvara u 2D načinu su 2D igre koje koriste ravnu grafiku. Komponenti 2D ravne grafike nazivaju se spriteovi te ne sadržavaju trodimenzionalnu geometriju. Na ekranima prikazani su kao ravne slike, a kamera igre nema perspektivnu. [9] [16]

6.2. Spriteovi (engl. Sprites)

Grafički objekti u 2D video igrama nazivaju se spriteovi. Oni su u osnovi samo standardne teksture, ali u Unity-u postoje posebne tehnike za kombiniranje i upravljanje sprite teksturama koje omogućuju njihovu učinkovitost i praktičnost tijekom razvoja video igre.

Sprite grafika u svijetu video igra vrlo je popularna jer su počeci izrade video igra počeli s najjednostavnijim grafikama koje su sadržavale spriteove. Korištenje sprite grafike omogućuje manjim programerima izradu video igra koje, sa svojim stilom, prosljeđuju svoj karakter na druge. Izgled sprite grafike ovisi o onome tko ju radi, pošto ona ne sadrži nikakva određena pravila po kojima bi se trebala raditi. Neke od najpopularnijih i najjednostavnijih tipa sprite grafije je piksel grafika. Razlog njezine popularnosti je mogućnost lake izrade i učenja te krajnji izgled. Većina novih programera video igara koji žele izraditi svoju grafiku za 2D igru počinje s njom. [17]

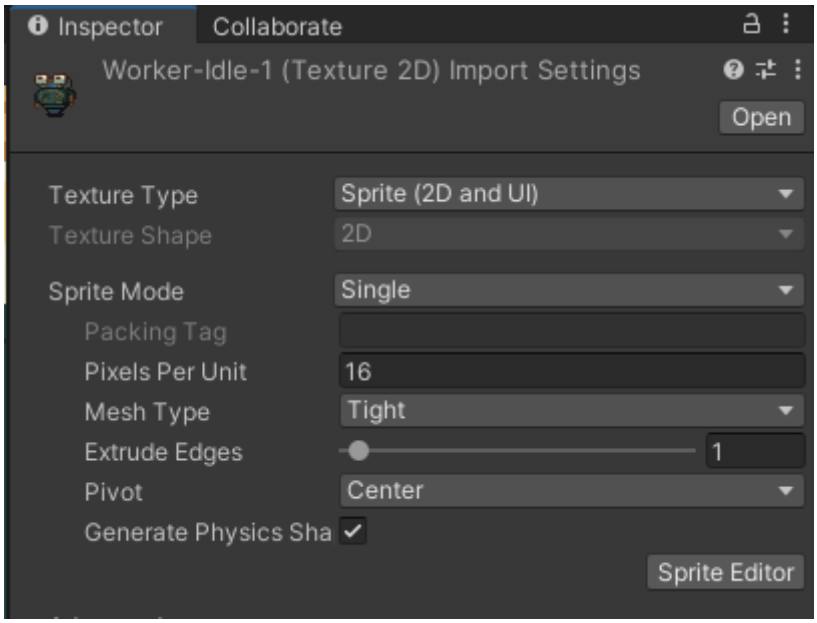
Unity sadrži ugrađeni Sprite Editor koji omogućuje izdavanje sprite grafika iz većih slike što dalje omogućuje uređivanje broja komponentnih slika unutar jedne teksture. Primjer korištenja toga je za izrezivanje i odvajanje slika koje se koriste za stvaranje mape pločica (engl. Tilemap). Spriteovi se generiraju pomoću Sprite Renderer komponente, za razliku od 3D objekta koje renderira Mesh Renderer. [9]

Slika 25: Spriteovi likova u stilu piksel grafike

6.2.1. Uređivač spritea (engl. Sprite Editor) [9]

Sprite teksture većinom sadrže samo jedan grafički element no ima situacija u kojima je prikladnije kombinirati nekoliko povezanih grafika u jednu sliku. Na primjer, slika može sadržavati dijelove jednog lika sa dijelovima koji se pomiču neovisno od njega samoga. Unity olakšava izdvajanje elemenata iz takvih složenih slika pružanjem uređivača spritea.

Kako bi neku sprite teksturu mogli uređivati u uređivaču spriteova, za nju je potrebno u Inspektoru postaviti da je „Sprite(2D and UI)“ te ako sprite tekstura sadrži više elemenata također je potrebno postaviti njezin „Sprite Mode“ na „Multiple“. Nakon ispravno postavljenih svojstva teksture gumb za otvaranje uređivača spriteova nalazi se ispod njih.



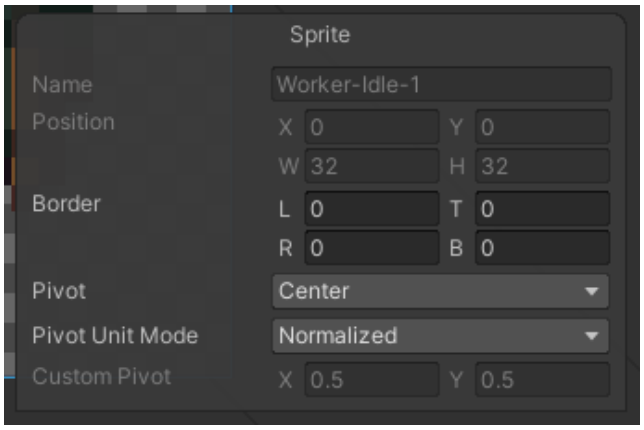
Slika 26: Postavke sprite teksture unutar Inspektora

U prozoru uređivača spritea uz kompozitnu sliku, u njemu se nalaze brojne kontrole na traci pri vrhu prozora. Klizač u gornjem desnom dijelu kontrolira zumiranje, dok gumb na traci boja s lijeve strane bira hoćete li biti prikazana sama slika ili njezine alfa razine. Desni klizač kontrolira pikselaciju teksture. Pomicanje klizača ulijevo smanjuje razlučivost sprite teksture. Najvažnija kontrola je izbornik „Slice“ u gornjem lijevom kutu koji daje mogućnosti za automatsko odvajanje elemenata slike. Konačno, gumbi Primijeni i Vrati omogućuju vam zadržavanje ili odbacivanje svih izvršenih promjena.

Najizravniji način korištenja uređivača je ručno prepoznavanje elemenata. Klikom na sliku, pojavljuje se pravokutno područje odabira s ručicama u uglovima. Tada je samostalno moguće micati ručke ili rubove pravokutnika kako bi promijenili veličinu oko određenog elementa. Nakon što je element izolirani u pravokutniku, moguće je dodati drugi povlačenjem novog pravokutnika u zasebni dio slike. Kod odabira pravokutnika, na desnom donjem kutu, pojavljuje se ploča sa kontrolama za njegovu manipulaciju.

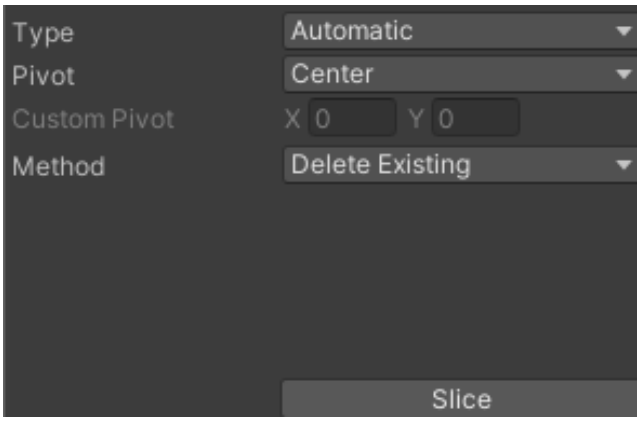
Kontrole na ploči omogućuju odabiranje naziva za Sprite grafiku te postavljanje položaja i veličine pravokutnika prema njegovim koordinatama. Širina obruba za lijevu, gornju, desnu i donju stranu može se odrediti u pikselima. Granice su korisne kod 9-Slicing Spritea. Postoje i postavke za Spriteov pivot, koji Unity koristi kao ishodišnu koordinatu i glavnu "točku

sidrišta" grafike. Moguće je birati između određenog broja zadanih položaja koji se odnose na pravokutnik (npr. U sredini, gore desno itd.) ili koristiti prilagođene koordinate.



Slika 27: Ploča s kontrolama pravokutnika elementa sprite teksture

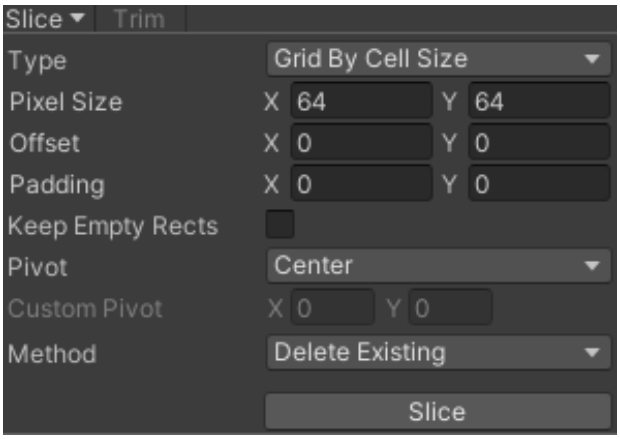
Ručno izoliranje Sprite pravokutnika dobro funkcionira, ali u mnogim slučajevima Unity može uštedjeti vrijeme kod odvajanja spriteova svojom opcijom automatskog izdvajanja. S vrstom rezanja postavljenom na automatsko, uređivač granice sprite elemenata pogađa pomoću transparentnosti. Tada je moguće postaviti zadani pivot za svaki identificirani Sprite. Izbornik Metoda omogućuje odabir načina postupanja sa trenutnim odabirom u prozoru.



Slika 28: Opcije automatskog izdvajanja sprite tekstura

Opcije „Grid by Cell Size“ ili „Grid by Cell Count“ također su dostupne za vrstu rezanja. To je vrlo korisni kada su spriteovi već postavljeni u pravilnom uzorku tijekom stvaranja. Vrijednosti veličine određuju visinu i širinu pločica u pikselima. Kod odabira „grid by cell count, „Column & Row“ određuje broj stupaca i redaka koji se koriste za rezanje. Također je moguće koristiti vrijednosti „Offset“ za pomicanje položaja mreže s gornjeg lijevog dijela slike, a

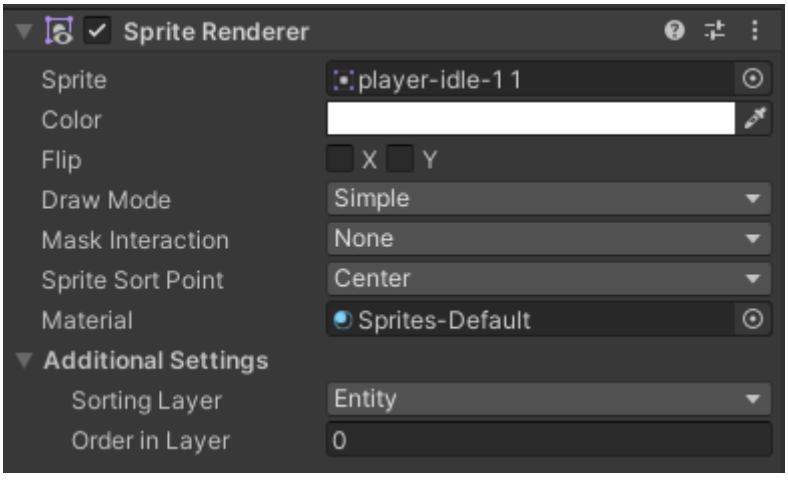
vrijednosti „Padding“ za umetanje Sprite pravokutnika malo izvan mreže. Pivot se može postaviti na jedno od devet unaprijed postavljenih mjesta ili se može definirati prilagođeno mjesto pivota.



Slika 29: Opcije „Grid by Cell Size“ izdvajanja sprite tekstura

6.2.2. Sprite Renderer [9]

Komponenta Sprite Renderer prikazuje Sprite i kontrolira kako se vizualno pojavljuje u sceni za 2D i 3D projekte. Kod kreiranja spritea kroz GameObject > 2D Object > Sprite, Unity tomu objektu automatski dodaje komponentu Sprite Renderer. Sprite Renderer komponentu također je moguće dodati na već postojeće objekte kroz Inspektor ili izbornika komponenata (Component > Rendering > Sprite Renderer).



