Kreiranje modularne animacije u programskom okruženju Spriter Pro te implementacija riješenja u 2D računalnoj igri

Hatlak, Dorja

Sveučilište
Sjever

Završni rad br. 531/MM/2017

Kreiranje modularne animacije u programskom okruženju Spriter Pro te implementacija rješenja u 2D računalnoj igri

Student
Dorja Hatlak, 5434/601

Mentor

dr.sc. Andrija Bernik, pred.

Varaždin, rujan 2018. 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, oblikovanje i primjenu

PRISTUPNIK: Dorja Hatlak | MATIČNI BROJ: 5434/601

DATUM: 07.06.2017 | SMJER: Video animacija

NASLOV RADA: Kreiranje modularne animacije u programskom okruženju Spriter Pro te implementacija rješenja u 2D računalnoj igri

NASLOV RADA NA ENGL. JEZIKU: Creating of modular animaiton in IDE Spriter Pro and solution impleмntation in 2D computer game

MENTOR: Andrija Bernik | ZVANJE: dipl.inf

ČLANOVI POVJERENSTVA:
1. mr.sc. Vladimir Stanisavljević, v.pred. - predsjednik
2. mr.sc. Dragan Matković, v.pred. - član
3. pred. Andrija Bernik, dipl.inf. - mentor
4. pred. Snježana Ivančić Valenko, dipl.ing. - zamjenski član
5.

Zadatak završnog rada

BROJ: 531/MM/2017

OPIS:

Rad će se bazirati na principima modularne animacije koja ne konstruira animaciju u sažetku svaki frame kao kompletnu sliku, već je konstruirana od nekoliko malih, višestruko upotrebljivih slićica (kao što su: dijelovi tijela). Modularna metoda animiranja ima mnogih prednosti u nekoliko aspekta razvoja računalnih igara kao što su: ušteda vremena, iterscije - odnosno mogućnost lako i brze promjene jedne slićice (primjerice, glave) drugom slićicom, fleksibilno prilagođavanje slićica, puno varijacija različitih ili istih karaktera, velika ušteda datotečnog prostora i tako dalje. Princip modularne animacije u Spriteru također omogućava kreiranje neograničenog broja "varijabli" za svaki karakter, što znači da je lik skup animacija, primjerice "idle" animacije i nekoliko dinamičkih poze/animacija koje će se pojavljivati u igri - skok, trčanje, pucanje i slično. Ova ideja biti će potkrijepljena konkretnim primjerom gdje će svaki korak biti pokazan i opisan. Prvi korak jest ideja, zatim slijedi crtanje tijela po dijelovima (svaki dio tijela na zasebnom layeru) u programu baziranom na vektorskoj ili piksel grafici, importiranje istih u Spriter Pro, rigging proces, povezivanje slićica sa kostima, animiranje, kreiranje karakterne mape, završetak projekta te na kraju integriranje animacija u Unity platformu. Također biti će prikazana i objašnjena osnovne operacije u Unity programu za kreiranje scene te pisanje i dodavanje skripti za kontrolu karaktera pisane u programskom jeziku C#.

U radu je potrebno:
1.) Objasniti programsko okruženje Spriter Pro, opisati korisničko sučelje, alate, princip animacije te opisati prednosti i nedostatke u usporedbi sa ostalim softverima na tržištu alata za razvoj animacija za računalne igre.
2.) Izrada 'sprite-ova', izrada animacije, kreiranje bazičnog koncepta igre i vizualnih elemenata.
3.) Kreiranje idejnog rješenja (platformska 2D igra).

ZADATAK URUČEN: 23.06.2017 | POTPIS MENTORA: Bernik

SVEUČILIŠTE SJEVER

Sveučilište
Sjever

Odjel za Multimediju, oblikovanje I primjenu

Završni rad br. 531/MM/2017

Kreiranje modularne animacija u programskom okruženju Spriter Pro te implementacija rješenja u 2D računalnoj igri

Student

Dorja Hatlak, 5434/601

Mentor

dr.sc. Andrija Bernik, pred.

Varaždin, rujan 2018. godine

Predgovor

U ovom radu ću prikazati radni tijek samostalne izrade modularne 2D animacije od samog početka, dakle od crtanja svih dijelova tijela, postavljanja istih i njihovo spajanje u kostur te u konačnici i animiranje u programu Spriter Pro, te implementaciju i povezivanje istih animacija u programu Unity Editor 5.0. Rad će biti podijeljen u dva ključna dijela. Prvi je izrada animacija a drugi izrada novog projekta u Unity Editoru te primjena tih istih animacija u njemu, povezivanje animacija s kontrolama, u grafičkom sučelju ali i putem skripti pisanih programskim jezikom C#. Osim toga, prikazat ću i neke druge osnovne funkcije Unity Editora kao što su izrada levela u igri te njegove scenografije, primjenu fizike na igrača ali i ostale, statične objekte u sceni, kreiranje generičnih funkcija u platformskoj igri kao što su sakupljanje objekata i ispis progresa na grafičkom sučelju. Na kraju rada također će biti objašnjeno kako dovršiti igru za testiranje ili distribuciju, a finalni produkt će biti samostalna 2D „side scroller“ igra s glavnim menijem i jednim levelom u kojem kontroler upravlja igračem iz trećeg lica te ima mogućnost proći ili izgubiti taj level.

Sažetak

Osnovna svrha ovog rada je izraditi kompletan i samostalan projekt u obliku 2D računalne igre te pritom pokazati principe izrade osnovnih karakternih animacija kao što su hod i skok te tehnička izrada istih u programu Spriter Pro. U konačnici će tim animacijama biti dana svrha i one će biti testirane i implementirane u igri budući da su za to i namijenjene, pa je najbolji način ostvarenja toga cilja izrada jednostavne platformerske igre s mogućnošću kontrole prethodno napravljenog karaktera i na njega vezane animacije. Kvalitetna i kinematički točna animacija, centralni je dio svake igre, a njena kvaliteta odlučuje o cjelokupnom dojmu gotovog projekta. Kako bi karakter imao neki smisao, bit će smješten u prikladnu scenu, nacrtanu istim dvodimenzionalnim stilom, kako bi dobio svoju priču. Proces izrade svake, pa čak i jednostavne igre poput ove, dugotrajan je proces koji nema točno određenu formulu pa zahtijeva mnogo promišljanja, smišljanja logičkih rješenja, isprobavanja i višestruko ponavljanje istih radnji u svrhu boljeg rezultata.

Ključne riječi: 2D igra, C# programski jezik, Modularna animacija, Side-Sroller igra, Sprite objekt, Objektno orijentirano programiranje, Parallax efekt

Abstract

The basic purpose of this paper is to create a complete, stand-alone project in form of a 2D computer game, and to demonstrate the principles of making basic character animations such as walking and jumping and also making technical performance of creating animations. All of the animations will be tested and implemented in the game in order to demonstrate their purpose. The best way to accomplish this goal is to create a simple platformer game with the ability to control the previously created character and related animations. Good quality and kinematic accuracy of animation is the central part of every game, and its quality decides on the overall impression of the finished project. In order for the character to have some meaning, it will be placed in a suitable scene, drawn in the same two-dimensional style, to get its story. The process of making even simple games like this is very time-consuming without a specific formula but requires a lot of thinking, designing, making logical solutions, testing, and multiple repetitions of the same actions in order to obtain optimal results.

Key words: 2D game, C# programming language, Modular animation, Side-Sroller game, Sprite object, Object-oriented programming, Parallax effect

Popis korištenih kratica

IK Inverse Kinematics
Sintagma za obilježavanje inverzne kinematike u robotici ili računalnoj animaciji

FK Forward Kinematics
Sintagma za obilježavanje prednje kinematike u robotici ili računalnoj animaciji

2.5D Dva i pol dimenzionalna perspektiva
Pojava koja označava 2D grafička projekcija te na nju vezane tehnologije koje uzrokuju da se slike ili prizori ponašaju, ili izgledaju kao da simuliraju trodimenzionalan prostor

API Application Programming Interface
Sintagma koja označava softversko sučelje koje omogućuje dvjema aplikacijama međusobnu razmjenu podataka

CGI Computer Generated imagery
Označava sintagmu koja se odnosi na specijalne efekte, najčešće generirane u 3D okruženju i primjenjivane u filmovima, televizijskim programima, reklamama, printanim medijima i video igrama

UI User Interface
U industrijskom dizajnu označava prostor u kojem se odvija interakcija između ljudi i strojeva (hrv. Korisničko sučelje).

Sadržaj

1. Uvod

Video igra je svaka elektronička igra koja uključuje interakciju s korisničkim sučeljem u svrhu generiranja vizualne povratne veze na nekom video uređaju kao što je TV zaslon ili računalni monitor. Riječ video u sintagmi „video igra“ referira se neki uređaj s rasterskim prikazom slike, no u novije vrijeme može se odnositi na bilo koji uređaj koji može proizvesti dvodimenzionalnu ili trodimenzionalnu sliku. Elektronički sustavi koji se koriste za igranje video igara poznati su kao platforme, a glavna podjela platformi odnosi se na osobna računala i konzole. Te platforme mogu varirati od velikih računala s opsežnim ulazno izlaznim uređajima s mogućnosti opterećenja na odvojenim motorima, pa do manjih ručnih uređaja kao što su pametni telefoni. [11]

Esencijalni dio gotovo svake računalne igre je interakcija, a preduvjet za interakciju često je animacija. Dok u igrama može biti animirano sve, od razno raznih objekata, tekstova, specijalnih efekata i grafičkog sučelja, osjećaj interaktivnosti i angažiranosti često nam daje upravo onaj objekt kojeg sami pokrećemo, a to je najčešće neki karakter, bilo da se radilo o RPG-ovima, avanturama, pucačkim, borbenim ili sportskim igrama, simulatorima ili strategijama, neovisno o žanru, rakursu gledanja ili bilo kakvoj drugoj kategorizaciji igara.

Ako gledamo povijest video igara, gledamo i povijest računalne tehnologije. Veza između tehnoloških fenomena kao što su računala i kulture nije jednostavna. Moglo bi se reći da tehnologija određuje kulturu ili da kultura određuje tehnologiju, no najrazumnije bi bilo protumačiti ovu vezu kao povijest zajedničkih utjecaja tehnologije i popularne kulture u kojoj tehnologija inspirira, odnosno omogućuje kulturološki razvoj, a kulturni razvoj može također inspirirati razvoj tehnologije. Ovu tvrdnju dobro potkrepljuje činjenica da su prve računalne igre originalno napravljene za projektiranje opreme vojnih snaga kao i u akademske svrhe. Prva video igra na svijetu napravljena je u znanstvenom laboratoriju. Najstarija video igra u Sjedinjenim državama patentirana je 1947 godine. U to vrijeme patent je nazvan katodni zabavni uređaj (Cathode Ray Tube Amusement Device). [11]

Ako pogledamo prvu računalnu igru s vizualnim prikazom ikad, a to je OXO (križić kružić), te najnovije tehnologije u svijetu igara kao što su virtualna realnost, možemo vidjeti da je većini tih igara, bez obzira bile one prikazane na osciloskopu ili VR naočalama, zajednička upravo animacija. [11]

Samo jedna od mnogih danas popularnih tehnologija za animaciju je 2.5D ili pseudo 2D animacija, također zvana i modularna animacija, koja zapravo nije nikakva novost budući da su isti principi korišteni još 70-ih godina 20. stoljeća. No, prodor indie video igara na tržište i popularizacija 2D grafike do koje je došlo zasićenjem postignutim stupnjem fotorealizma u filmovima i igrama, otvorila je prostor za ponovni razvoj starih tehnologija te adaptacija istih u

nove tehnološke mogućnosti. Samo neki od softvera koji koriste tu tehnologiju a namijenjeni su za izradu animacija za igre su Spriter, Spine, Toon Boom Harmony, Marionette Studio, DragonBones Pro, Anima2D i još mnogi drugi.