Slika 30: Komponenta Sprite Renderer

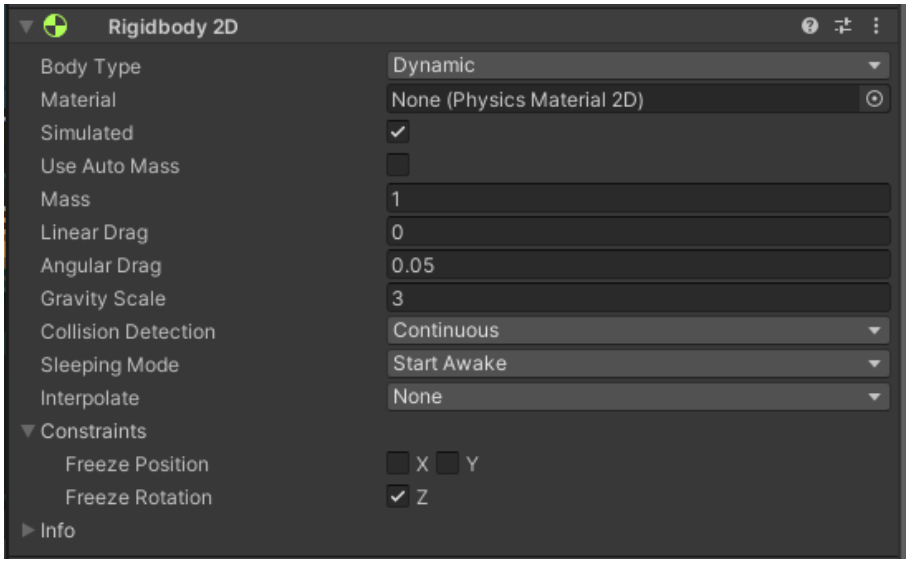
U komponenti Sprite Renderer moguće je postavljanje i mijenjanje njezinih svojstva. Neke od bitnijih svojstva su „Sprite“ koji definira koju će objekt teksturu prikazivati na svojem mjestu, „Color“ pomoću kojega je moguće promijeniti boju cijele sprite slike, „Flip koji okreće teksturu po X i Y osi bez da utječe na komponentu „Transform“ na objektu i ostale.

6.3. 2D fizika (engl. 2D Physics) [9]

Unity ima zaseban fizički motor za rukovanje 2D fizikom kako bi se iskoristile optimizacije dostupne samo s 2D. Komponente odgovaraju standardnim komponentama 3D fizike kao što je Rigidbody, Box Collider i Hinge Joint, ali uz naziv "2D". To znači da spriteovi mogu koristiti komponente Rigidbody 2D, Box Collider 2D i Hinge Joint 2D. Većina 2D fizičkih komponenata jednostavno su "spljoštene" verzije 3D ekvivalenata (npr. Box Collider 2D je kvadrat, dok je Box Collider kocka), uz nekoliko iznimaka.

6.3.1. Rigidbody 2D

2D komponenta Rigidbody stavlja objekt pod kontrolu fizičkog motora. Mnoga svojstva poznata iz standardne komponente Rigidbody-a se prenosi na Rigidbody 2D. Razlike su u tome što se u 2D predmeti mogu kretati samo u ravnini XY i mogu se okretati samo na osi okomitoj na tu ravninu.

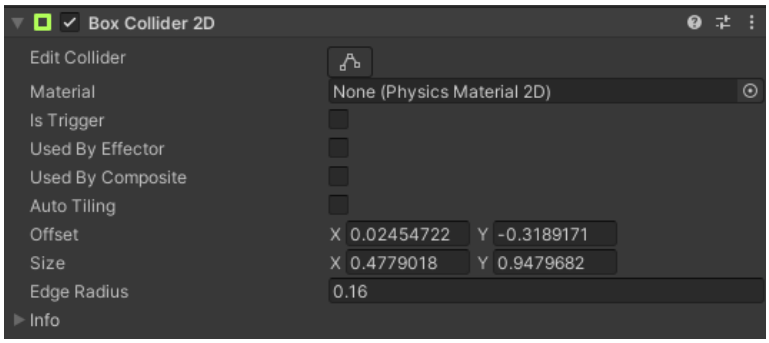


Slika 31: Komponenta Rigidbody 2D i njezina svojstva

6.3.2. Collider 2D

2D komponente Collidera definiraju oblik 2D GameObject-a za potrebe fizičkih sudara. Collider, koji je nevidljiv, ne mora imati potpuno isti oblik kao što ima GameObject u sceni, postavljanje gruba aproksimacija često učinkovitije i ne razlikuje se kod igranja video igre. Svi Collider-i za GameObjekte imaju nazive koji završavaju sa „2D“. Collideri koji u nazivima nemaju „2D“ namijenjeni su za uporabu na 3D objektima. Vrlo je bitno ne miješati 3D objekte i 2D Collidere te obrnuto.

Neki od 2D Collidera koji se mogu koristiti sa Rigidbody 2D komponentom su Circle Collider 2D, Box Collider 2D, Polygon Collider 2D, Capsule Collider 2D i ostali.



Slika 32: Komponenta Box Collider 2D i njezina svojstva

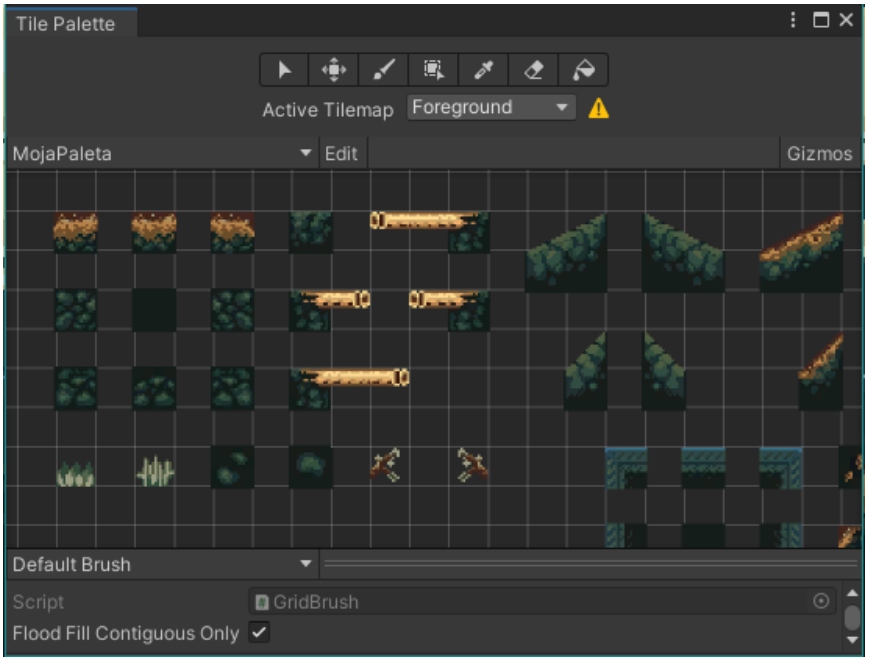
6.4. Tilemap

Komponenta Tilemap sustav je koji pohranjuje i obrađuje pločice (engl. Tiles) za stvaranje 2D razina u projektu. Potrebne podatke s pločica postavljenih na njega prenosi na druge povezane komponente kao što su Tilemap Renderer i Tilemap Collider 2D.

Komponenta 2D Tilemap Editor nije uključen u zadanu instalaciju Unity-a, stoga ga je potrebno putem Upravitelja Paketa (engl. Package Manager) preuzeti.

Slika 33: Komponenta Tilemap

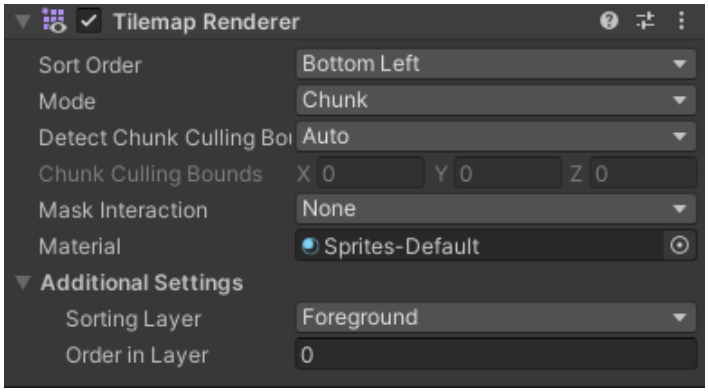
Kod izrade Tilemap-a, komponenta rešetke (engl. Grid) automatski je dodijeljena kako bi služila kao vodič prilikom izrade nivoa. Paleta pločica (engl. Tile Palette) i njezini alati koriste se za stvaranje, izmjenu i odabir pločica za slikanje na Tilemap. Alati za slikanje na Tilemapu nalaze se na gornjem dijelu prozora palete te se oni također mogu koristiti za uređivanje nje same. Ti alata su Alat za odabir (engl. Select Tool), Alat za premještanje (engl. Move Tool), Kist za slikanje (engl. Paintbrush Tool), Alat za popunjavanje kutija (engl. Box Fill Tool), Alat za biranje (engl. Picker Tool), Alat za brisanje (engl. Eraser Tool) i Alat za popunjavanje (engl. Flood Fill Tool).



Slika 34: Prozor palete pločica (engl. Tile Palette window)

6.4.1. Tilemap Renderer

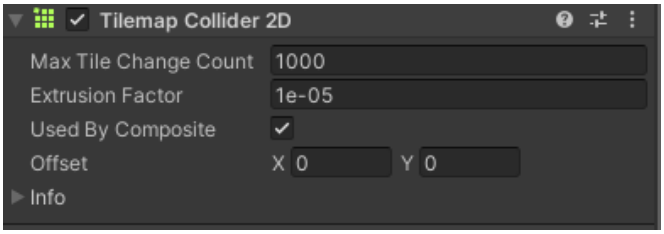
Tilemap renderer je komponenta koja je uvijek dio Tilemap objekta. Ona preko svojih svojstva određuje kako će pločice postavljene na Tilemapu biti prikazane. Neke od bitnijih svojstva su „Sort Order“ koji određuje smjer po kojem su pločice na toj odabranoj Tilemapi sortirane, „Mode“ koji određuje na koji tip načina će se mapa prikazivati („Chunk“ odnosno u grupama ili „Individual“ odnosno zasebno).



Slika 35: Komponenta Tilemap Renderer i njezina svojstva

6.4.2. Tilemap Collider 2D

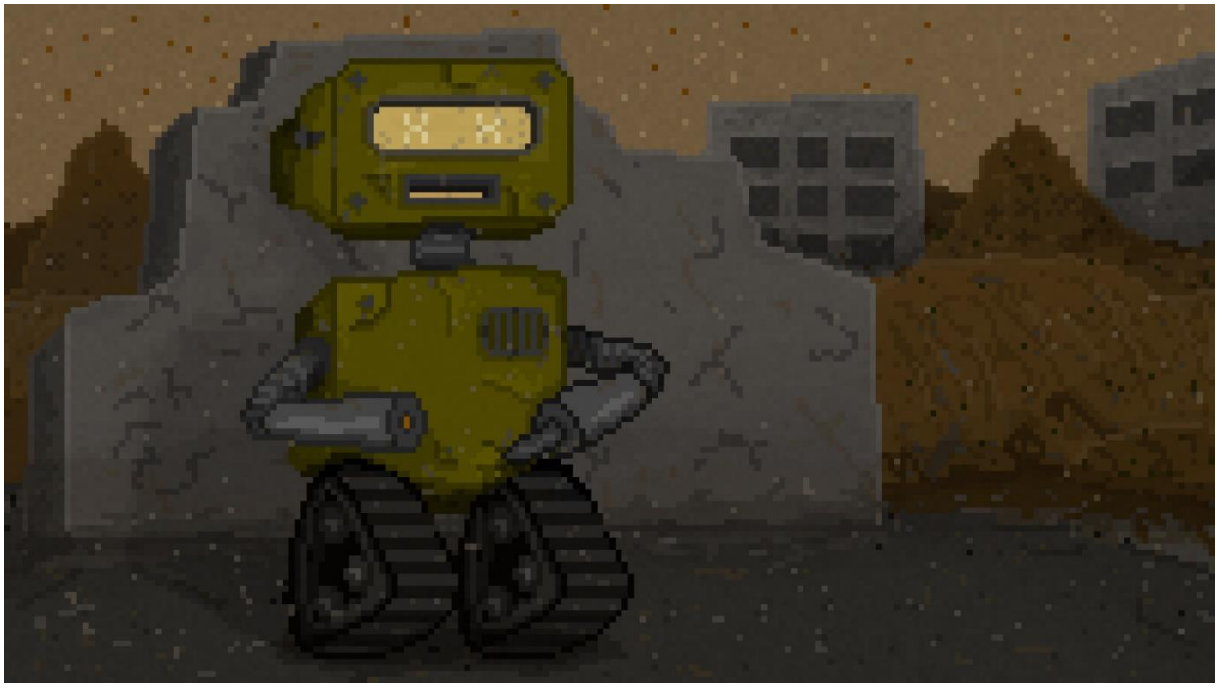
Komponenta Tilemap Collider 2D generira Collider za sve pločice na Tilemap komponenti koja se nalazi na istom objektu. Kod dodavanja ili uklanjanja pločice na komponenti Tilemap, Tilemap Collider 2D ažurira oblike Collidera tijekom LateUpdate funkcije. Zajedno kombinira višestruke promjene pločica kako bi osigurao minimalan utjecaj na performanse.



Slika 36: Komponenta Tilemap Collider 2D i njezina svojstva

7. Praktičan dio

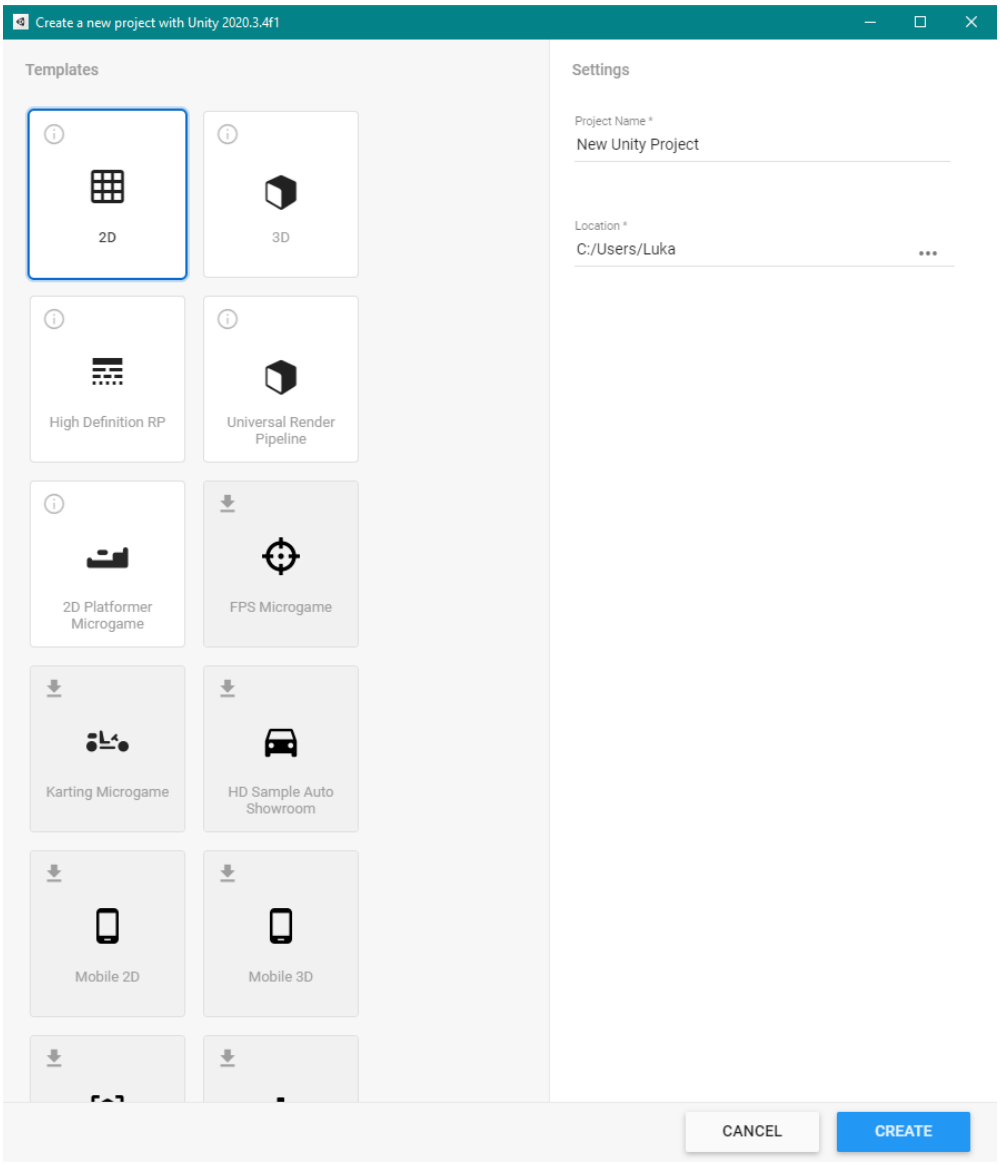
Za praktičan dio rada odlučena je izrada jednog nivoa 2D platformer računalne igre. Rad je započeo izradom koncept slike koja je bila namijenjena za sliku glavnog izbornika te je služila kao referenca za izgled glavnog lika u samoj igri. Tema igre je bila inspirirana vlastitom umjetnošću i Run and gun tipom računalnih igara. Nakon izrade osnovnih spriteova rad se odvijao u Unity-u gdje su na početku bile izrađene i testirane sve osnovne skripte za kretanje glavnog lika te su uz nje napravljene i prve animacije stanja lika. Nakon dovršetka izrade osnovne funkcionalnosti i animacije fokus je bio na izradi piksel grafike odnosno Tilemap-e koja je, kao i svi ostali spriteovi, bila izrađena u programu Adobe Photoshop. Nakon postavljanja testnog nivoa u video igru dodani su interaktivni elementi i neprijatelji te sistemi poput života igrača. Rad je završio finalnim testiranjem i izgradnjom video igre za platformu Windows. [10]



Slika 37: Praktičan dio, Slika za glavni izbornik

7.1. Postavljanje projekta u Unity-u

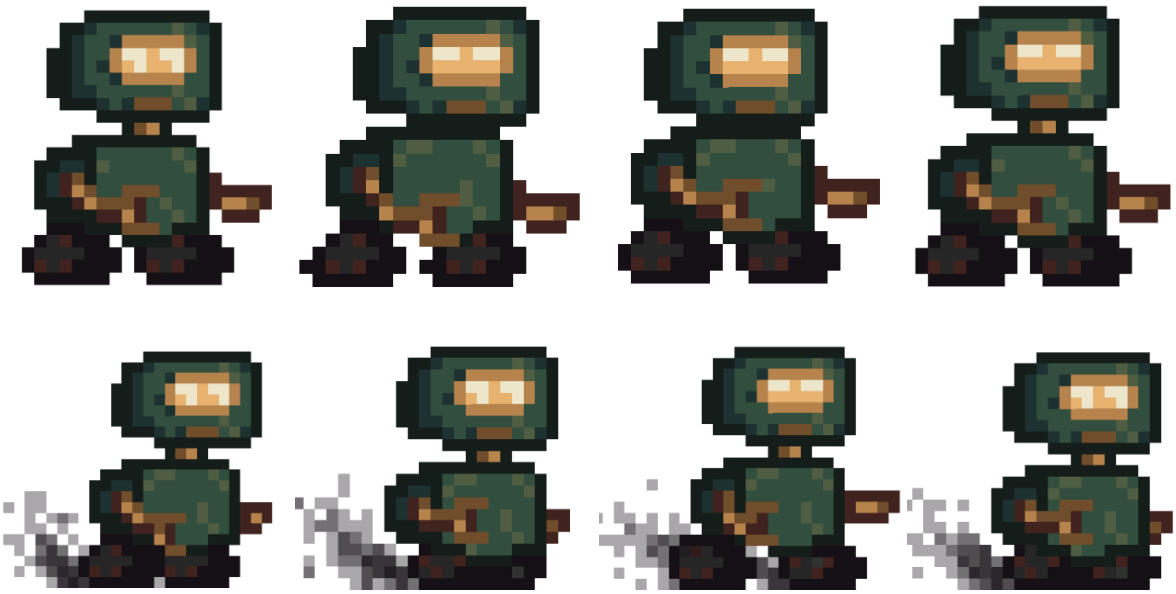
Izrada video igre započeta je kreiranjem novog projekta u Unity-u. Prilikom instalacije samog programa izrađen je korisnički račun koji omogućuje korištenje Unity Hub-a. Unity Hub služi korisnicima za pregledan prikaz svih projekata te za jednostavno otvaranje istih. Kod izrade novog projekta u postavkama je odabrani predložak za izradu 2D igre pomoću kojega su sve postavke unutar toga projekta prilagođene za rad u 2D prostoru. Također u postavkama projekta bilo je odabrano mjesto spremanja svih datoteka koje su vezane uz taj projekt te ime projekta.



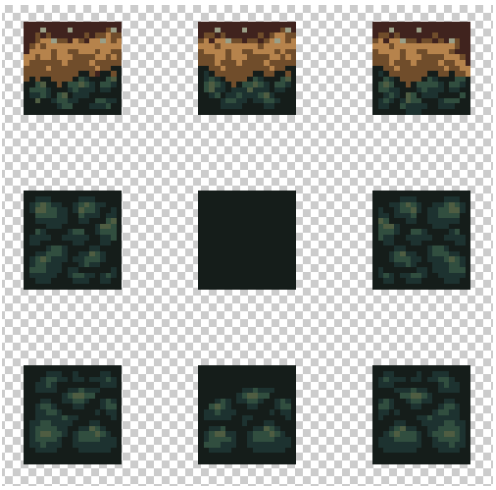
Slika 38: Praktičan dio, Postavke kreiranja projekta

7.2. Izrada prvih spriteova (piksel grafike)

Nakon postavljanja projekta, u programu Adobe Photoshop bili su kreirani prvi spriteovi odnosno dizajn glavnog lika kojeg kontrolira igrač. Za njega su izrađena neka od stanja potrebnih stanja, to su bila stanje mirovanja i stanje hodanja. Cilj je bio dobiti lika koji daje dojam 3D-a, te ne samo bočno spljoštenog lika. Kako bi rad u Unity-u mogao pravilno započeti primijećeno je da je glavnom liku također potreban pod na kojem će moći stajati, stoga je bio izrađen i prvi dio seta pločica (engl. Tileset) koji je sadržao šest pločica. Spriteovi lika i prvi dio seta pločica uveden je u Unity jednostavnim povlačenjem i ispuštanjem u Prozor Projekta.

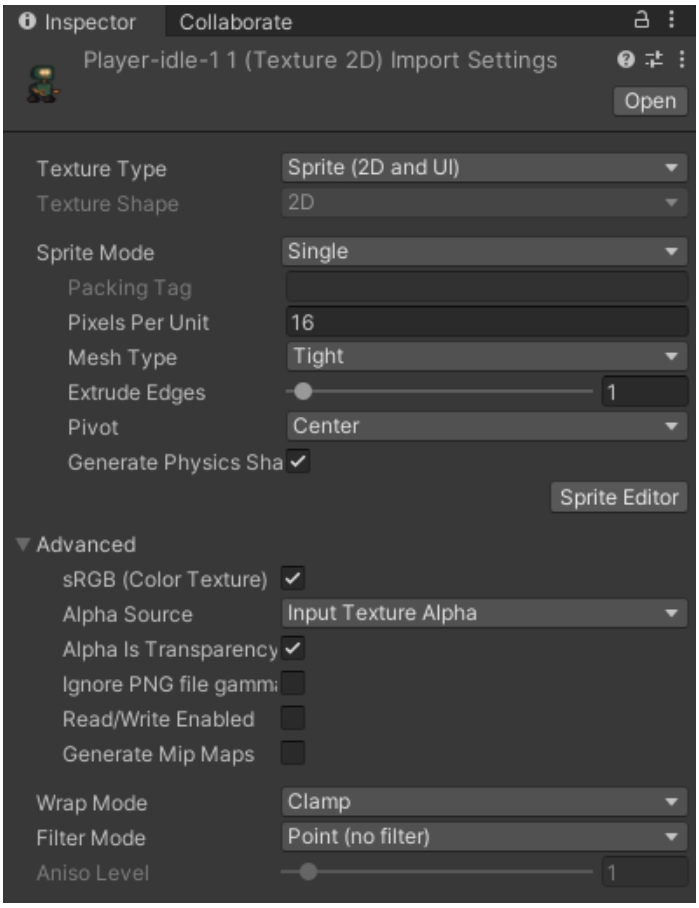


Slika 39: Praktičan dio, Prvi spriteovi i animacije glavnog lika



Slika 40: Praktičan dio, Početni set pločica (engl. Tileset)

Nakon što u Unity uvezeni početni dijelovi grafike bilo im je potrebno prilagoditi svojstva, u prozoru Inspektora, da bi se pravilno pokazivali na sceni. Za tip grafike ovog rada kod svih slika korištenih kao spriteovi bilo je potrebno postaviti da je njihov Texture Type Sprite (2D i UI), također zbog korištenja piksel grafike svojstvo Pixel per Unit mora biti postavljeno na 16 da bi spriteovi bili normalne veličine. Opciju „Filter mode“ također je potrebno postaviti na Point (no filter) kako grafika ne bih bila mutna.



Slika 41: Praktičan dio, Postavljena svojstva svakog spritea

7.2.1. Postavljanje palete pločica

Nakon postavka svih svojstva, datoteku koja sadrži set izrađenih pločica bila je izrezana odnosno izdvojena pomoću Sprite Uređivača. Kako bi to bilo moguće potrebno je izmijeniti Sprite Mode na više spriteova (engl. Multiple). U Sprite Uređivaču set je bio izrezan pomoću opcije Grid by Cell Size u kojoj je veličina ćelije bila postavljena na 16x16 piksela. U prozoru palete pločica izrađena je nova paleta pod nazivom MojaPaleta u koju je bio povučen prethodno izrezani set. U MojuPaletu biti će spremljeni svi setovi koji će biti korišteni za izradu prednjeg dijela nivoa po kojem se igrač kreće.