2.5D animacija govori sama za sebe, a označava 2D slike koje su animirane 3D tehnologijom i kinematikom jer koriste inverznu kinematiku, što će biti bolje objašnjeno kroz rad. [2]

2. Spriter Pro softver za kreiranje modularne animacije

Program Spriter Pro omogućava modularan način animiranja, što znači da je animacija konstruirana od mnogo malih sličica za višekratnu uporabu, kao što su, primjerice, dijelovi tijela. Dakle, svaki frame nije zasebna slika. Sve te sličice koje se koriste za konstrukciju cijelog frame-a, mogu se rotirati, skalirati ili transformirati na drugi način. Modularna metoda animiranja nudi mnogo prednosti, no svakako ima i svoja ograničenja. [2]

Animacija je širok pojam, a varira, osim u metodama, i u odgovarajućim namjenama. Animacija u filmovima i animacija u igrama imaju mnogo različitosti. Dok se animatori u filmovima moraju pobrinuti da određena scena izgleda dobro u jednom rakursu, gledana iz određene perspektive pod određenim kutem koji je određen kamerom, zadatak animatora u igrama jest da animacija bude vidljiva, te da pravilno izgleda, iz svakog mogućeg kuta koji igraču može biti vidljiv, a to ovisi o tome ima li igrač mogućnost manipuliranja kamerom, je li igra 2D, 3D, 2.5D, ima li igra jednu od 3 moguće aksonometrijske projekcije (izometrijska, dimetrijska ili trimetrijska projekcija). Ako je igra 3D, razmatra se i je li gledana iz prve perspektive, treće perspektive, je li u pitanju fiksirana 3D grafika gdje se prednji objekti renderiraju u pravom vremenu dok je pozadina statična i tako dalje. [2]

Modularna animacija samo je jedna metoda animiranja, te se koristi isključivo za interaktivne namjene, odnosno igre. Modularna animacija rijetko je samostalna, te se često kombinira s drugim metodama tijekom razvoja igre. Kod 2D igara, osobito onih koje spadaju u grupu side scrolling igara, svakako je najbolja i najefikasnija metoda. [2]

2.1. Klasična podjela vrsta animacija

Kako bi razumjeli što je modularna animacija i generalno animacija u igrama, bitno je najprije razumjeti što uopće jest animacija, koje sve vrste animacije postoje, kako se ona kroz povijest razvijala i zašto i kada koristiti modularnu animaciju, te kako je i s kojim metodama možemo kombinirati, pa u nastavku slijedi kratki uvid u tipove animacija.

2.1.1. Tradicionalna animacija

Tradicionalna animacija je rukom crtana animacija, također zvana i "cel" animacija (zbog korištenja celuloidne prozirne folije). Sastoji se od zasebnih, rukom crtanih sličica koje zajedno tvore animiranu sekvencu. Brzo projicirane jedna iza druge, tako tvore iluziju pokreta. Tradicionalna animacija, s jedne strane animacija općenito, ima vrlo dugu povijest koja se proteže

čak od 5000 godine prije Krista. U današnje vrijeme, tradicionalna animacija radi se najčešće preko računala. Sličice se crtaju na računalnom tabletu, a obično se animira po principu 12 sličica u sekundi, s povremeno bržim akcijama od 24 sličica po sekundi. Za "nove" tradicionalne animacije koriste se programi kao što su Adobe Flash, Toon Boom Studio/Harmony, TVPaint i Photoshop. [7]

2.1.2. 2D animacija bazirana na vektorima

Vrlo često, pojam 2D animacija koristi se za tradicionalnu animaciju, no zapravo je to animacija koja je generirana preko računala, a može koristiti metode tradicionalne animacije. Ova vrsta animacije uzela je zamah tijekom 90-ih godina 20. stoljeća, kada su animatori počeli koristiti program Flash i slične tehnologije. No samom početku, zbog vrlo ograničene propusnosti, te su animacije bile namijenjene za web, relativno kratke i ograničene. Upravo ta ograničenja uvjetovala su velikom apelu mnogih nezavisnih umjetnika i animatora. Renesansno doba 2D animacije (vrijeme nakon tzv. mračnog doba animacije, počinje krajem 80-ih i završava početkom 2000-ih), obilježava značajno povećanje tehničke kvalitete animacija i umjetničke slobode (zbog spomenutih ograničenja tadašnjih tehnologija), te animatori ponovno dolaze do točke umjetničkog poštovanja prema animaciji koja nije bila viđena od zlatnog doba animacije (Zlatno Disneyevo doba). Najpopularniji softveri koji se trenutno koriste za 2D, vektorski baziranu animaciju su Adobe Flash, Adobe After Effects, Toon Boom Studio, Toon Boom Animate, Toon Boom Harmony i Adobe Animate. [7]

2.1.3. 3D računalna grafika

Funkcionira na potpuno drugačiji način od tradicionalne animacije, iako zahtijeva isto razumijevanje principa pokreta i kompozicije dok su tehničke vještine drugačije za svaki pojedini zadatak. Dok tradicionalna animacija zahtijeva od animatora da bude izuzetan tehnički crtač, računalna 3D grafika ne. Sličnija je igranju lutkama nego crtanju. 3D grafika, također zvana i CGI ili samo CG, napravljena je generiranjem slike pomoću računala (render). Karakterna 3D animacija ima sličnosti sa stop motion animacijom budući da se obje bave animiranjem i pozicioniranjem modela, dok još uvijek zadržava pristup "frame-by-frame", no pruža mnogo više kontrola budući da se radi u digitalnom prostoru. Likovi u 3D animaciji modelirani su u programu a potom spojeni za kostur koji služi za pokretanje modela. Animacija se radi pozicioniranjem modela i određivanjem ključnih key frameova, dok računalo generira interpolacija između tih ključnih frameova stvara pokret. Kada je modeliranje i animiranje gotovo, računalo renderira

svaki frame zasebno, što može uzeti mnogo vremena, a to ovisi o kvaliteti slika i broju poligona. 3D animator se, za razliku od tradicionalnog animatora neće zamarati ručnim "in-betweeningom" već će se baviti podešavanjem krivulja koje predstavljaju brzinu ili pokret određenih dijelova tijela. Također, za razliku od tradicionalne animacije, dijelovi tijela uvijek su "tu" iako možda nisu vidljivi u sceni u nekim trenucima. Na primjer, u tradicionalnoj animaciji pogled sa strane neće uključivati prednji, stražnji i drugi bočni dio tijela jer oni, tehnički uopće neće postojati, dok će u 3D animaciji cijeli model biti prisutan u scenu, bez obzira jesu li oni vidljivi kamerom ili ne. Još jedna bitna razlika je frame rate. Dok tradicionalne animacije koristi svaki parni frame (svaka 2 framea nacrtan je novi crtež, što se također naziva i "2's"), 3D animacija renderirana je posebno za svaki frame te pokret uvijek teče glatko. 3D animacija definitivno je promijenila cijelu industriju animiranog filma i igara. Čitava stvar započela je 1995 s animiranim filmom Toy Story koju je direktirao John Lasseter. No ipak, tada to nije bila baš potpuno nova stvar budući da se 3D grafika već koristila u reklamama, igrama i kratkim animiranim filmovima, no ipak je Toy Story bio dugometražni 3D film "high - end" kvalitete. Od tada počinje era težnji ka foto realizmu, računala postaju brža a standardi se penju sve više. U kombinaciji sa sve boljim video kamerama, to rezultira filmova kao što su Lord of The Rings, Gladiator, Planet of Apes, Avatar i ostalim visokobudžetnim filmovima. Najprestižniji softveri za ovaj oblik animacije su Autodeskovi softveri (3DS Max, Maya, Softimage), open source program Blender, te Cinema 4D koji postaje sve više prihvaćen i korišten zbog intuitivnog sučelja te mogućnošću rada direktno sa Adobeovim After Effectsima (nakon puštanja Cineware plug-ina koji radi direktno između 4D-a i AE, bez potrebe prethodnog renderiranja). [Knjiga Andrije Bernika]

2.1.4. Grafika pokreta ili „Motion Graphics“

Pokretna grafika ili Motion Graphic je termin vezan s reklamama, aplikacijskim animacijama, logotipovima, TV-om, natpisima u filmovima, spotovima ali i webom. U suštini, pokretne grafike su dijelovi video snimke ili animacije koje su napravljene kako bi stvorile iluziju pokreta. Najčešće se kombiniraju sa zvukom te zajedno čine multimedijalnu cjelinu. Vještine potrebne za animiranje pokretne grafike ne zahtijevaju nužno znanje i razumijevanje karakterne animacije i tjelesnog mehanizma, ali zahtijevaju određene vještine i znanja vezanih uz kompoziciju, tipografiju, pokrete kamere i dizajnom. Proces stvaranja pokretne grafike zavisi od korištenog softwarea budući da različiti programi za video editiranje imaju različita korisnička sučelja. Takvo video editiranje može se koristiti bilo gdje, a najčešće podrazumijeva animiranje slika, teksta i video klipova, gifova, info grafika, prezentacija, titlova, ikona, dodavanje posebnih efekata na video klipove, te u većini slučajeva znači dio post produkcije koji se ne odnosi na kolor korekcije.

Nekad davno, kada se još nije radilo u potpunosti putem računala, post produkcija koja zahtjeva pokretne grafike bila je iznimno težak, dugotrajan i skup posao. Danas zbog programa kao što su Adobe After Effects ili Apple Motion ovaj je proces puno brži, lakši te pruža mnogo više mogućnosti uz manji budžet. Kako na televiziji, online videima i reklamama, pokretna grafika potrebna je i u video igrama. Bilo da se radi o animiranju UI elemenata, animiranju bilo kakvih interaktivnih ikona/elemenata u igrama, posebnim efektima kao što su track motion, pokretna grafika gotovo uvijek se na neki način koristi u igrama. Osim toga, cijeli vizual igre zapravo može biti načinjen od pokretnih grafika. Zamislimo 2D puzzle igru načinjenu od vektorskih 2D nacrtanih elemenata, međusobno posloženih u kompoziciju. Ti elementi su uvezeni u SVG formatu tako da mogu funkcionirati na gotovo bilo kojem formatu, zauzimati malo memorije a pritom zadržati originalan izgled bez obzira na rezoluciju. Najpopularniji softveri za animiranje pokretne grafike su Adobe After Effects, i Cinema 4D. [7]

2.1.5. Stop animacija ili „Stop motion“

Posljednja na popisu klasičnih oblika animacija je Stop animacija (Stop motion). Stop animacija napravljena je od serija fotografija poredanih i reproduciranih u sekvencu tako da stvaraju iluziju pokreta. Stop animacija najčešće je načinjena od objekata. Svaka sljedeća fotografija drugačija je od prethodne tako da su odabrani objekti, jedan ili više njih, malo pomaknuti u prostoru. Ova vrsta animacije slična je tradicionalnoj animaciji, samo što se koriste materijali i objekti iz stvarnog života. Proces stvaranja stop animacije može biti vrlo dug, a svaki objekt mora se pažljivo pomicati za svaki frame kako bi animacija bila fluidna. Postoji više vrsta Stop motion animacije.