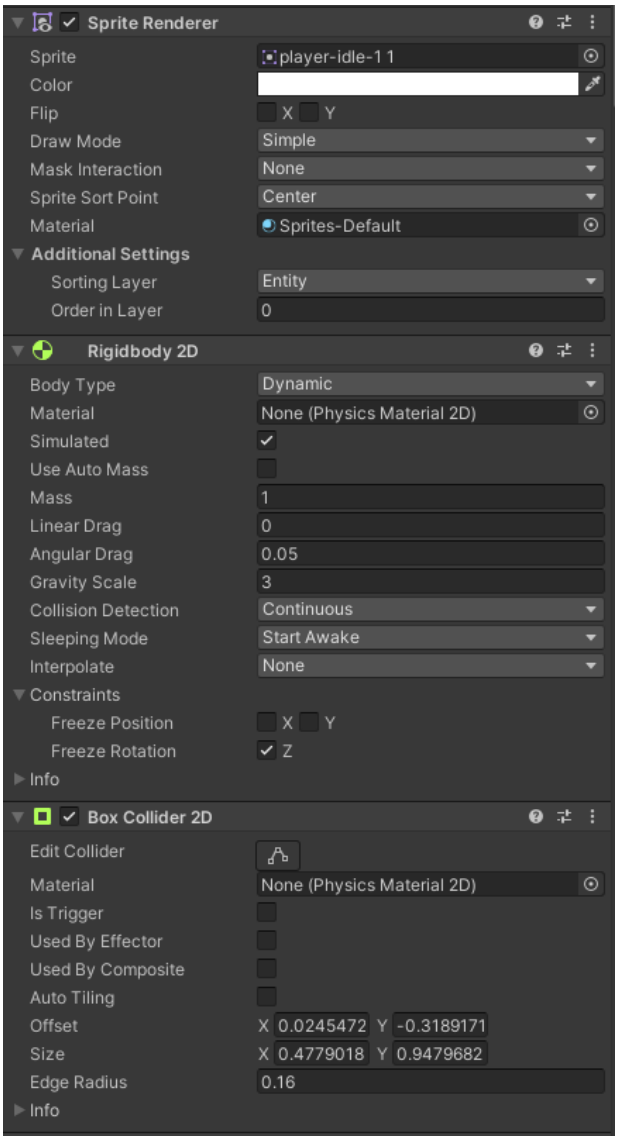
Slika 42: Praktičan dio, Prva paleta pločica

7.2.2. Izrada testnog prostora u sceni

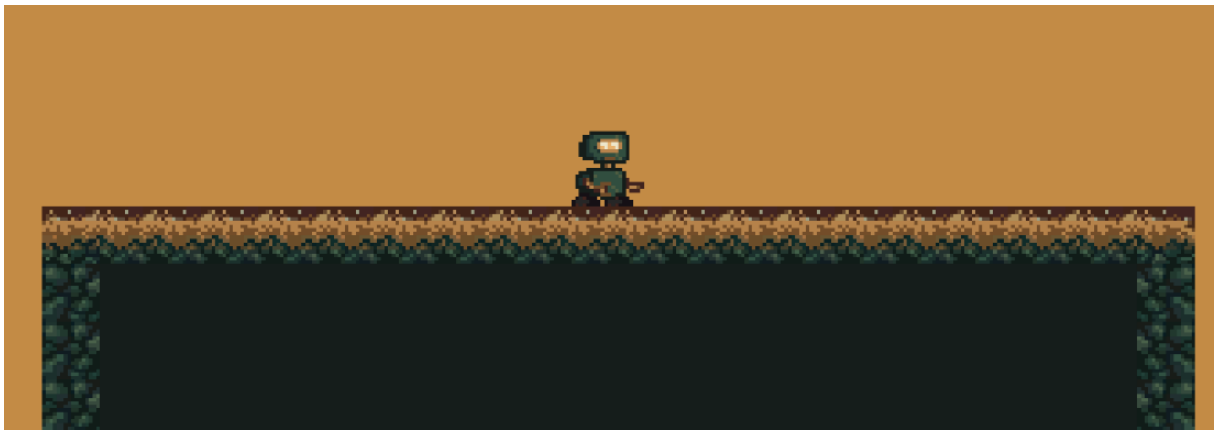
Za primjenu izrađene palete u prozoru Hijerarhije, odnosno na sceni, izrađen je novi prazni objekt pod imenom Level, u njega su ugniježđeni još 4 prazna objekta koji su bili nazvani Background, Foreground, „MidBG“ i MidBGprops. U te objekte biti će stavljeni spriteovi i setovi pločica koji su namijenjeni primarno za izgled nivoa, stoga je svakom objektu bilo potrebno dodati komponente Tilemap i Tilemap Renderer kako bi se prikazivali na sceni. Jedini objekt koji će sadržavati interaktivne pločice je Foreground u kojeg je zbog toga dodana komponenta Tilemap Collider 2D.

Slika 43: Praktičan dio, Hijerarhija objekta Level

Na objektu Foreground, korištenjem izrađene palete pločica, izrađeni je prvi testni prostor na kojem će sprite glavnog lika moći stajati. Tada je kreirani novi prazan objekt pod imenom player. Na njega su bile dodane komponente Sprite Renderer na kojega je u svojstva bio povučen već uvezen sprite glavnog lika, Rigidbody 2D kod kojeg su bila podešena svojstva gravitacije i zamrznuta rotacija po osi Z i Box Collider 2D koji je podešen kako bi odgovarao veličini spritea lika.



Slika 44: Praktičan dio, Dodane komponente objektu "player" i njihova svojstva



Slika 45: Praktičan dio, Testni prostor u sceni

7.3. Glavni lik

7.3.1. Skriptiranje glavnog lika

Nakon izrade osnovnog prostora na kojem sprite glavnog lika može stajati, na objektu player, kroz inspektor, kreirana je skripta PlayerControls koja će sadržavati sav potreban kod kako bi igrač mogao fluidno kontrolirati likom.

Prvo kako bi mogli manipulirati s pozicijom objekta, izrađene su varijable koje povezuju komponente „Rigidbody2D“ i „Collider2D“ sa skriptom. Varijable su privatne stoga su im u funkciji Start() dodijeljeni određeni komponenti.

U skriptu su dodane tri funkcije koje su zaslužene za kretanje lika. Glavna funkcija Movement() govori skripti da, kod pritiska inputa za horizontalne osi, lik se kreće po x osi. Smjer kretanja ovisi o inputu igrača, a brzina je zadana pomoću varijable speed koju je moguće mijenjati i modificirati. Funkcija za skok izvedena je posebnom funkcijom Jump() koja se poziva u funkciji Movement() u slučaju kada igrač pritisne input „W“ i kada je brzina, na Y osi, objekta player jednaka nuli. Funkcija Jump() daje objektu vertikalnu brzinu jednaku izrađenoj varijabli jumpforce koja se može modificirati. Funkcija Flip() ima svrhu okretanja spritea modela za 180 stupnjeva prilikom kretanja u suprotnu stranu, također je pozvana iz funkcije Movement(). Funkcija Movement() na kraju je prizvana u već definiranoj funkciji Unity-a, Update(), kako bila aktivna kroz cijelo vrijeme igranja.

```
Unity Script | 2 references
7   public class PlayerControls : MonoBehaviour
8   {
9
10      private Rigidbody2D rb;
11      private Collider2D coll;
12      [SerializeField] private float speed = 5f;
13      [SerializeField] private float jumpforce = 8f;
14
```

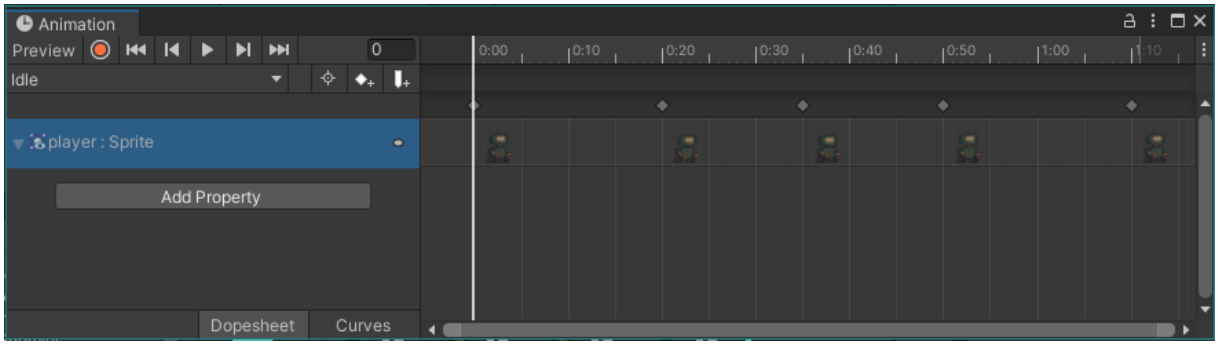
Slika 46: Praktičan dio, Varijable potrebne u funkcijama za kontrolu lika

```
1 reference
private void Movement()
{
    float hDirection = Input.GetAxis("Horizontal");
    //left
    if (hDirection < 0)
    {
        rb.velocity = new Vector2(-speed, rb.velocity.y);


    }
    //right
    else if (hDirection > 0)
    {
        rb.velocity = new Vector2(speed, rb.velocity.y);


    }
    if ((rb.velocity.x > 0.1 && !facingRight) || (rb.velocity.x < -0.1 && facingRight))
    {
        flip();
    }
    //jump
    if (Input.GetButtonDown("Jump") && Mathf.Abs(rb.velocity.y) < .001f)
    {
        Jump();
    }
}
1 reference
void flip()
{
    facingRight = !facingRight;
    transform.Rotate(0f, 180f, 0f);
}

2 references
private void Jump()
{
    rb.velocity = new Vector2(rb.velocity.x, jumpforce);
    state = State.jumping;
}
```

Slika 47: Praktičan dio, Funkcije za kontrolu lika

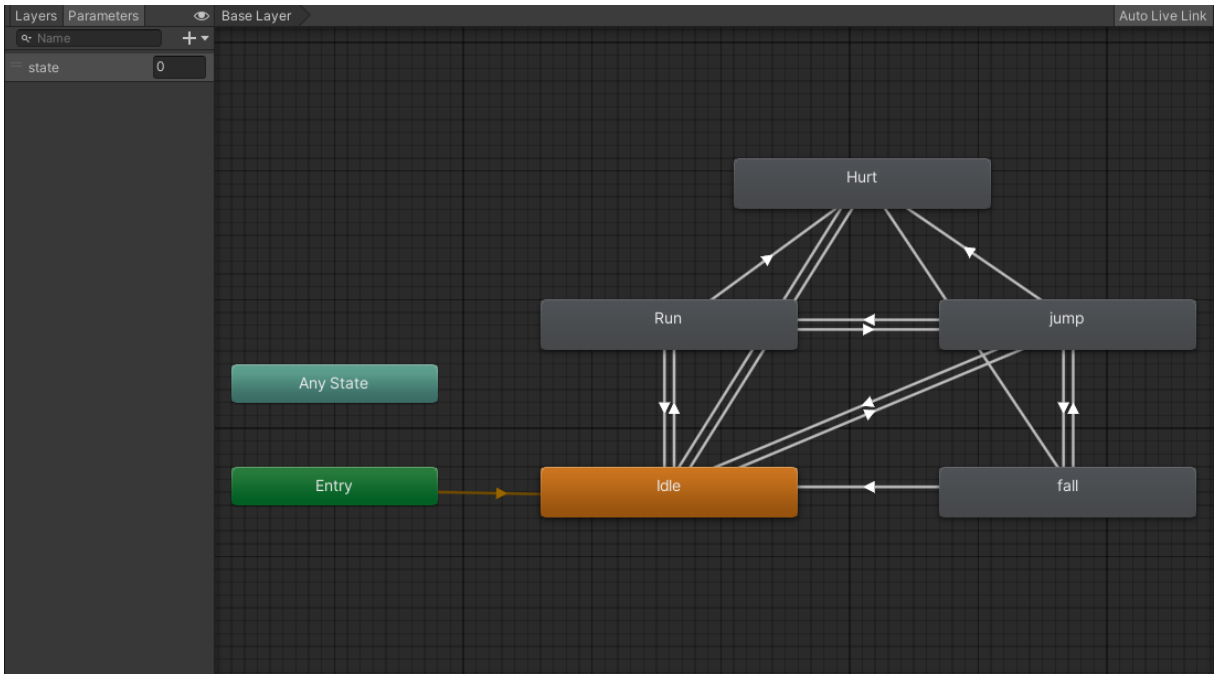
7.3.2. Animiranje i postavljanje stanja glavnog lika

Animacije svih stanja glavnog lika izrađene su u prozru za animiranje na jednostavan način dodavanja svakog pojedinog sprajta stanja u vremensku traku. Za glavnog lika kreirane su 5 animacija za stanja koja će biti korištena u igri. Kreirane animacije su za stanje mirovanja, stanje trčanja, stanje skoka, stanje pada i stanje ozlijede. Spriteovi za neke od njih naknadno su izrađeni tokom rada.



Slika 48: Praktičan dio, Primjer izrade animacije stanja mirovanja

Nakon toga, kako bi promjene između stanja radile, postavljene su tranzicije između svakog stanja u prozoru Animator. To je izvedeno na način da jedan parametar „state“, koji je bio tipa Int, u Animatoru kontrolira koje će stanje biti kada aktivno. Stoga je stanje mirovanja imalo vrijednost 0, stanje trčanja 1, stanje skoka 2, stanje padanja 3 i stanje ozljede 4.



Slika 49: Praktičan dio, Tranzicije između stanja u prozoru Animator

U skripti PlayerControls izrađena je funkcija koja mijenja stanja postavljena u Animatoru preko njegovog parametra state. U skriptu je prvo dodana privatna varijabla koja u funkciji Start() povezuje komponentu Animator sa skriptom kako bi bilo moguće manipulirati s njegovim parametrom state i mijenjati animacije glavnog lika. Nakon toga dodana je varijabla tipa enum koja može sadržavati više stringova na koje se u kodu gleda kao brojeve od 0 na dalje. Ta varijabla State omogućuje nam mijenjanje stanja preko koda, do Animatora. Funkcija VelocityState() zadužena je za mijenjanje stanja na temelju brzine i smjera kretanja objekta playera.

```
16      private Animator anim;
17
        14 references
18      private enum State {idle, running, jumping, falling, hurt}
19      private State state = State.idle;
20
```

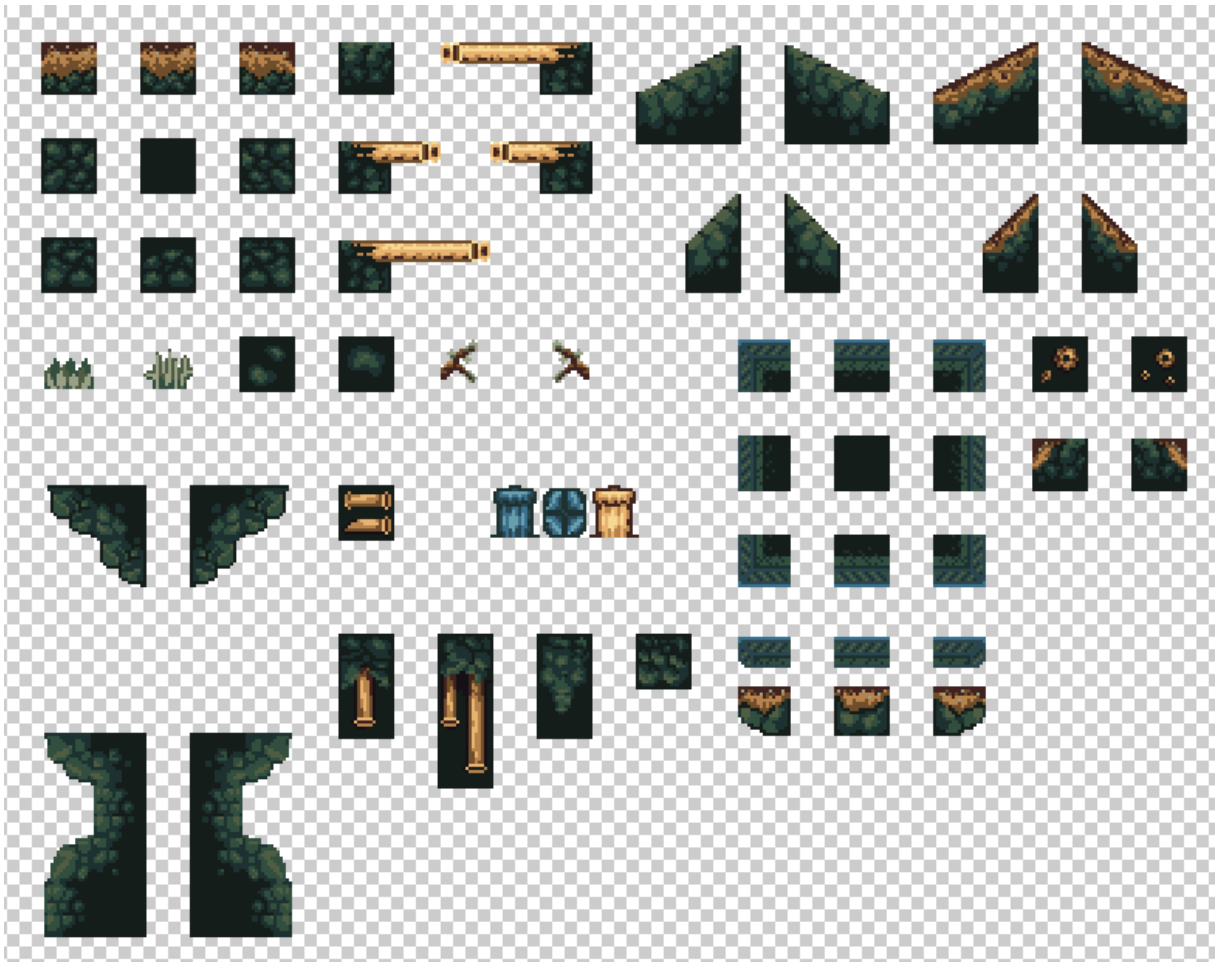
Slika 50: Praktičan dio, Varijable korištene za funkciju kontroliranja stanja

```
167     private void VelocityState()
168     {
169
170
171         if(state == State.jumping)
172         {
173             if(rb.velocity.y < .1f)
174             {
175                 state = State.falling;
176             }
177         }
178         else if(state == State.falling)
179         {
180             if(coll.IsTouchingLayers(ground))
181             {
182                 jumpSound.Play();
183                 state = State.idle;
184             }
185         }
186         else if (state == State.hurt)
187         {
188             if(Mathf.Abs(rb.velocity.x) < .1f)
189             {
190                 state = State.idle;
191             }
192         }
193         else if(Mathf.Abs(rb.velocity.x) > 2f)
194         {
195             //moving
196             state = State.running;
197         }
198         else
199         {
200             state = State.idle;
201         }
202     }
203
```

Slika 51: Praktičan dio, Funkcija zadužena za kontroliranje stanja glavnog lika

7.4. Izrada cjelokupnog dizajna nivoa

Završavanje izrade spriteova i seta pločica koje će biti korištene za na cjelokupnom nivou. U paletu pločica dodani su različiti tipovi podova i zidova, izrađene su platforme i razni elementi koji će se nalaziti unutar zemlje kako bi video igra postavila igraču što bolji vizualni dojam. Izrađene su i slike koje će se nalaziti u pozadini, odnosno slike neba i grada. Za bliži plan izrađeno je nekoliko tipova spriteova poput kanta za smeće, velikih hrpa smeća, trave, starog razbijenog auta. Svi novi spriteovi i slike uvezeni su u prozor projekta te su pomoću Uređivača Spriteova odvojeni i postavljeni u nove palete. Dizajn nivoa se tokom izrade rada mijenjao po potrebama dizajna i boljeg osjećaja igrača.



Slika 52: Praktičan dio, Završeni set pločica korišten u dizajnu nivoa

Slika 53: Praktičan dio, Prikaz dizajna nivoa

7.5. Neprijatelji

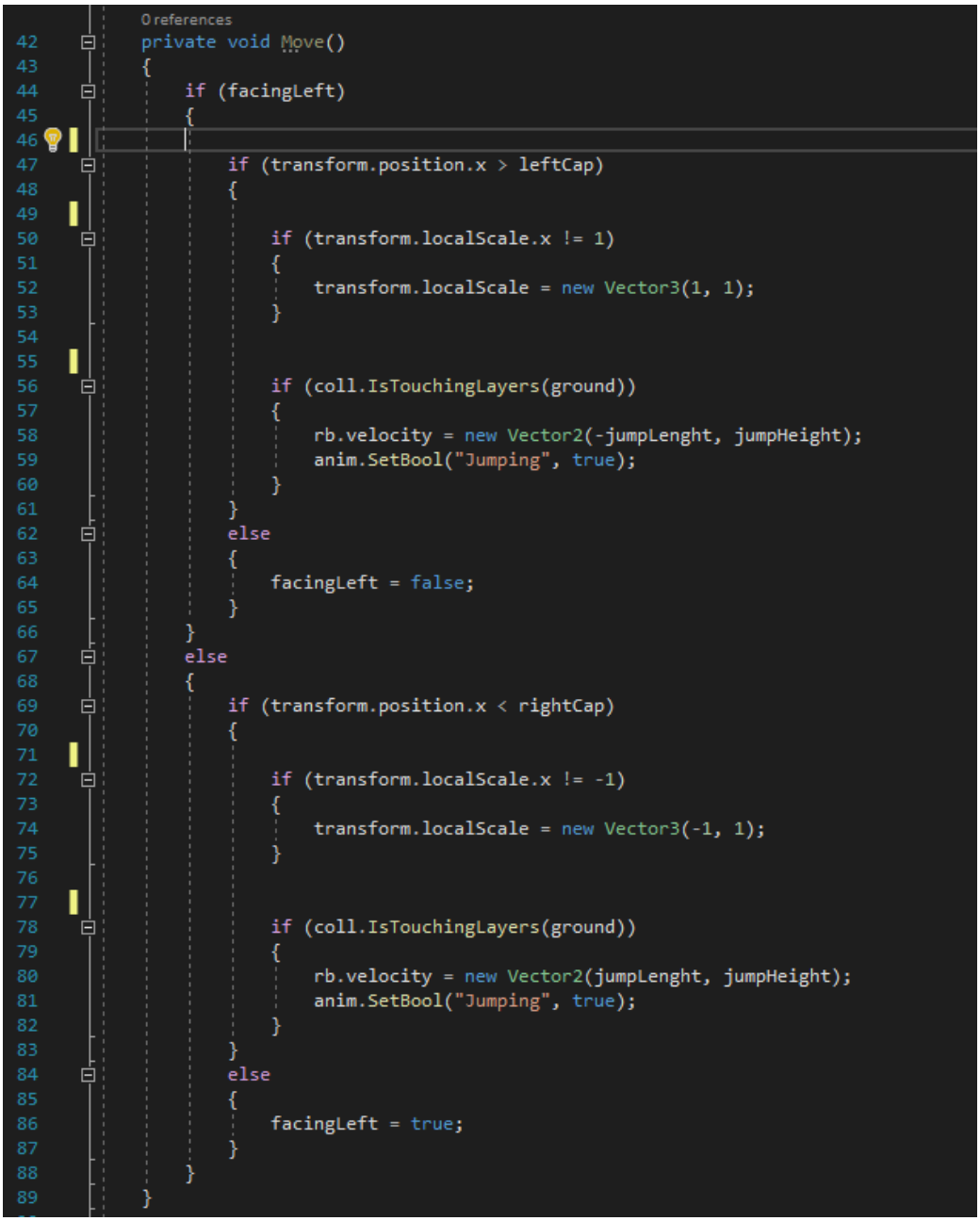
Za dizajn i tip neprijatelja u radu je odlučeno izraditi dvije vrste od kojih svaka ima drugačije funkcije kretanja i radnje. Jedan od tipa neprijatelja je obični radnik koji patrolira, odnosno skače, od točke do točke, koje mogu biti zadane. Igrač u slučaju kontakta s njime izgubi život te ga radnik odbaci u stranu, no igrač može jednostavno izbjeći takav tip neprijatelja. Drugi tip neprijatelja zamišljen je kao leteći robot koji nadgleda radnike te sa svojim laserom puca na igrača. Ovi tipovi neprijatelja postavljeni su na mjesta gdje ih igrač ne može zaobići te djeluju puno agresivnije.

7.5.1. Radnik

Inspiracija za dizajn robota radnika dobivena je iz vlastitih crteža te je cilj njegovog izgleda bio da ne djeluje agresivno pošto samo predstavlja radnika. Stoga su u programu Adobe Photoshop izrađeni spriteovi za sva njegova stanja. U Unity-u izrađen je novi objekt pod imenom „Worker“ te su u njega postavljene sve komponente Sprite Renderer, Rigidbody 2D i Box Collider 2D kako bi se sprite prikazivao na sceni. Zatim su izrađene animacije te su njegova stanja bila postavljena u animatoru kako bi bila spremna prilikom izrade skripte.