- **Claymotion**

Radi se tako da se na načinjenu metalnu konstrukciju kostiju glinom ručno modelira karakter koji se kasnije može pomicati kao lutka. Ovo je poseban oblik animacije, a najreprezentativniji primjeri su filmovi Frankenweenie, Coraline, ParaNorman, The Nightmare Before Christmas, Wallace and Gromit... Bitno je spomenuti i igru *Armikrog,* "point-and-click" avanturističku igru čiji su likovi i ambijenti načinjeni od prave gline.

Slika 2.1 Scena lutke iz animiranog filma „Coraline“, IMDB

- **Puppet**

Razlikuje se od *claymotion* animacije po tome što lutke nisu napravljene od gline, te mogu, ali ne moraju u sebi imati neki oblik pomičnog kostura. Ustvari je claymotion vrsta puppet animacije, a puppet animacije ne mora nužno biti stop motion, već se može odnositi na samo lutkarstvo (pupperty). Na pitanje je li lutkarstvo oblik animacije ili nije, ne postoji jasan odgovor. Dok neki tvrde da je lutkarstvo animacija u umjetničkom smislu zbog toga što je karakterima "udahnut život" te u tehničkom smislu zato što je snimljena, a klasična definicija definira lutkarstvo kao oblik performansa. Svakako se na lutkarstvo može u određenim okolnostima gledati kao na oblik animacije ili u kombinaciji s drugim tehnikama kao eksperimentalnu animaciju. Osim lutke (puppet), postoji i *muppet*, specijalan pojam osmišljen od strane Jima Hensona (direktor filma The Muppet Show), a kombinira lutke za ručno animiranje (Puppet) i marionete (Marrionette).

U zadnje vrijeme popularan oblik Puppet animacije je *Brick film* ili Lego animacija, a odnosi se na stop animaciju čiji su glavni objekti lego figurice, standardne frame rate stope od 15 FPS-a.

- **Cut out**

Ovaj oblik animacije snima se odozgo prema površini na koju su postavljene papirnate ili kartonske sličice, te se snimanjem i pomicanjem "frame-by-frame" stvara iluzija pokreta. Na ovaj način snimljena je prva sezona popularne serije South Park. Princip Cut Out animacije može se koristiti i na računalu, primjerice u programu Toon Boom Harmony. Umjesto kostura, svaki dio tijela ima samo pivot točku koja se zatim može transformirati ili rotirati. Svakako je moguće ovu metodu koristiti na vrlo jednostavnim likovima, no u normalnim okolnostima nema smisla od kada za 2D animaciju također postoji IK tehnika generiranja i animiranja kostura.

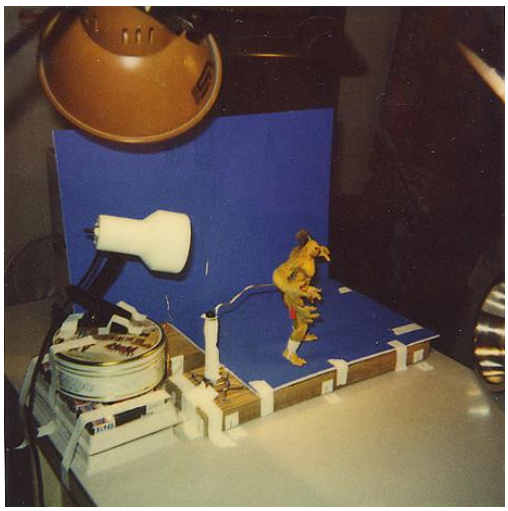
Slika 2.2 Scena iz prve sezone animiranog filma „South Park“, IMDB

- **Pixeleration**

Pixeleration je oblik stop motion animacije koji kao glumce koristi stvarne ljude i stvarne okoline. Animacija je napravljena puštanjem "mirnih slika" načinjenih fotoaparatom jedne iza druge.

2.2. Modularna animacija

Uzevši u obzir ovu klasičnu podjelu animacije, kako razumjeti što je modularna animacija? Svaki oblik animacije, u umjetničkom i tehničkom smislu razlikuje se od svih drugih po principu kreiranja, potrebnom vremenu te specifičnim namjenama. Svaki se umjetnički žanr konstantno širi, što donosi nove mogućnosti kreiranja ali i kreativnih kombiniranja tehnika. Na primjer, uzmimo da stop animacija ne postoji, John Tobias u igri *Mortal Kobat* nikada ne bi animirao Goroa stop motion tehnikom pomoću glinene lutke.



Slika 2.3 Lutka „Goro“ prilikom snimanja puppet animacije za igru „Mortal Combat“, mortalkombat.wikia.com

Također zanimljiv primjer kombiniranja staro – novog vidljiv je u akcijskoj igri *Cuphead* objavljene u rujnu 2017. Cjelokupan vizual igre temeljen je na Warner Bross/Disney stilu 30-ih godina 20. stoljeća. Detaljna i zapanjujuća pozadina koja se čini kao da je naslikana ručno vodenim bojama, te jednostavni stilizirani likovi velikih očiju ovalnog oblika, bijelih rukavica te blage i suptilne linije uz neizbježan „slapstick“ humor karakteristike su ovog stila zlatnog doba animiranih filmova.

Slika 2.4 Scena iz igre „Cuphead“, Steam

Osim u praktičnom smislu, ova kombinacija realno – stilizirano odlična je formula za lakše poistovjećivanje publike s likovima a koristi se odavno u stripovima i crtanim filmovima. Kada pričamo o animaciji, za razliku od, primjerice, klasičnog slikarstva, razvoj tehnologije ne isključuje razvoj umjetničkih žanrova. Video igre podrazumijevaju vrlo, vrlo širok spektar tehnika, žanrova, stilova, a modularna animacija u suštini je kombinacija 2.5D animacije koja može, ali i ne mora biti bazirana na vektorima, puppet animacije koja isključuje stop animaciju (iako, tehnički je moguće kreirati spriteove načinjene od pravih fotografija čovjeka ili objekata, iako to još nitko

nije napravio). I ono najbitnije, sva pravila tradicionalne animacije vrijede i za sve ostale vrste i podvrste animacije, i ne mijenja se ovisno o korištenoj tehnici.

Modularna animacija, ponekad zvana i puppet animacija ali u drugim kontekstima, u zadnjih je nekoliko godina sve popularnija metoda u području razvoja 2D igara. Osnovna je ideja izbjegavanje crtanja novih frameova za svaku posebnu akciju (npr. skok, hod, trčanje, idle). Korištenje alata koji omogućavaju animiranje spriteova iz kompozicije mnogo dijelova koristeći transformaciju, rotaciju i translataciju tih djelića na timelineu, odnosno vremenskom okviru, zapravo je vrlo ekonomično jer štedi vrijeme, prostor i „težinu“ projekta. Modularna animacija jest dakle podvrsta 2.5D računalne animacije (2.5D jer se koriste kosti). Moglo bi se također reći i da je modularna animacija oblik puppet animacije gdje se, umjesto pomicanjem žičane konstrukcije pomiču računalno generirani kinematički lanac (IK), a umjesto gline, taj se kinematički kostur povezuje s nacrtanim dijelovima tijela, koji su fizički smješteni na takozvani sprite sheet. Skraćeno, modularna animacija je pseudo 3D animacija koja se koristi za interaktivne namjene, odnosno u razvoju 2D igara.

Zanimljivo je kako se modularna animacija često kombinira s pikselastim stilom. Nekada je pikselasta grafika bila nužna, a danas definira umjetnički stil. Početkom 21. stoljeća, kada su PC igre, PlayStation i Nintendo već uzeli zamaha i kročili u budućnost, mobiteli su još uvijek bili u dobu kada nisu bili „pametni“. Zasloni tih mobitela i tehnička ograničenja zato su diktirala stilove i tehnike potrebne za razvoj igara za te mobitele. Tada se koristila ova tehnika. No nije niti to početak. Ova tehnika, kao i sami spriteovi, nisu nikakva novost. U prijelaznoj eri 16 bitnih igara, najpopularnije su bile borbene igre, te platformske igre s mnogo „bosseva“. Tehnička inovacija ove ere bilo je otkriće faux 3D ili parallax scrollinga. Ta je tehnika bila iskorak prema 3D grafici. Cijela mudrost parallax scrollinga je u tome da se pozadine kreću mnogo sporije s obzirom na kameru od likova, te se tako stvara privid dubine. Spriteovi su tada bili nužni za korištenje upravo zbog sistemskih ograničenja veličine i rezolucije spriteova, ograničenja memorije i tako dalje. Sjajno prikazani likovi u ograničenoj 16-bitnoj rezoluciji izgledali su zaista posebno, a tehnička ograničenja su, još jednom omogućila animatorima da budu kreativni i stvore poseban stil. Final Fantasy IV i The Legend of Zelda: A link to The Past bile su najmoćnije 16 bitne vizualne igra 90-ih godina. Kako su se igre sve više i više pomicale u 3D, većina 2D tehnika postale su uvelike zaboravljene, sve do nedavno. [8]

Slika 2.5 Scena iz igre Final Fantasy IV, gamefaqs.gamespot.com

Slika 2.6 Ortografska projekcija u igri „The Legend of Zelda: A link to The Past“, www.zeldadungeon.net

U današnje dobu, kada je CGI dostigao vrhunac, a realnost u grafici više nije ništa novo, zasićenje realnošću te indie-pokret vratili su staro doba. Kako svaka umjetnost uvijek teži promjenama i razvoju novih stilova, često se događa da okrenemo prošlošću, a starim tehnikama daje se „novo ruho“. Video igre nisu iznimke. Tako su spriteovi opet zaživjeli, a spojem spriteova i IK konstrukcije, nastala je modularna animacija. [8]

2.2.1. Karakteristike modularne animacije i ključni pojmovi

Iteracije – priroda modularne metode pruža animatoru da kreativno, brzo i lako mijenja izgled likova. Na primjer, recimo da se u toku igre igrač „nadogradi“ te promijeni svoj izgled. Koristeći

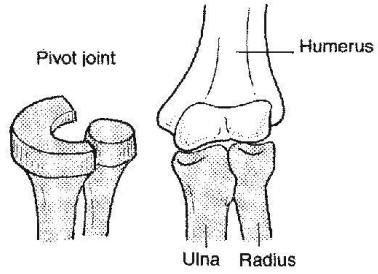
Sprite sheet, za postojeću konstrukciju lutke možemo koristiti sasvim drugi dio tijela ili nadodati novi (na primjer, oružje). Ono što je zapravo specifično za Spriter i slične programe jest da se već gotova animacija ne treba ponovno dorađivati za svaki frame. Sve što je potrebno je zamijeniti sliku, recimo, glave u sprite sheetu, a ostalo će Spriter napraviti sam.

Tweak – odnosno podešavanje animacije (ili tweening – dodatno uštimavanje pokreta između ključnih frameova).

Varijacije karaktera – kao što je već i spomenuto, alternativni karakteri, drugačiji dijelovi tijela ili nadograde, kreirani su na temelju podataka izvornog karaktera tako da u memoriji nema redundantnih i duplih podataka. Ovo nije nikakva novost u igrama, ali pomoću sprite sheetova čitav proces nadogradnje, dodatnih efekata na karakteru (na primjer power – upovi ili nova oprema) mnogo je lakši i štedi prostor

Veći i robusniji animirani likovi – animacije su glatke, jer se u memoriju spremaju samo podaci položaja, rotacije i slično za svaki „ponovno iskorišteni" dio tijela ili sprite.

Pivot točke - Pivot točka ili zaokretna točka je centar oscilacije. U usporedbi s anatomijom, pivot točka na tijelu može imati ulogu okretnog zgloba. Kod okretnog zgloba, os konveksije artikulirane površine paralelna je s uzdužnom osi kosti.



Slika 2.7 Interpretacija pivot točke kao zgloba kostura

Slika 2.8 Namještanje pivot točke na sprite u programu Spriter Pro (User Manuals)

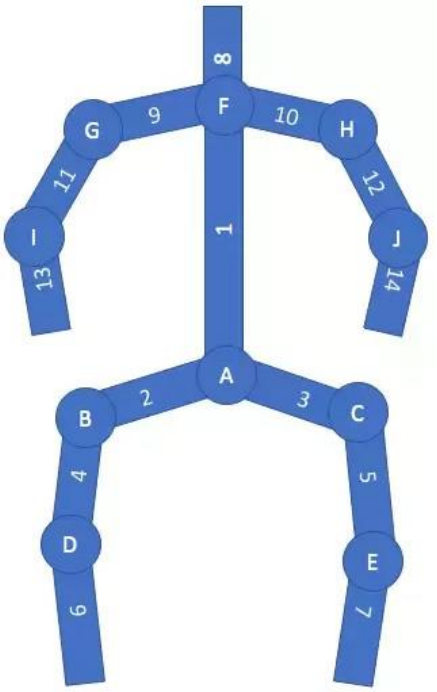
Z os – U 3D projekciji, postoje tri dimenzije, ili 3 osi – X, Y i Z. U pravilu, u 2D projekciji ne postoji Z os, no u programu Spriter pro, postoji „Z order of Sprites" i pripadajuća paleta gdje se spriteovi mogu mijenjati po Z osi. Također, Spriter omogućava da Z redoslijed za svaki frame bude drugačiji, a taj Z redoslijed zapravo se može shvatiti kao raspored spriteova po slojevima.

Rigidno tijelo - U fizici, rigidno tijelo je čvrsto tijelo u kojem je deformacija jednaka nuli, ili je zanemarivo mala. Udaljenost između bilo koje dvije date točke na rigidnom tijelu ostaje konstantna u vremenu bez obzira na vanjske sile koje djeluju na nju. To se čvrsto tijelo smatra kontinuiranom raspodjelom mase. Svako rigidno tijelo ima svoju linearnu i kutnu poziciju. Linearna pozicija je položaj jedne čestice tijela, posebno odabrane kao referentne točke, koja se obično odabire kao centar mase ili samog tijela. Kutna pozicija je orijentacija tog tijela. Osim pozicije, rigidna tijela isto tako imaju kutnu i linearnu brzinu. Linearna brzina je vektorska veličina koja je jednaka vremenskoj stopi promjene linearne pozicije tijela. Kod čistog translatacijskog gibanja (gibanja bez rotacije), sve točke na krutom tijelu imaju istu brzinu. Na primjer, linearna brzina tijela bila bi kada bi se, na primjer, podlaktica i nadlaktica kretale jednakom brzinom, bez rotacije u laktu. Međutim, kada gibanje uključuje rotaciju, trenutna brzina bilo koje dvije točke na tijelu općenito neće biti ista, a do toga dolazi, primjerice, prilikom hoda i prirodnog gibanja ruku u hodu. [2, 8]

Važno je napomenuti kako svaka znanstvena grana ima drugačije poimanje rigidnog tijela. U fizici, rigidno tijelo ne postoji ako se ne kreće brzinom bliskom brzine svjetlosti dok se u klasičnoj mehanici rigidno tijelo smatra kao kontinuirana raspodjela mase. U kvantnoj mehanici, rigidno tijelo je pak skup točaka masa. Na primjer, molekule se sastoje od tih točaka – elektrona i jezgri i na njih se gleda kao na rigidna tijela. No, što je rigidno tijelo u karakternoj animaciji?

U karakternoj animaciji možemo shvatiti rigidno tijelo kao kruto tijelo raspoređenih masa, gdje svaka od tih masa ima svoju točku deformacije ili referentnu točku (zglob) s ograničenom linearnom i angularnom pozicijom i brzinom. [8]

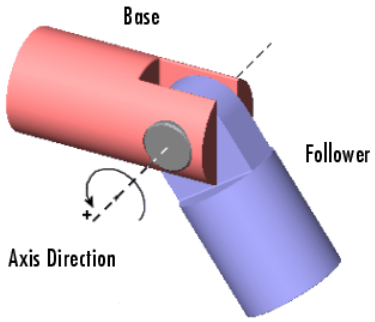
IK (Inverse Kinematics) - Inverzna kinematika (IK) je metoda animiranja koja preokreće smjer manipulacije lanca. Umjesto da radi iz korijena, to radi iz hijerarhijski nižih komponenti. Uzmimo na primjer ruku. Za animiranje ruke pomoću prednje kinematike, gornji dio ruke, ili podlaktica, rotira se od ramena, rotiranje podlaktice prati ruka, a rotiranje ručnog zgloba pomiče se cijela ruka. Svaki od tih objekata smješten je u određenu poziciju na hijerarhiji. Ruka je „dijete“ podlaktice, koja je dijete nadlaktice, koja je dijete ramena. To se naziva **parenting.** No, preduvjet za IK uvijek je artikulirano tijelo (rigid body). [8,9]



Slika 2.9 Artikulirano tijelo humanoida, quora.com

Artikulirano ili rigidno tijelo može predstavljati većinu animiranih tijela kao što su ljudi i životinje s kostima. Artikulirano tijelo može se shvatiti i kao stablo povezanih lanaca. Ti povezani lanci napravljeni su od zglobova i veza, gdje su veze rigidni cilindri. Postoje više vrsta zglobova odnosno spojki, a dvije vrste najčešće vrste su *Okretni zglob* (revolute joint) i *Prizmatički zglob*.

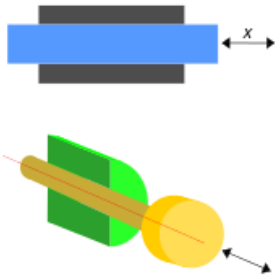
Okretni zglob je kinematički par jednog stupnja slobode korišten u mehanizmima. Okretni zglobovi omogućuju rotaciju po jednoj osovini. Primjer okretnog zgloba, odnosno spoja su šarke na vratima, mehanizmi za presavijanje i ostali uređaji s rotacijom u jednom smjeru, ali i zglobovi na ljudskom tijelu. U tom slučaju, okretni zglob može se rotirati i po više osi.



Slika 2.10 Primjer okretnog zgloba, www.revolvy.com

Prizmatičan zglob daje linearni klizni pokret između dva tijela, a često se naziva i klizač (slider), kao u spojnici klizača. Prizmatičan zglob može biti formiran s poligonalnim poprečnim presjekom tako da se odupire rotaciji. Prizmatički se zglob, dakle, ne rotira, već klizi uzduž jedne osi, i uvijek

ima gornju i donju granicu. U inverznoj kinematici, uglavnom se ne koristi prizmatički već okretni zglob.



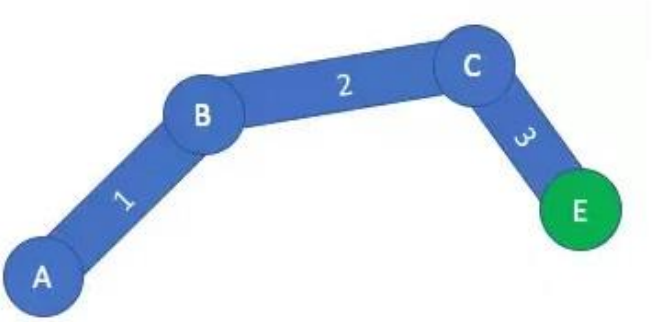
Slika 2.11 Prizmatički zglob, www.revolvy.com

Preduvjet za postizanje inverzne kinematike jest hijerarhijska struktura rigidnog tijela. Artikulirano tijelo ima svoj korijenski, glavni zglob koji je temelj cijele strukture. Kod humanoidnih tijela, korijenski zglob uglavnom je središte kukova, a artikulirano tijelo može se shvatiti kao stablo veza, koje počinje od korijenskog zgloba. Novi spoj je svaka novo račvanje stabla na grane i grančice. *Unutarnji vezni zglob* bliži je korijenu u hijerarhiji artikuliranog tijela, a *vanjski vezni zglob* je spoj koji je dalji od korijenskog zgloba i niži na ljestvici hijerarhije.

Slika 2.12 Figurativni prikaz stabla kao hijerarhijske strukture rigidnog tijela, postandcourier.com

Svaka takva hijerarhija ima svoj *krajnji efektor*, ili više njih. Na tijelu, to mogu biti prsti ili šake na rukama, te stopala na nogama. Krajnjih efektora dakle može biti više.

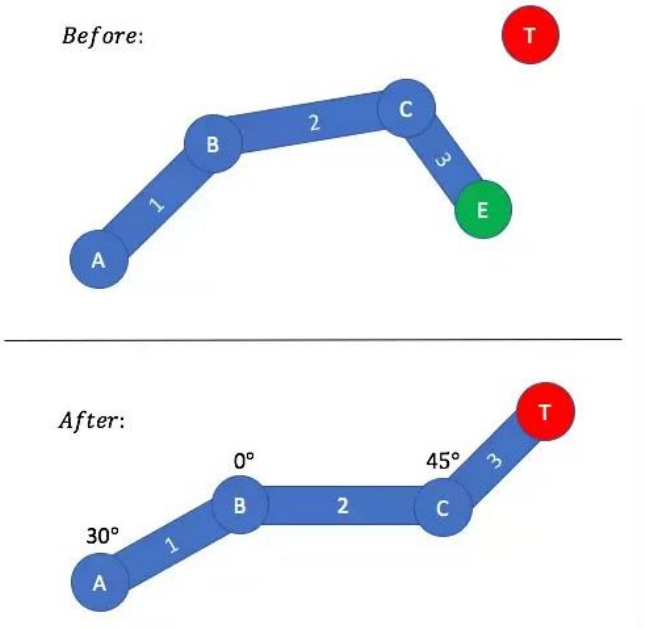
Na slici je primjer krajnjeg efektora koji je prokazan zelenom bojom. Također, u ovom je primjeru zglob A korijenski zglob, veza 1 je unutarnji vezni zglob B, a veza 2 je njegov vanjski vezni zglob. [9]



Slika 2.13 Primjer krajnjeg efektora (E), korijenskog zgloba (A), unutarnji (B) i vanjski (C) vezni zglobovi te njihove veze (1, 2, 3), quora.com

FK (Forward Kinematics) - prednja kinematika (FK) je algoritam prema kojemu artikulirana poza daje ulaz i izlaz, gdje se kao ulaz uzima korijen, a kao krajnji efektor kalkulira se izlaz. To je dakle suprotno od inverzne kinematike. Kada bi koristili prednju kinematiku, svaki se djelić rigidnog tijela treba pojedinačno podesiti kako bi računalni algoritam izračunao izlaz (output) te poze. To znači da se mora posebno definirati artikulacija svakog pojedinog zgloba i veze, što bi bilo dobro za tijela s malim brojem zglobova.