Slika 54: Praktičan dio, Izgled spritea radnika

Skripta radnika sadržava nekoliko varijabla koje sadrže informacije koje su potrebne za funkcije. U funkciji Update() nalazi se kod koji je zadužen za mijenjanje stanja radnika te je postavljena na sličan način kao i kod skripte glavnog lika. Funkcija Move() zadužena je za patroliranje radnika te radi na načina da provjerava ako je radnik okrenut prema lijevoj strani i ako je njegova pozicije veće od varijable leftCap (koja određuje točku okretanja) tada se okreće desno i testira ako se nalazi na sloju ground, ako da nastavlja skače po dužini i visini koja je zadana varijablama. I obrnuto.



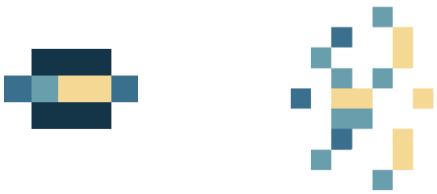
```
0 references
private void Move()
{
    if (facingLeft)
    {
        if (transform.position.x > leftCap)
        {
            if (transform.localScale.x != 1)
            {
                transform.localScale = new Vector3(1, 1);
            }

            if (coll.IsTouchingLayers(ground))
            {
                rb.velocity = new Vector2(-jumpLenght, jumpHeight);
                anim.SetBool("Jumping", true);
            }
        }
        else
        {
            facingLeft = false;
        }
    }
    else
    {
        if (transform.position.x < rightCap)
        {
            if (transform.localScale.x != -1)
            {
                transform.localScale = new Vector3(-1, 1);
            }

            if (coll.IsTouchingLayers(ground))
            {
                rb.velocity = new Vector2(jumpLenght, jumpHeight);
                anim.SetBool("Jumping", true);
            }
        }
        else
        {
            facingLeft = true;
        }
    }
}
```

Slika 55: Praktičan dio, Varijabla za patroliranje radnika

7.5.2. Leteći robot

Izrada dizajna spriteova i postavljanje letećeg robota u Unity odrađena je na isti način kao i kod prošlog neprijatelja. Leteći robot imao je samo jedno stanje pošto je bio zamišljen da leti na jednom mjestu stoga za njega nije bilo potrebno postavljanje Animatora i njegovo skriptiranje. No pošto je on bio zamisljen da puca metke u igrača, bilo je potrebno izraditi sprite metka i malu animaciju raspršenog metka.

Slika 56: Izgled spritea letećeg robota



Slika 57: Praktičan dio, Izgled spritea metka

Skripta letećeg robota sadržava varijable i funkciju koja omogućuje robotu da puca sprite metka svake tri sekunde. Kako bi to bilo moguće potrebno je objektu letećeg robota ugnijezditi još dodatni prazni objekt koji će služiti kao pozicija iz koje robot puca odnosno kao cijev njegove puške. Ta pozicija sa skriptom je spojena preko varijable weaponMuzzle, a za objekt koji robot puca izrađena je varijabla bullet koja sadrži sprite metka. Funkcija Fire() radi na način da upit povjerava ako je prošlo određeno vrijeme, zadano varijablom, te u tomu slučaj varijabla vremena se resetira, a metak se stvara na poziciji varijable weaponMuzzle. Funkcija Fire() pozvana je u funkciji Update() kako bi se konstantno odvijala.

```
10      public Transform weaponMuzzle;
11      public GameObject bullet;
12      public float fireRate = 3000f;
13      private float shootingTime;
```

Slika 58: Praktičan dio, Varijable korištene za funkciju Fire()

```
28      private void Update()
29      {
30          Fire();
31      }
        1 reference
32      private void Fire()
33      {
34          if (Time.time > shootingTime)
35          {
36              shootingTime = Time.time + fireRate / 1000;
37              Instantiate(bullet, weaponMuzzle.position, weaponMuzzle.rotation);
38              shootSound.Play();
39          }
40      }
```

Slika 59: Praktičan dio, Funkcija Fire(), Letećeg robota

7.5.3. Skripta metka

Prilikom stvaranja objekta metka on ostaje statičan te je njegovu skriptu potrebno izraditi posebno. Varijable definirane u skripti Bullet su varijabla koja određuje brzinu, varijabla koja određuje štetu, varijabla koja omogućuje manipulaciju komponenta Rigidbody 2D i varijabla za prizivanje drugog game objekta kod sudara metka. U funkciji Start(), odnosno kod njegove pojave, postavlja se brzina metka. Dok druga funkcija provjerava ako je pogođen Collider2D od skripte Enemy ili skripte PlayerControls, ako je to istina njima se uzima onolika količina života koliko je zadana varijabla šteta. Na kraju funkcije, u slučaju sudara s bilo kojim komponentom Collider 2D, poziva se animacija eksplozije metka te se objekt uništi.

```
public class Bullet : MonoBehaviour
{
    public float speed = 20f;
    public int damage = 20;
    public Rigidbody2D rb;
    public GameObject impactEffect;

    Unity Message | 0 references
    void Start()
    {
        rb.velocity = transform.right * speed;
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D hitInfo)
    {
        Enemy enemy = hitInfo.GetComponent<Enemy>();
        if (enemy != null)
        {
            enemy.TakeDamage(damage);

        }

        PlayerControls player = hitInfo.GetComponent<PlayerControls>();
        if (player != null)
        {
            player.TakeDamage(damage);

        }

        Instantiate(impactEffect, transform.position, transform.rotation);

        Destroy(gameObject);
    }


}
```

Slika 60: Praktičan dio, Kod skripte Bullet

7.5.4. Zajednička skripta neprijatelja

Zajednička skripta neprijatelja, Enemy, sadrži funkcije koje vrijede za svakog neprijatelja u projektu. Ona sadrži varijablu koja kontrolira zdravlje te funkciju koja omogućava ozljeđivanje neprijatelja. Funkcija TakeDamage(int damage) omogućuje korištenje varijable damage u drugim skriptama kako bi, na primjer, u dodiru s objektom koji sadrži zajedničku skirptu Enemy tu vrijednost damage oduzeli od varijable za zdravlje health. Ona također provjerava ako je varijabla health manja ili jednaka nuli, ako je to istina aktivira se animacija za smrt te se objekt uništi.

```
1 reference
public void TakeDamage (int damage)
{
    health -= damage;

    if (health <= 0)
    {
        anim.SetTrigger("Death");
        death.Play();
        rb.velocity = Vector2.zero;
        rb.bodyType = RigidbodyType2D.Kinematic;
        GetComponent<Collider2D>().enabled = false;
    }
}
```

Slika 61: Praktičan dio, Funkcija TakeDamage (int damage) u zajedničkoj skripti Enemy

7.6. Dodatne funkcije glavnog lika

Nakon izrade neprijatelja, glavnom liku bilo je potrebno dodati funkcije koje omogućuju bolju interakciju kod sudaranja s njima. Također je bilo potrebno izraditi sistem pomoću kojeg igrač može ozlijediti neprijatelje, za to je odlučeno koristiti sličan sistem kao kod letećeg robota.

7.6.1. Sudaranje

Funkcija za sudaranje s neprijateljima dodana je u već postojeću skriptu PlayerControls. Funkcija provjerava ako je objekt player u dodiru s bilo kojim drugim objektom koji sadrži oznaku Enemy, ako da provjerava se dali igrač skače tome objektu na glavu odnosno ako je u stanju padanja. U tome slučaju igrač se odbija od toga objekta te aktivira funkciju koja ga uništava. Uostalom ako igrač dodiruje neprijatelja sa bočne strane, odbija se od njega i ulazi u stanje ozljede, u kojoj se ne može micati, te gubi život.

```
76      private void OnCollisionEnter2D(Collision2D other)
77      {
78          if(other.gameObject.tag == "Enemy")
79          {
80              Enemy enemy = other.gameObject.GetComponent<Enemy>();
81              if (state == State.falling)
82              {
83                  enemy.JumpedOn();
84                  Jump();
85              }
86              else
87              {
88                  state = State.hurt;
89                  gothit.Play();
90                  HandleHealth(); //Updates health and resets lvl if ded
91                  if (other.gameObject.transform.position.x > transform.position.x)
92                  {
93                      //Enemy is to my right so I should be damaged and move left
94                      rb.velocity = new Vector2(-hurtForce, rb.velocity.y);
95                  }
96                  else
97                  {
98                      //Enemy is to my left so I should be damaged and move right
99                      rb.velocity = new Vector2(hurtForce, rb.velocity.y);
100                 }
101             }
102         }
103     }
```

Slika 62: Praktičan dio, Funkcija za sudaranje

7.6.2. Sistem pucanja

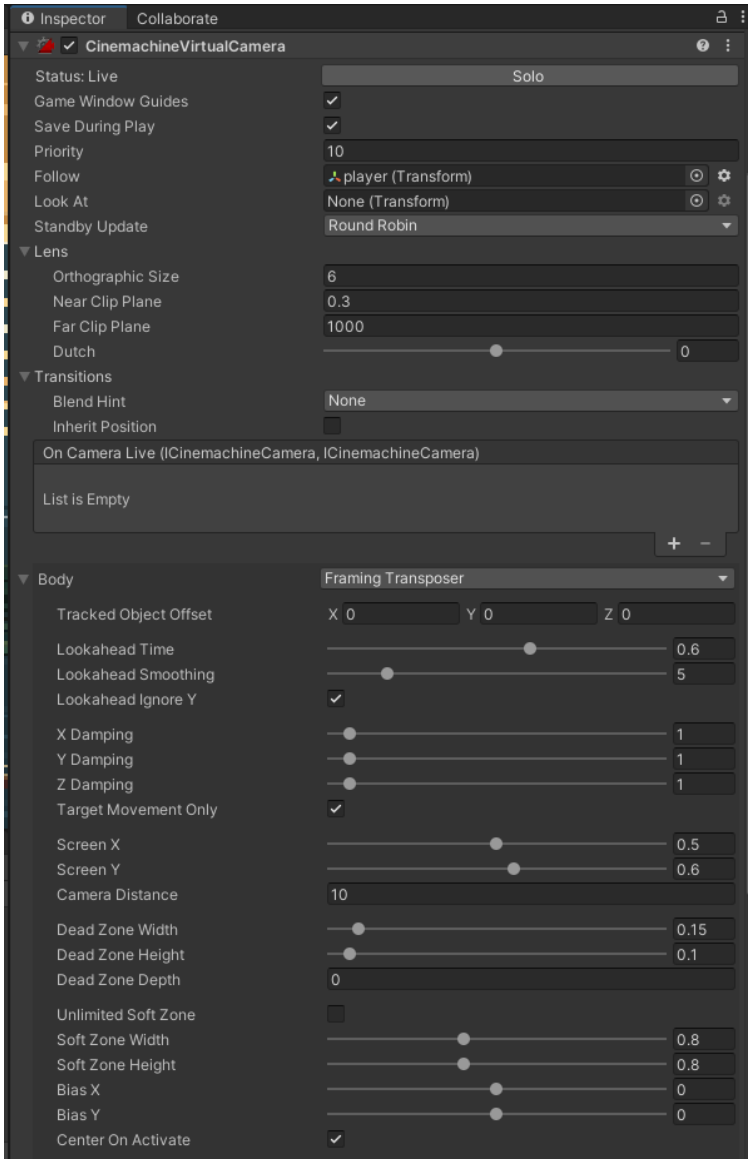
Sistem koji omogućuje igraču da puca izrađen je na sličan način kao i kod neprijatelja, letećeg robota, no umjesto da se metak stvara s obzirom na vrijeme bilo je potrebno zadati uvjet za input. Metak i skripta za metak iskorišteni su od letećeg robota.

```
4
        Unity Script | 0 references
5       public class Weapon : MonoBehaviour
6       {
7           public Transform firePoint;
8           public AudioSource shotSound;
9           public GameObject bulletPrefab;
10
11
            Unity Message | 0 references
12          void Update()
13          {
14              if (Input.GetButtonDown("Fire1"))
15              {
16                  Shoot();
17                  shotSound.Play();
18              }
19          }
20
            1 reference
21          void Shoot()
22          {
23              Instantiate(bulletPrefab, firePoint.position, firePoint.rotation);
24          }
25      }
26
```

Slika 63: Praktičan dio, Skipta za sistem pucanja

7.7. Postavljanje kamere

Do ovoga dijela izrade rada, kamera koja daje pogled igraču na video igru bila je jednostavno ugniježdena na objekt player te je grubo pratila kretanja igrača. To nije optimalno rješenje kod izrade video igra, stoga je u rad dodani predefinirani sustav kamera kodi sadrži mnogo predefinirani postavka kojima je moguće kontrolirati kako će kamera pratiti glavnog lika. Taj sustav, koji se zove Cinemachine, potrebno je preuzeti pomoću kroz Unity-evog Package Managera koji se nalazi pod izbornikom Window na glavnom izborniku. Nakon instalacije na glavnom izborniku pojavljuje se novi izbornik, Cinemachine, pomoću kojeg je u scenu dodana nova Cinemachine 2D kamera. Kako bi novokreirana kamera pratila objekt igrača, potrebno ga je povući pod svojstvo Follow koje se nalazi na toj kameri unutar Inspektora te nakon toga svojstva kamere su postavljena kako bi postigli glatke prijelaze prilikom kretanja igrača.



Slika 64: Praktičan dio, Svojstva Cinamachine kamere

7.8. Izrada UI-a

Nakon što je igra imala fluidan osjećaj za igrača, na ekran su dodane potrebne informacije poput trake koja prikazuje zdravlje glavnog lika. Uz inspiraciju na većinu 2D platformer igra također je izrađen sistem za skupljanje novčića odnosno u slučaju ovog rada to su bili mali zupčanici. U ovome radu prvog nivoa zupčanici nisu imali nikakvu svrhu osim da se pokupljuju, no biti će korišteni u daljnjem razvoju igre. Prvi korak kod postavljanja tih informacija na igračev ekran bila je izrada prilagođene grafike za njih kako bi odgovarala cjelokupnom osjećaju igre. Spriteovi za nje izrađeni su i uvedeni u program na isti način kao i svi prijašnji te su svi objekti postavljeni u prazan objekt „Player UI“.

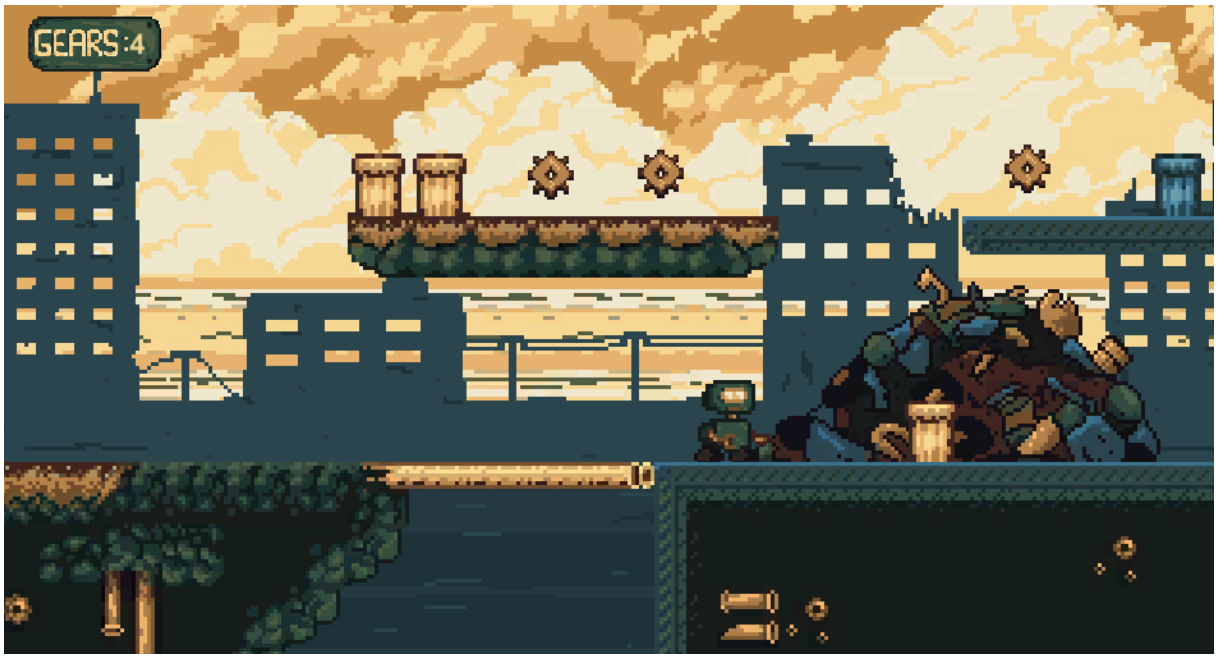
Slika 65: Praktičan dio, Spriteovi za izradu korisničkog sučelja

7.8.1. Zupčanici

Funkcija za brojanje zupčanika dodana je u skriptu koja se nalazi na igraču. Izrađena ja na način da prilikom sudaranja s objektom koji ima oznaku „Collectable“ u zadanoj varijabli gears broj se povećava za 1, ta varijabla tada se prebacuje u drugu varijablu gearText (mijenja se iz tipa Int u String). U varijablu gearText povučen je UI element tekst u koji služi za pokazivanje broja skupljenih zupčanika, uz njega je dodan i UI element slike u kojeg je stavljen sprite grafike korisničkog sučelja. Kod dodira zupčanika oni nestaju sa scene. Kako bi funkcija radila na scenu je dodani sprite zupčanika s malom animacijom te je bio označen oznakom „Collectable“. U prozoru hijerarhije, zbog preglednosti, izrađen je prazni objekt koji sadržava sve zupčanike koji se nalaze na sceni.

```
Unity Message | 0 references
65  private void OnTriggerEnter2D(Collider2D collision)
66  {
67      if (collision.tag == "Collectable")
68      {
69          pickup.Play();
70          Destroy(collision.gameObject);
71          gears += 1;
72          gearText.text = gears.ToString();
73      }
74  }
75
```

Slika 66: Praktičan dio, Funkcija za brojanje zupčanika



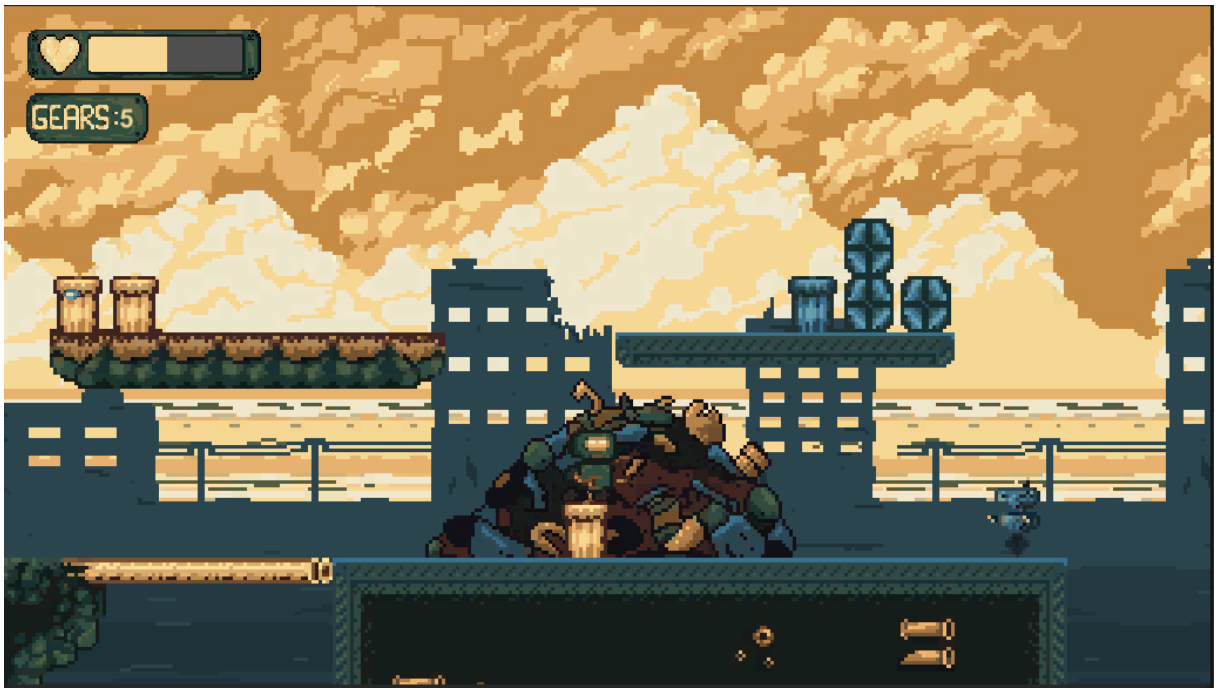
Slika 67: Praktičan dio, Izgled brojila zupčanika unutar igre