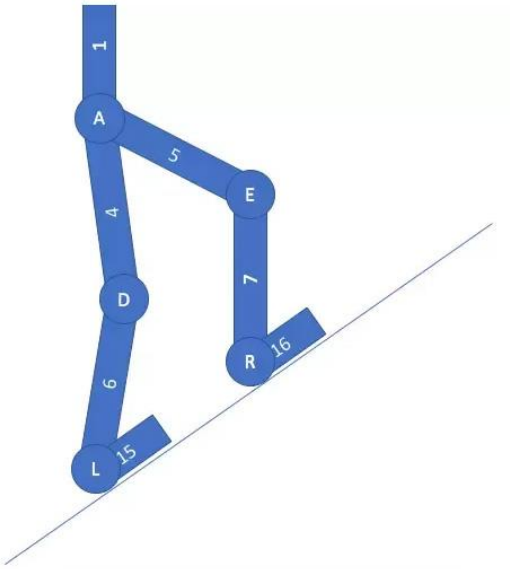
Recimo da neki pokret, na primjer, pomicanje ruke, ima svoj krajnji efektor, što jest da šaka dođe do određene točke. Koristeći prednju kinematiku, trebalo bi početi s poziranjem veza i zglobova od samog korijena, kako bi došli do dane točke. [9]



Slika 2.14 Primjer usporedbe prednje i inverzne kinematike, quora.com

Ulaz i izlaz prednje i inverzne kinematike su promijenjeni, budući da su te funkcije inverzne jedna drugoj. Dakle, funkcija prednje kinematike uzima ciljnu poziciju (na slici označeno crvenom bojom) kao input te kalkulira pozu za krajnji efektor. Cijela poza je tada output. Kod IK funkcije, nije potrebno definirati cijelu pozu, već je IK funkcija računa s obzirom na naš ulaz. Na slici iznad, vidljiva je scena prije i poslije primjene IK algoritma. Krajnji efektor je u tom slučaju povučen do ciljane pozicije, a sve ostalo je izračunato automatski.

S obzirom na spomenute razlike između IK i FK funkcija, može se zaključiti kako je IK najbolje rješenje za pomicanje rigidnog tijela. IK se može koristiti kako bi šaka stigla do cilja, ali isto tako i za noge, pomicanjem stopala. U tom je slučaju stopalo krajnji efektor koji utječe na koljeno i prepone za hod. IK je dakle najmoćniji alat za animiranje humanoida ili ostalih živih ili neživih bića s konstrukcijom kostura.



Slika 2.15 Primjer prilagodljivosti krajnjih efektora kutu nagiba, quora.com

Na prethodnoj slici može se vidjeti kako cilj nije limitiran na samu poziciju. Stopala su rotirana na neravnoj površini, a ostatak tijela pozicioniran je kao da je površina normalna, što znači da se nije cijelo tijelo prilagodilo kutu nagiba stopala. [9]

Fizika iza inverzne kinematike te njena primjena - Inverzna kinematika pronalazi svoju primjenu u robotici, dizajnu igara, 3D i 2.5D animacijama. U robotici, IK koristi kinematičke jednadžbe za određivanje parametara spojnica koji pružaju željeni položaj za svaki krajnji efektor robota. Prethodna specifikacija pokreta robota potrebna je kako bi krajnji efektor postigao željeni zadatak. To se naziva *motion planning* (planiranje gibanja). IK transformira plan gibanja zglobne pogone robota. Slične formule određuju položaje kostura animiranog karaktera koji se kreće na određen način.

Inverzna kinematika, u robotici, zapravo je vrsta kinematske analize. *Kinematska analiza* omogućuje dizajneru da dobije informacije o položaju svake komponente unutar mehaničkog sustava. Te se informacije nadalje koriste za *dinamičku analizu i upravljačke putove*. Inverzna kinematika je stoga kinematska analiza ograničenog sustava rigidnog tijela ili kinematičkog lanca. Kinematske jednadžbe robota zatim se koriste za definiranje jednadžbe petlje kompleksnog artikuliranog sustava. Te petlje su *nelinearna ograničenja* na konfiguracijskim parametrima sustava, dok su neovisni parametri u jednadžbama poznati kao *stupnjevi slobode* sustava.

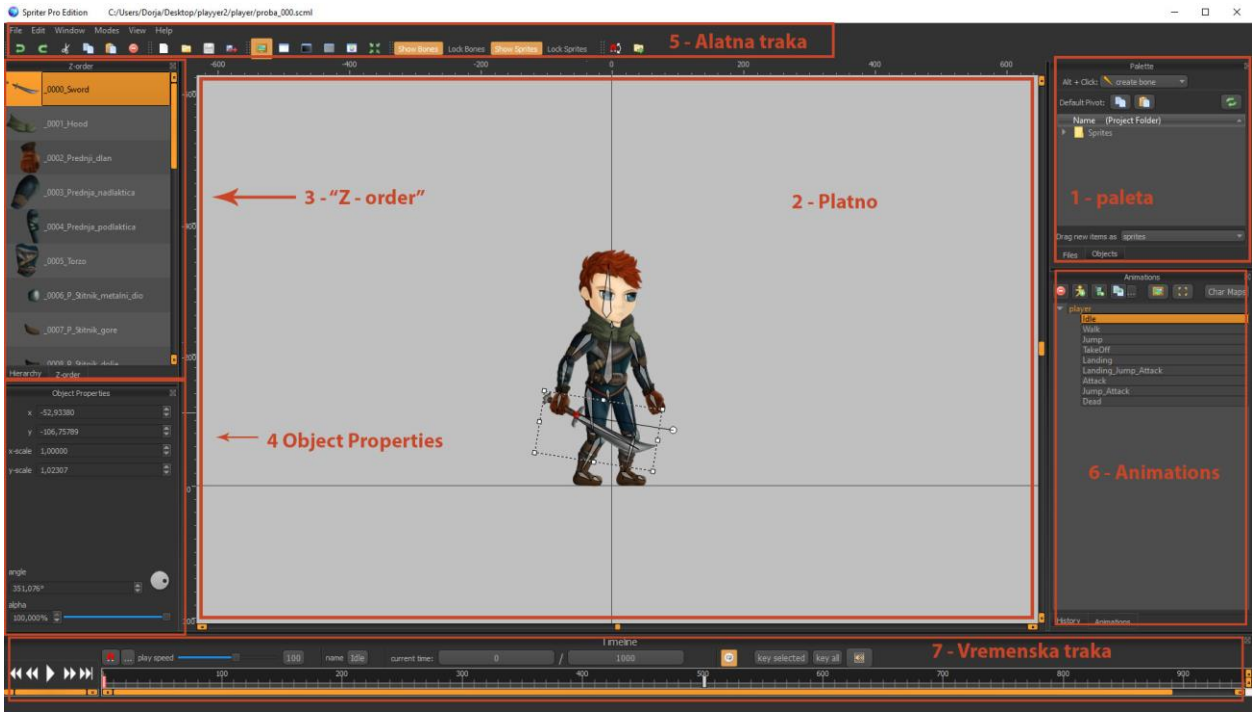
No, za razliku od robotike, kod računalnog modeliranja i animiranja alati često koriste *Newton - Raphsonovu metodu* kao rješenje za nelinearne kinematičke jednadžbe. U numeričkoj analizi, to je metoda za uzastopno pronalaženje boljih aproksimacija korijena (ili nula) stvarne vrijednosti.

To je ustvari matematička metoda pronalaženja korijena. Svaka uspješna primjena računalne animacije i IK sustava često zahtjeva i pomak tijela unutar razumnih antropomorfskih ograničenja. To se odnosi na prethodno spomenute linearna ograničenja i stupnjeve slobode, ali za čovjeka. Što znači da će i ograničenje lakta biti da se ne može pomaknuti u negativne stupnjeve jer bi to dovelo do pucanja kostiju.

U robotici i računalnom animiranju, IK algoritam je zapravo izraz zatvorenog oblika koji uzima krajnji efektor poze kao input te daje poziciju zglobova kao output, $Q = f(x)$. Cijela se ta operacija u računalnom algoritmu izračunava u milisekundnim brzinama te generira C++ kod razumljiv programima kao što su Maya, 3DS MAX, Toon Boom, pa i Spriter Pro. [9]

2.2.2. Grafičko sučelje i osnovne karakteristike programa Spriter Pro

Spriter Pro je program namijenjen isključivo za animiranje prethodno nacrtanih spriteova i grafika, tako da je prije kreiranja projekta potrebno imati unaprijed pripremljene materijale, spremljene u željenu korijensku mapu, što je kasnije u radu i demonstrirano.



Slika 2.16 Grafičko sučelje programa Spriter Pro, kreirala autorica rada

Prvi označeni dio sučelja na slici je prozor „Palette“, gdje su sve teksture s odgovarajućim mapama vidljive. Drugi označeni dio je platno, na koje se smještaju sve teksture i Spriteovi, te se ovdje ručno vrše sve transformacije. Treći ključan dio je prozor „Z – Order“. Svaki sprite iz palete

odvučen na platno pojavljuje se automatski u tome prozoru, a on određuje njihov redoslijed po slojevima, odnosno po Z osi, te je najbolji način za uređivanje toga redoslijeda.

Ispod prozora „Z -order“ nalazi se prozor „Object properties“, koji olakšava rad jer omogućuje numeričko pomicanje i skaliranje označenog sprite-a.

Grafičko sučelje programa ima još i alatnu traku s osnovnim izbornicima, ispod koje se nalaze alatna traka za brzi pristup, a sadrži ikone za lakši i brži pristup najključnijih i najkorištenijih alata kao što su: „Undo“, „Redo“, „Cut“, „Copy“, „Paste“, „Delete From All Frames“, „New Project“, „Open“, „Save“. Osim toga ovdje su i alati za promjenu boje pozadine platna, opcija centriranja pogleda, opcije za pokazivanje i sakrivanje spriteova i kostiju i tako dalje.

S desne strane nalazi se „Animations“ prozor koji služi za organizaciju i kreiranje animacija. Jednom kada je kostur napravljen i povezan sa spriteovima, konstrukciji se dodjeljuje ime entiteta. Novi entitet dodaje se pritiskom na za to određenu ikonu u prozoru, kao i nova animacija za taj entitet. U ovom primjeru, entitet je „player“ koji ima animacije idle, walk, jump i ostale. Ako u istom projektu želimo napraviti dodatnu animaciju s drugim spriteovima, primjerice efekt pucnja, stvaramo novi entitet koji potom ispočetka animiramo. Dodavanje novih animacija omogućava izradu animacije od samog početka. Najbolji način rada jest povezivanje animacija. Na primjer, najprije je napravljena „idle“ animacija“, pa se „walk“ animacija radi sljedeća. Kod kreiranja nove animacije, dobivamo prazno platno i stoga je potrebno kopirati neku od poza, s istovremeno označenim željenim frameom na kome se ta poza nalazi.

Posljednja komponenta u programu je vremenska traka, koja služi za organizaciju i kontrolu sadržaja dokumenta u vremenu kroz slojeve i frame-ove. Slojevi su vidljivi tek kada se vremenska traka poveća, a tada je vidljiva svaka kost i sprite, te se one zasebno mogu animirati kako bi se izbjegla kaotična organizacija ključnih frameova na osnovnoj traci. To je korisno kod dugih i kompleksnih animacija, no u većini slučajeva nije potrebno.

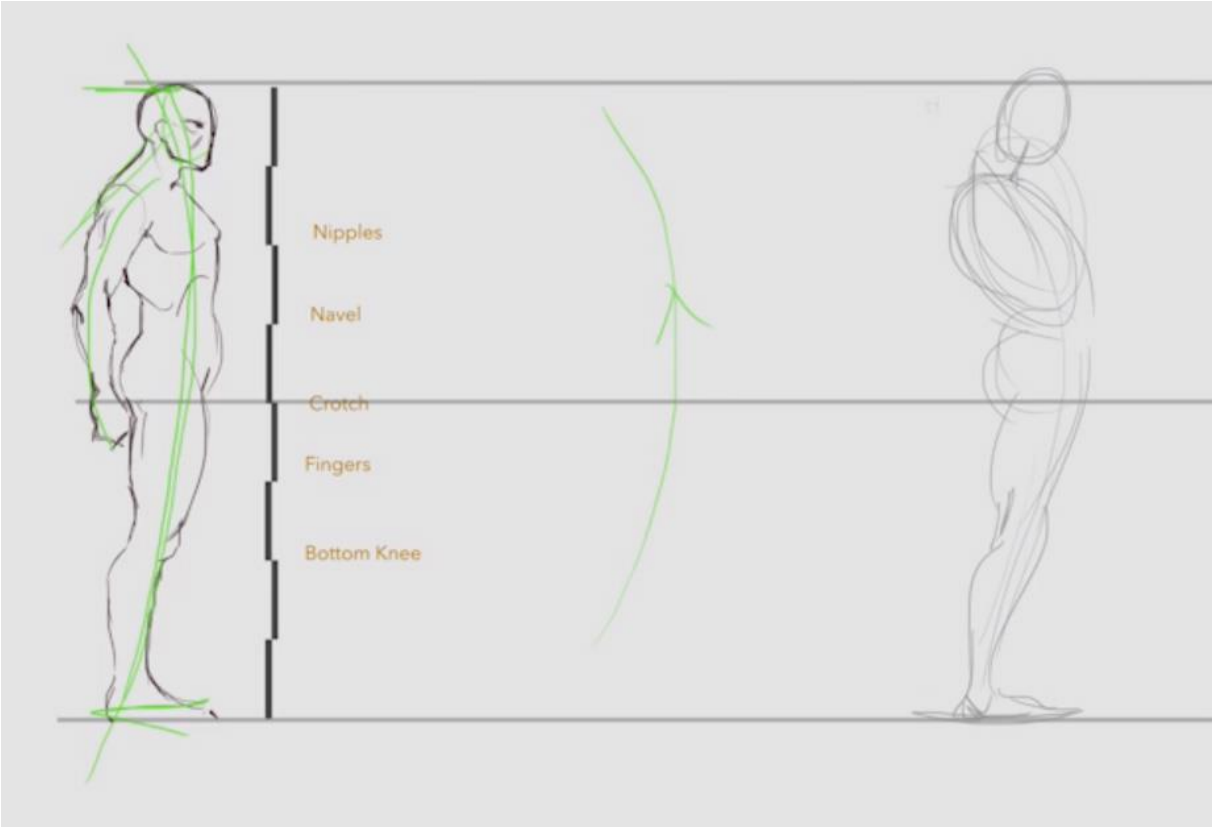
Slojevi su kao višestruke filmske vrpce složene jedna na drugu, od kojih svaka sadrži drugačiju sliku. Glavni dijelovi vremenske trake u suštini su slojevi, frameovi, klizač za navigaciju po frameovima te gumbi za reprodukciju, premotavanje animacije na početak ili kraj te gumbi za pomicanje naprijed nazad za jedan frame.

Ostale kontrole u vremenskoj traci koje pruža Spriter Pro su automatsko postavljanje ključnih frameova (ikona magnetića), klizač za brzinu animacije prilikom reprodukcije, indikatori trenutnog vremena te ukupne duljine animacije, koji se mogu mijenjati klikom i upisivanjem željene numeričke vrijednosti, a ukupna se duljina može mijenjati i podopcijom "stretch keys", što se radi ako želimo proporcionalno usporiti ili ubrzati animaciju. Ovdje su još i opcije "key all", "key selected" i "repeat playback". [2]

3. Izrada modularne animacije

U daljnjem tekstu bit će opisan postupak izrade karakterne animacije u programu Spriter Pro, principi animiranja osnovnih pokreta kao što su hod i skok te način pristupa zadatku. Osnovne animacije za igru su idle, walk, jump, a ovdje su dodane i animacije za napad (attack) i animacija umiranja. Kada se karakter izrađuje od nule, svaki je korak u radu bitan, jer svaki prethodni utječe na sve sljedeće, i upravo zbog toga je uvijek preporučljivo posvetiti mnogo vremena crtanju lika kao i animiranju, testirati i popravljati uočene pogreške.

Realistična figura mjeri se u približno 7 i pol do 8 veličina glave. Kod 2D igara i animiranih filmova, likovi se oduvijek stiliziraju, pa se taj omjer glave i tijela spušta, do najekstremnijih slučajeva čak i do 1:1. U skici karaktera koji će biti animiran za igru, glava ide u ukupnu duljinu tijela otprilike 3.5 puta. Kada gledamo skicu figure, bitno je shvatiti ključne točne na tijelu, bez obzira koliko lik bio stiliziran. Ukoliko radimo na stiliziranom liku, rezultat će uvijek biti bolji ako se pogleda na referencu realistične figure. Ako pogledamo „idealnu“ mušku figuru, dvije glave padaju na točki gdje se nalaze bradavice na prsima, kod pupka tri, kod prepona otprilike 4 glave, kraj prstiju ruku oko 5, donji dio koljena 6, i do kraja figure ostaje otprilike glava i pol do dvije, kao na primjeru ispod.

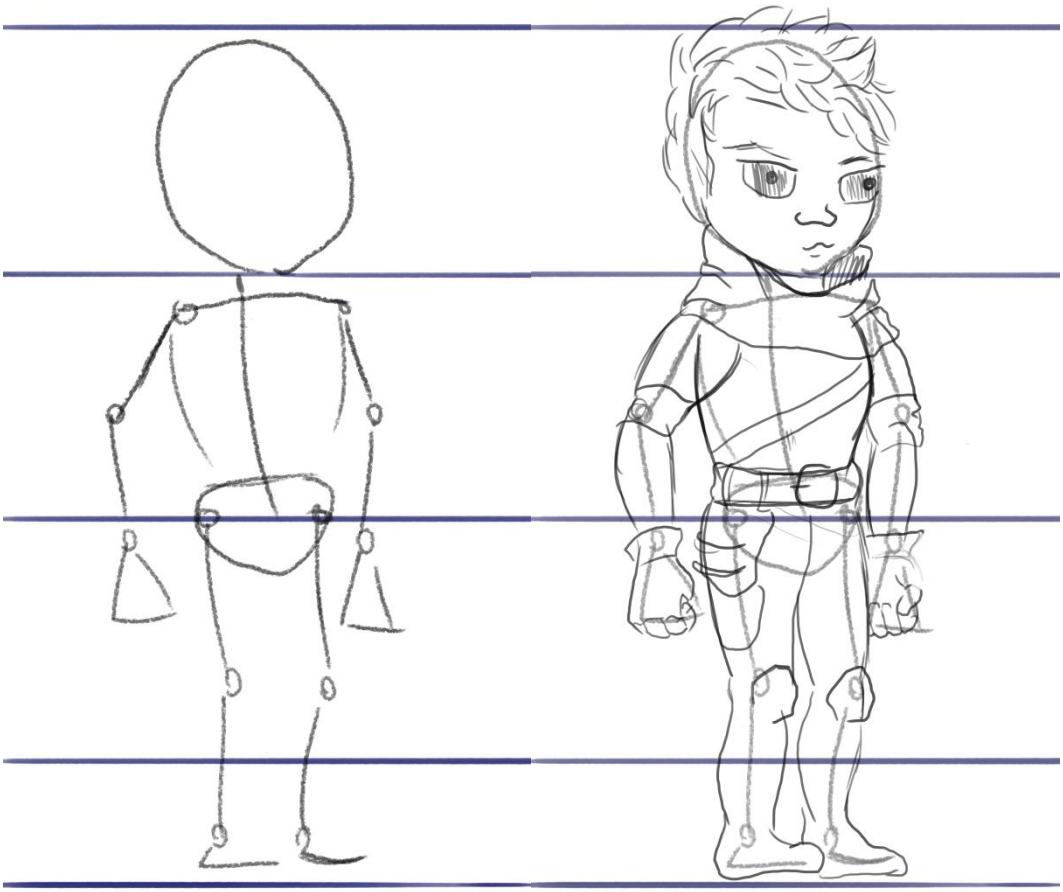


Slika 3.1 Klasičan omjer proporcija odrasle muške osobe [4]

Kada se crta figura, također je bitno uzeti u obzir krivulju na leđima leđa, jer je kralješnica fleksibilna, a krivina pozu čini dinamičnom a samim time bolje upada u oko i djeluje „življe“. Akcijska figura zato se uvijek crta od kralješnice, koja određuje i samu pozu. Na primjeru se može vidjeti kako su akcijske linije bile početna točka ove faze. Nacrtane su na jednom layeru, a zatim je ista stvar na drugom bolje definirana. [4]