7.8.2. Traka zdravlja

Traka zdravlja postavljena je pomoću izrađenog spritea i komponente klizača koji se prilagođuje brojčanoj vrijednost zdravlja glavnog lika. Prvo dio kod izrade trake je prilagođavanje klizača unutar praznog prostora spritea te stavljanje svega u jedan poseban objekta kako se prilikom mijenjanja rezolucije igre ne bi odvojili. Klizač je UI komponenta čiju vrijednost je moguće manipulirati pomoću skripta, stoga je bilo potrebno izraditi skriptu koja spaja zdravlje lika s vrijednošću klizaća. Skripta je spojena na klizač te su u nju dodane dvije funkcije, prva funkcija prilikom pokretanja igre postavlja vrijednost klizača na maksimalno zdravlje igrača dok druga funkcija mijenja tu vrijednost ako se vrijednost zdravlja promjeni.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
        Unity Script | 1 reference
6   public class HealthBar : MonoBehaviour
7   {
8       public Slider slider;
9
            1 reference
10      public void SetMaxHealt(int health)
11      {
12          slider.maxValue = health;
13          slider.value = health;
14      }
15
            2 references
16      public void SetHealth(int health)
17      {
18          slider.value = health;
19      }
20  }
21
```

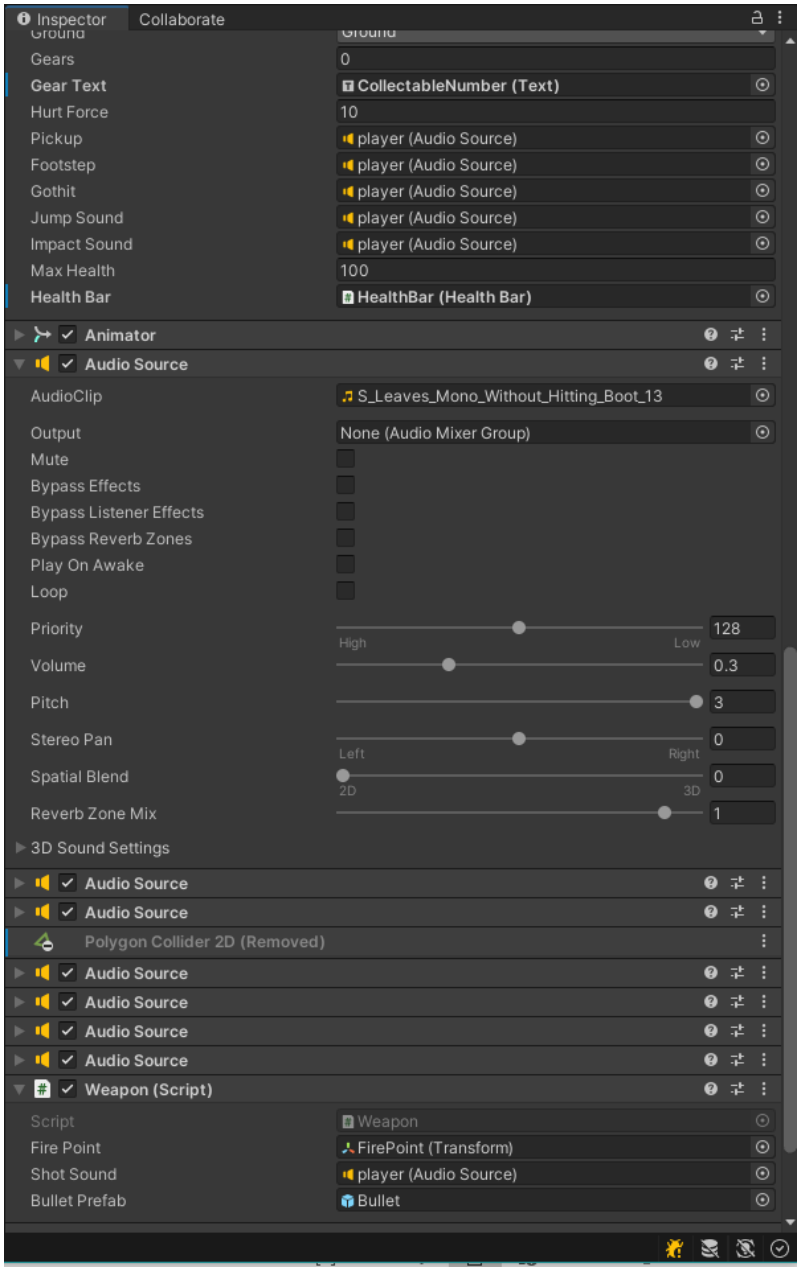
Slika 68: Praktičan dio, Skripta za prikaz zdravlja igrača



Slika 69: Praktičan dio, Izgled trake zdravlja unutar igre

7.9. Dodavanje zvukova

Svi zvukovi korišteni u radu bili su preuzeti besplatno s Unity-evog Asset Stora, neki od njih uključuju pozadinsku glazbu, zvukove hodanja, zvukove pucanja igrača i neprijatelja, zvukove interakcija igrača s objektima i tako dalje. Svi oni dodani su na objekte pomoću komponente Audio Surce te su u skriptama, preko varijabla, postavljeni na određenim i odgovarajućim funkcijama. Jedan od jednostavnih primjera toga je zvuk pucanja igrača koji je dodan u skriptu Weapon te se aktivira svaki puta kada igrač pritisne input za pucanje. Objekt player na sebi sadrži najviše zvukova jer se većina radnja odvija oko njega. Kod nekih komponenta Audio Source svojstva su bila prilagođena kako bi zvukovi odgovarali osjećaju igre. Najčešća mijenjana svojstva bila su Pitch i Volume.



Slika 70: Praktičan dio, Prikaz svih dodanih zvukovaa na objekt "player"

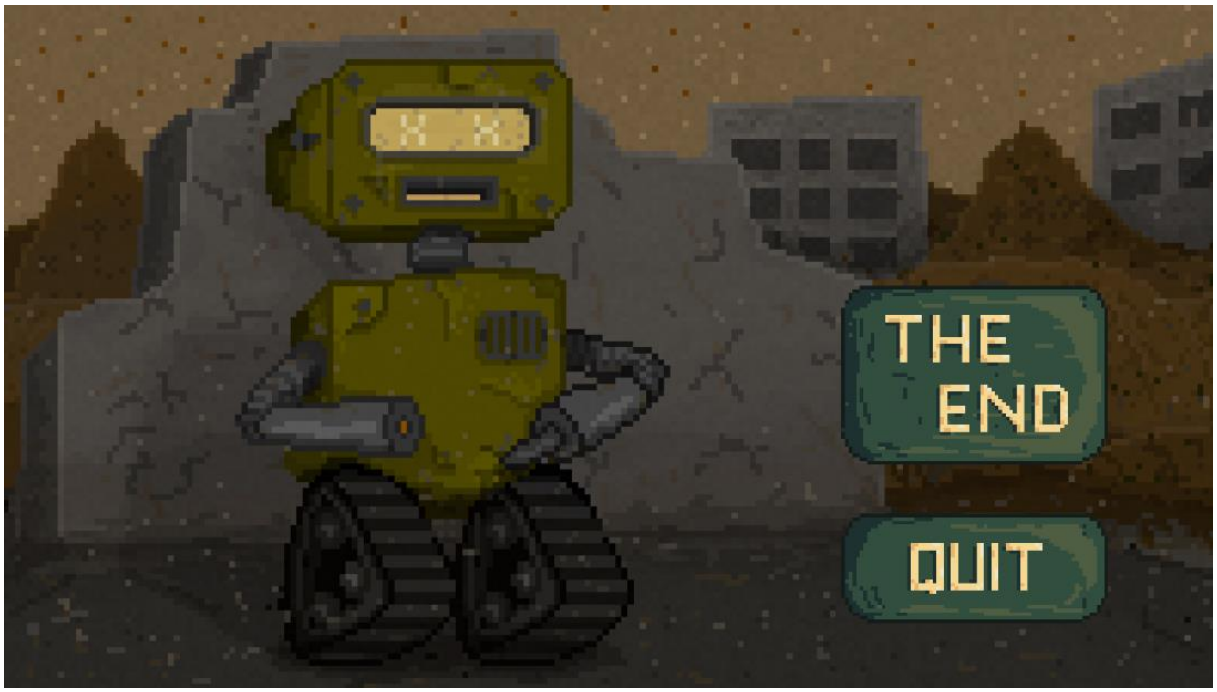
7.10. Dodatne scene

Zadnji dio rada uključivao je izradu dodatnih scena kao što su glavni izbornik i ekran završetka nivoa. One su izrađene pomoću UI komponenata te su na glavni izbornik dodana dva gumba, „Play“ i „Quit“. Spriteovi gumba su bili vlastito izrađeni te je uz pomoć komponente button na njih dodan efekt kada igrač prolazi mišem preko njih. Izrađena je skripta MainMenu koja je sadržavala dvije funkcije, funkciju Play() koja je mijenjala scenu na glavnu odnosno na scenu izrađene igre te funkciju Quit() koja je namijenjena za izlaz iz aplikacije. Skripta je pridružena na UI objekt koji sadrži oba gumba te se njezine funkcije pozivaju pomoću svojstva komponente gumba. Scena za završetak nivoa sadržavala je izrađenu grafiku „THE END“ i gumb za izlaz iz aplikacije koji samo kopiran iz scene glavnog izbornika. Na glavnu scenu igre dodani su još dva dodatni objekti koji su sadržavali skripte koje prilikom interakcije s glavnim likom aktiviraju scenu završetka nivoa i prilikom smrti, odnosno pada s platformi, aktiviraju scenu glavnog izbornika.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
        Unity Script | 0 references
6   public class MainMenu : MonoBehaviour
7   {
        0 references
8       public void PlayGame()
9       {
10          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
11      }
        0 references
12      public void QuitGame ()
13      {
14          Debug.Log("quit!");
15          Application.Quit();
16      }
17  }
18
```

Slika 71: Praktičan dio, Skripta MainMenu

Slika 72: Praktičan dio, Glavni izbornik

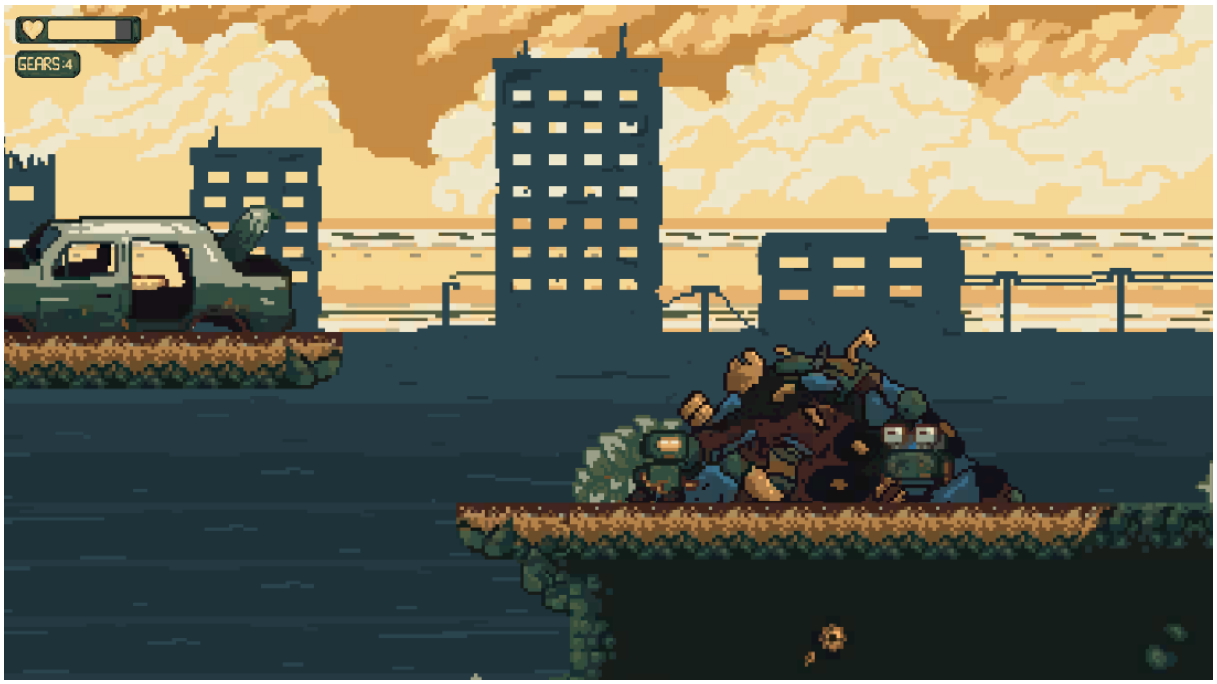


Slika 73: Praktičan dio, Ekran završetka nivoa

7.11. Završni prikaz nivoa

Slika 74: Praktičan dio, završni prikaz nivoa 1

Slika 75: Praktičan dio, završni prikaz nivoa 2



Slika 76: Praktičan dio, završni prikaz nivoa 3

Slika 77: Praktičan dio, završni prikaz nivoa 4

8. Zaključak

U današnje doba industrija video igara raste eksponencijalno te potražnja za novim originalnim idejama i svjetovima koje pružaju video igre nikad se neće smanjiti. Sve više i više ljudi video igre koristi kao svoj primarni način medijskog zadovoljstva i načina na koje provode svoje slobodno vrijeme. Video igre služe kao prijenos karaktera i zanimljivih priča te igraču omogućuju istraživanje novih svjetova koji nemaju nikakve granice. Stoga video igre mogu se smatrati kao opsežni tip medijske umjetnosti.

Iako postoji nekoliko popularnih motora za izradu video igra, Unity-ev cilj je prema korisnicima postaviti prijateljski pristup i jednostavnim dizajnom sučelja te jednostavnim načinom rada kojeg može naučiti svatko tko je zainteresiran u to područje. Unity je poznat kao motor u kojemu su izrađene neke od vrhunskih 2D video igara, poput Cuphead-a i Hollow Knight-a.

Izrada ovog rada dokazala je da jedna osoba može, jednostavno uz trud i inspiraciju, dizajnirati okvir odnosno jedan nivo 2D video igre te prikazat čitatelju mogućnosti i korake kod toga procesa.

U Varaždinu, _____ _____

Potpis studenta

9. Literatura

[1] »2D platform video games,« Gamicus, 2017. [Mrežno]. Available: https://gamicus.fandom.com/wiki/2D_platform_video_games.

[2] B. W. Michelle Menard, »Game Development with Unity, Second Edition,« CENGAGE Learning, 2015. [Mrežno]. Available: https://freepdf-books.com/game-development-with-unity-2nd-edition-pdf-books/.

[3] E. Peckham, »How Unity built the world's most popular game engine,« Tech Crunch, 17 10 2019. [Mrežno]. Available: https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAMyq7SrdgOVfPaTc6GFRZm0HJwXapdF67aEiVCbOfg-u5wD2UB-YwaLEMdosyWdIhkXQw8QAXFgMksOrLEOr1bG.

[4] »Unity Timelines,« Popular Timelines, 2021. [Mrežno]. Available: https://populartimelines.com/timeline/Unity-Technologies.

[5] J. Haas, »A History of the Unity Game Engine,« 2014. [Mrežno]. Available: http://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf.

[6] M. Staff, »The chaos of democracy,« MVC/DEVELOP, 7 7 2015. [Mrežno]. Available: https://www.mcvuk.com/development-news/the-chaos-of-democracy/.

[7] J. Bowey, »Learning to use the Unity 2D game engine,« 5 2017. [Mrežno]. Available: http://gamewithus.ca/wp-content/uploads/2017/05/learning-unity-2d.pdf.

[8] A. Sinicki, »What is Unity?,« Android Authority, 20 3 2021. [Mrežno]. Available: https://www.androidauthority.com/what-is-unity-1131558/.

[9] »Unity Manual,« Unity Technologies, 2021. [Mrežno]. Available: https://docs.unity3d.com/Manual/UnityManual.html.

[10] V. Pereira, »Learning Unity 2D Game Development by Example,« Packt Publishing, 2014. [Mrežno]. Available: https://freepdf-books.com/gpdf-learning-unity-2d-game-development-by-example/.

[11] J. W. Sue Blackman, »Unity for Absolute Beginners,« Technology in Action, 2014. [Mrežno]. Available: https://freepdf-books.com/unity-for-absolute-beginners/.

[12] A. Thorn, »Learn Unity for 2D Game Development,« Technology in Action, 2013. [Mrežno]. Available: https://freepdf-books.com/learn-unity-for-2d-game-development/.

[13] A. Thorn, »Pro Unity Game Development with C#,« Technology in Action, 2014. [Mrežno]. Available: https://freepdf-books.com/pro-unity-game-development-with-c/.

[14] A. Thorn, »Mastering Unity Scripting,« Packt Publishing, 2015. [Mrežno]. Available: https://freepdf-books.com/download/?file=3672.

[15] »Coding in C# in Unity for beginners,« Unity, 2020. [Mrežno]. Available: https://unity3d.com/learning-c-sharp-in-unity-for-beginners.

[16] D. Calabrese, »Unity 2D Game Development,« Packt Publishing, 2014. [Mrežno]. Available: https://freepdf-books.com/vpdf-unity-2d-game-development/.

[17] »Sprite,« FANDOM, 2021. [Mrežno]. Available: https://graphics.fandom.com/wiki/Sprite.

10. Popis slika

Sveučilište
Sjever

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Luka Žugec *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Izrada nivoa 2D računalne igre u programu Unity *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Luka Žugec
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Luka Žugec *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Izrada nivoa 2D računalne igre u programu Unity *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Luka Žugec
(vlastoručni potpis)