3.1. Blokiranje figure u vektorima

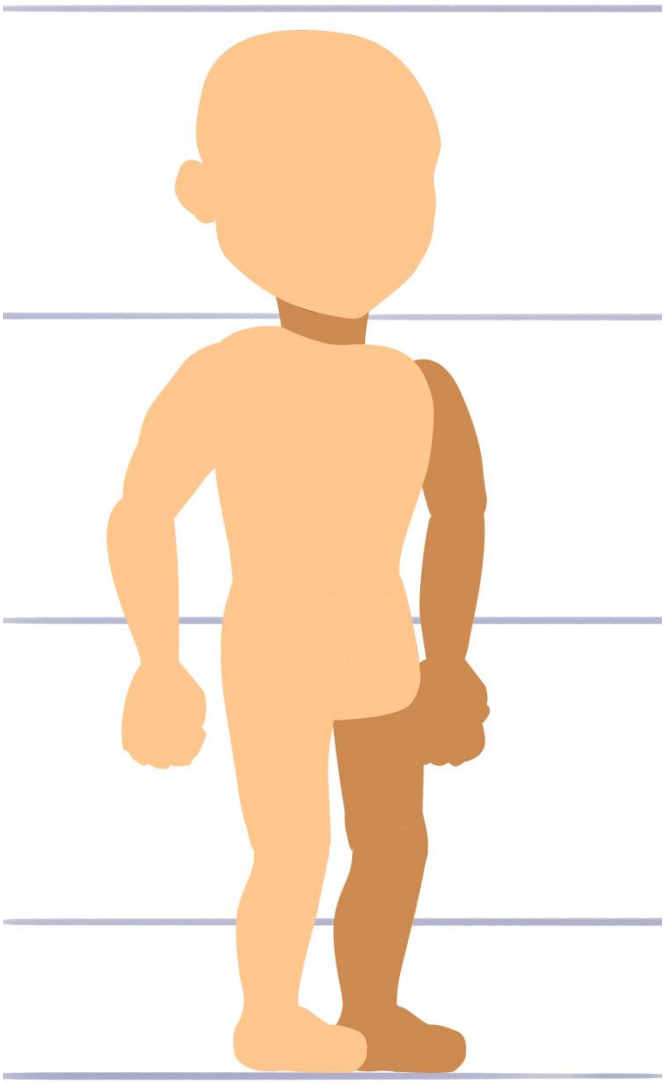
Nakon što se odluči o proporcijama željene figura, poželjno je nacrtati skicu te figure barem ugrubo, bilo na papiru ili digitalno. U ovom slučaju, odlučeno je da će glava biti malo predimenzionirana, tako da će proporcije tijela biti 1:3,5 (glava u odnosu za dužinu tijela). Prvi je korak bio samo linijski nacrtati dinamičnu pozu, a zatim tu istu pozu doraditi i ugrubo nacrtati. To je dobar i brz pristup kreiranja koncepta karaktera.



Slika 3.2 Određivanje željenih proporcija tijela karaktera, skica kostura sa zglobovima (lijevo) i skica karaktera (desno), kreirala autorica rada

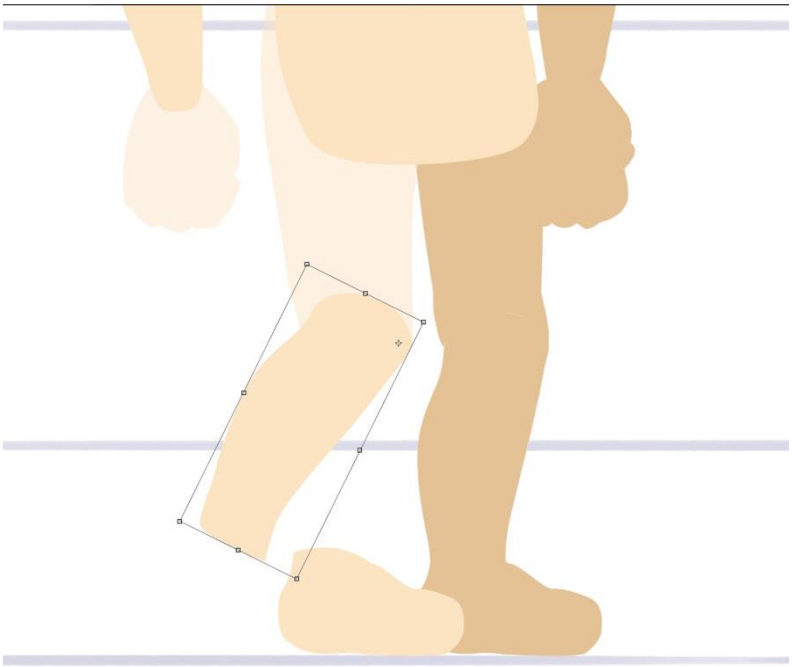
S baznom skicom na layeru ispod, drugi korak je bio blokirati dijelove tijela u vektorima u Photoshopu. Pogodan alat u programu Adobe Photoshop je *Vector Pencil (*P). Kod vektora je svakako poželjno koristiti samo nekoliko točaka (što manje) prilikom crtanja nekog oblika. Blokiranje dijelova tijela znači crtanje svakog posebno, na zasebnom layeru. Također, ako se ovaj proces radi u Photoshopu, bitno je na alatnoj traci Pencil alata na padajućem izborniku odabrati „Shape“ a ne „Path“ kako bi se izbjeglo nepotrebno ispunjavanje oblika bojom koja će ostati statična bez obzira na pomicanje vektora, a samim time i uštedjelo vrijeme.

Ono što je bitno kod blokiranja jest obratiti pažnju da se ti dijelovi tijela preklapaju tako da prilikom animiranja ne dođe do situacije da se između dva spritea vidi rupa. Na slici ispod, figura je blokirana u sljedeće dijelove: glava, vrat, torzo, zdjelica, podlaktica i nadlaktica svake ruka, natkoljenica i potkoljenica svake noge, peta i prsti svakog stopala. [4]



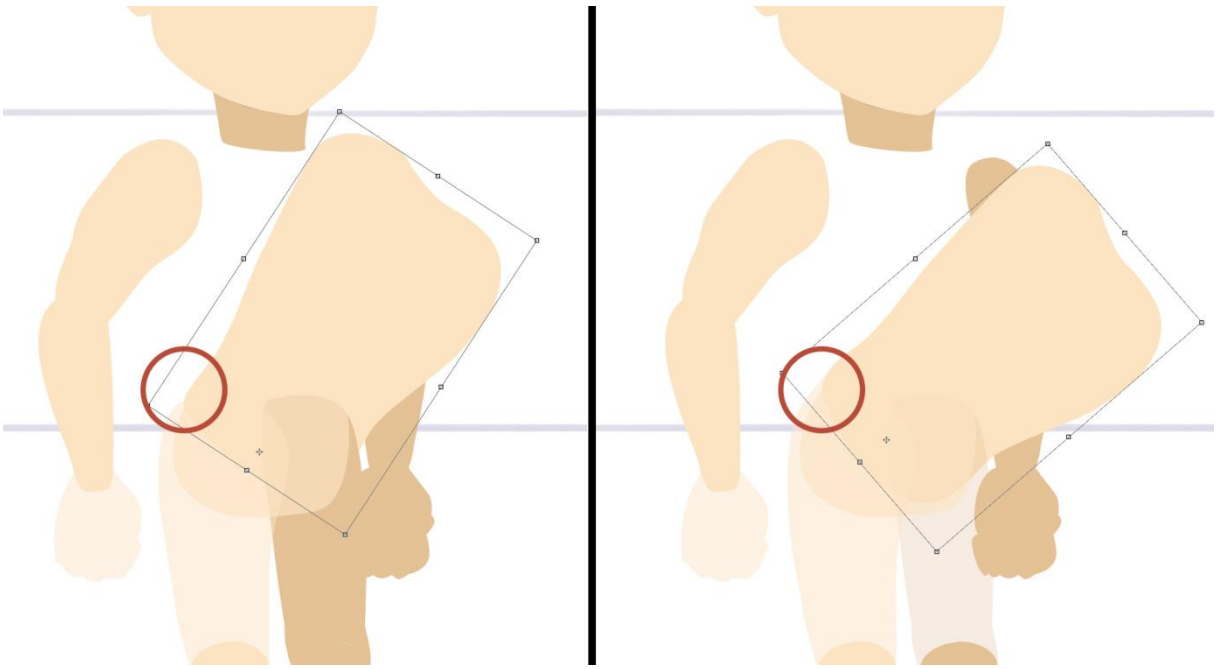
Slika 3.3 Prva verzija probne blokirane figure u spriteovima prije testiranja, kreirala autorica rada

Ova figura služit će samo za probu, te će kasnije biti nadograđena sa željenim karakteristikama lica kao što su oči, usta, nos, kosa, a odjeća će također biti nadodana. Kod ovog oblika 2D animacije, ne postoji poseban recept. Kako bi rig tijela funkcionirao, potrebno je isprobavati tako dugo dok se ne dođe do željenog rezultata. Ovo je zapravo i najdulja faza u čitavom procesu izrade animacije, jer jednom kada figura bude dobro razlomljena, animiranje neće biti teško. Pivot točkama u Photoshopu lako se može testirati rotacija spritea, iako IK još nije generiran. [3]



Slika 3.4 Testiranje rotacije sprite-a u photoshopu, kreirala autorica rada

Testiranjem svakog dijela tijela primijećeno je kako je zdjelica pre velika te takvo veliko preklapanje nije potrebno, jer pomicanjem noge ne smije vidjeti da se još nešto nalazi ispod. Također je primijećeno kako je prilikom nagiba tijela u struku vidljiva rupa između zdjelice i torza. Isto je odmah popravljeno u Photoshopu. [4]

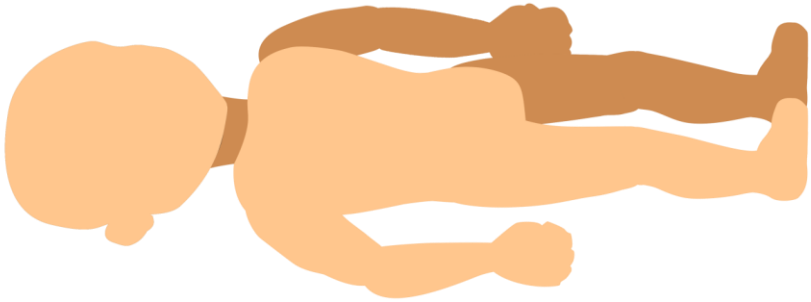


Slika 3.5 Testiranje nagiba torza te uočavanje i ispravljanje pogrešaka, kreirala autorica rada

Nakon što je blokiranje završeno, sve layere potrebno je pravilno eksportirati iz photoshopa kako bi se rig mogao testirati u Spriteru. [4]

3.2. Export rješenja iz Photoshopa u Spriter Pro

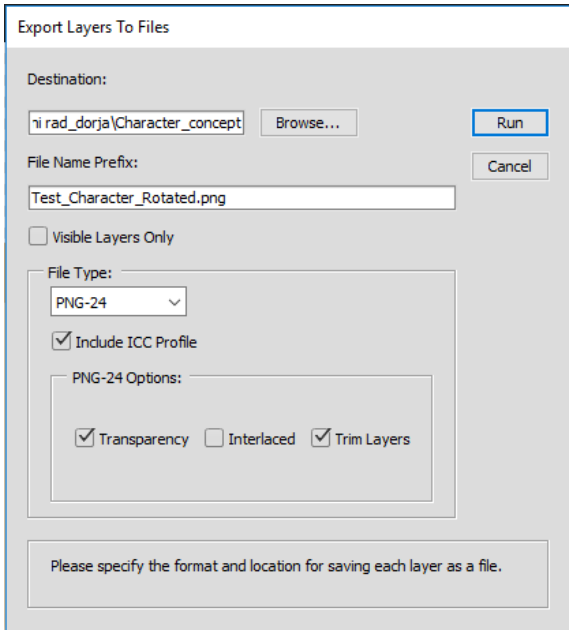
Kada se napravljeni dijelovi exportaju iz Photoshopa, najbolji je način postaviti cijelu grafičku sliku horizontalno. Tako će pikseli izgledati mnogo bolje jednom kada se uvezu u Spriter. Također, prije nego se počne koristiti skripta u Photoshopu, svaki layer mora biti pravilno imenovan, rasteriziran, a layeri ne smiju biti stavljeni u grupe te moraju biti pravilno postavljeni u hijerarhiji, odnosno na Z osi. Konverzija u piksele vrši se desnim klikom na layer te pritiskom na komandu „Rasterize Layer“.



Slika 3.6 Prva verzija probne figure za probu u Spriteru, kreirala autorica rada

Za pokretanje skripte, potrebno je u Photoshopu odabrati File>Layers>Layers to Files. Pojavljuje se dijaloški okvir gdje je potrebno specificirati tip datoteke, datotečno mjesto izvoza i odabrati prefiks imena svake datoteke. Budući da je svaki dio tijela već imenovan u layeru, prefiks može biti bilo što.

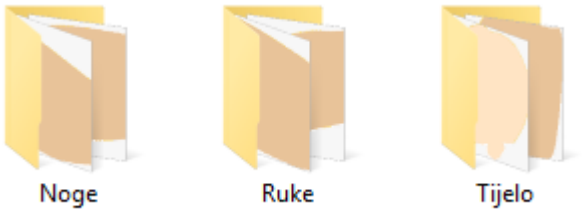
U ovom slučaju, odabran je format PNG-24, budući da figura ne sadrži HDR teksture, već samo jednostavne boje. PNG-24 također je zahvalan format jer podržava promjenu veličine. Također, „Transparency“ i „Trim Layers“ trebaju biti uključeni. Pritiskom na „Run“, pokreće se proces izvoza layera. [2, 4]



Slika 3.7 Dijaloški okvir izvoza spriteova iz Photoshopa, kreirala autorica rada

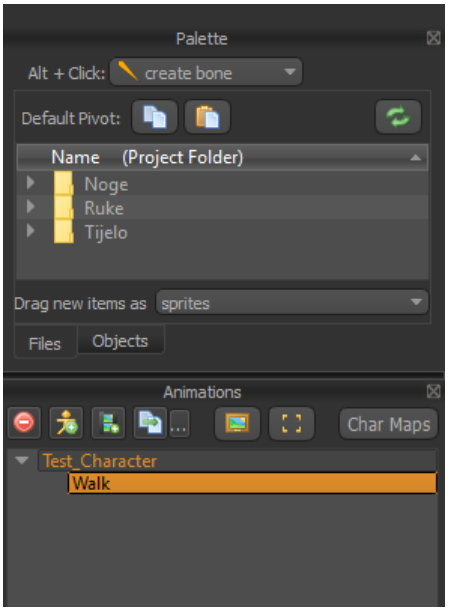
Rezultat je 18 png sličica, automatski odrezanih u Photoshopu od ruba do ruba s transparentnim pozadinama.Također, cijela je figura spremljena u jedan layer kako bi služila za referencu prilikom sastavljanja karaktera po spriteovima.

Prije nego importiramo u Spriter, kako bi si olakšali posao, dobro je podijeliti dijelove tijela u različite grupe. To pomaže lakšoj organizaciji u samom Spriteru.



Slika 3.8 Organizacija spriteova po datotekama, kreirala autorica rada

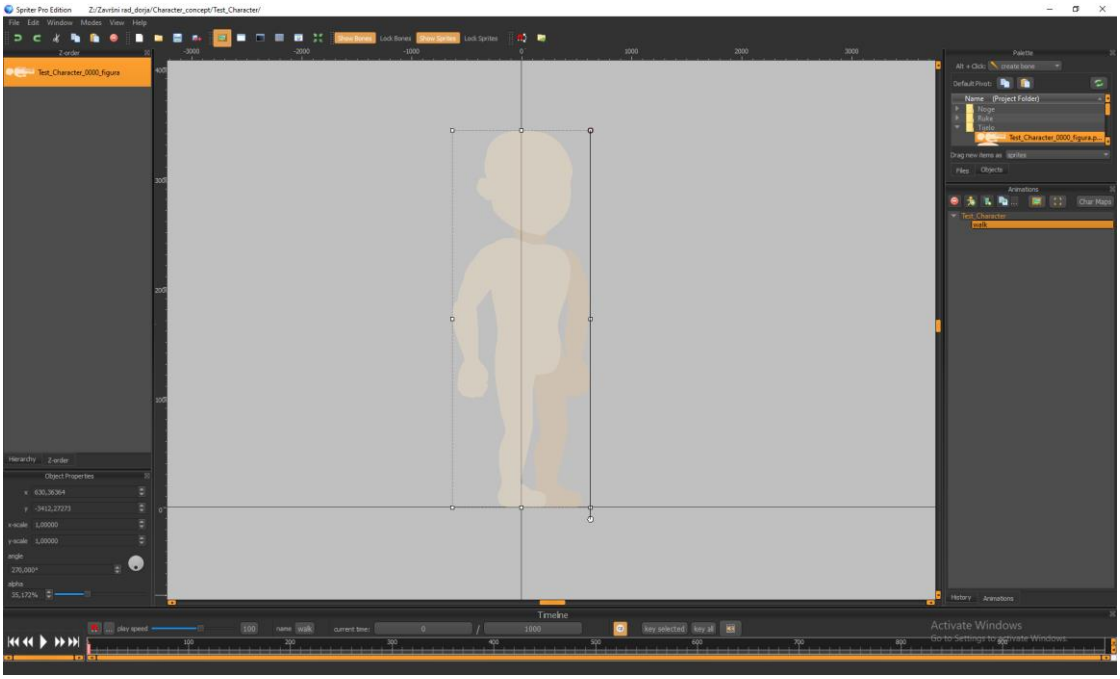
Import se u Spriteru vrši na glavnom izborniku odabirom na File > New Project. Zatim se otvara dijaloški okvir gdje se odabire korijenska mapa s dijelovima tijela karaktera. Na desnoj strani odmah se otvara okvir „Palette“ s odabranim datotekama, te se karakteru i animaciji vezanoj uz njega dodjeljuje ime.



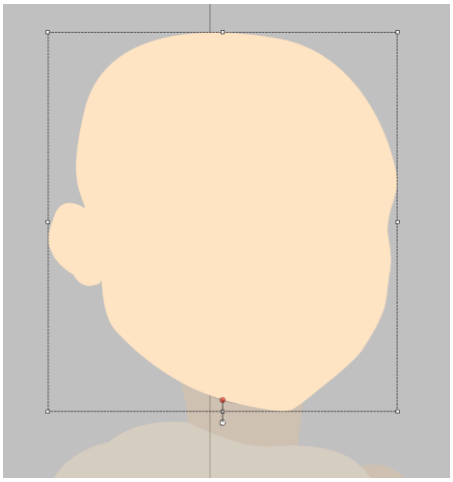
Slika 3.9 Izgled palete nakon uvoza datoteka sa spriteovima, kreirala autorica rada

Kako bi se zadržale proporcije originalne skice, najprije se iz palete na platno odvlači slika cijele figure. Za povećavanje i smanjivanje platna, služi se središnjim kotačićem na mišu, a za navigaciju se koristi tipka Space i lijevi klik miša. Nakon što je lik odvučen na platno, dno njegovih nogu odvlači se na liniju X osi, a zatim se u paleti „Character Properties“ na lijevom donjem uglu

može podešavati X, Y vrijednosti tako da upišemo brojku ili smanjujemo/povećavamo vrijednost na strelicama gore i dolje.



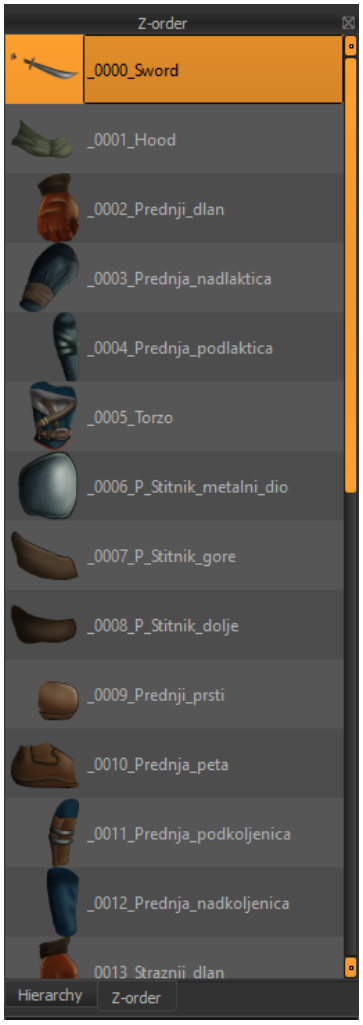
Slika 3.10 Referentna slika tijela sa smanjenom transparencijom, kreirala autorica rada



Slika 3.11 Smještanje pivot točke vrata, kreirala autorica rada

U istom prozoru (Object Properties), referentnoj slici figure dodijeljena je i njena alpha vrijednost, kako bi figura bila transparentna. Tako će biti lakše poravnati dijelove tijela s referentnom slikom.

Sljedeći je korak odvući svaki dio tijela na platno. Duplim klikom na dio tijela u paleti otvara se prozor „Edit Default Pivot“ gdje se mišem pivot točka odvuče na željeno mjesto. To je kasnije moguće i promijeniti. Pivot točku na glavi poželjno je dovući ispod linije čeljusti, ako je cijelo tijelo gledano bočno. U slučaju probnog karaktera, glava je gledana s polu profila pa je pivot točka stavljena na mjesto gdje bi se otprilike nalazila sredina vrata. Kada se glava odvuče i pozicionira na platno, pivot točka vidi se u obliku crvene točke. Ako ne odgovara, na isti način kao što se postavila, Pivot točka može se promijeniti ponovno.



Slika 3.12 Redosljed spriteova u prozoru „Z-Order, kreirala autorica rada

Pozicija pivot točke u programu se može promijeniti i na platnu, pomicanjem točke na drugo mjesto. Međutim, ona će tada promijeniti boju u bijelu, i ta će točka tada biti na tom mjestu samo za na odabranom frameu, pa je najbolje premještaj napraviti u paleti ili nakon premještanja točke odabrati točku desnim klikom i označiti „Override Default Pivot“. Druga opcija koju dobivamo desnim klikom je „Use Default Pivot“, što može biti korisno ako slučajno premjestimo točku na platnu kako bi je vratili na zadanu poziciju. Pivot točka se na vratu smješta na sredini dolje, a isto vrijedi i za torzo.

Isti postupak smještanja Pivot točke te djela tijela na platno ponavlja se za svaki dio tijela. Te eke pivot točke kasnije će se, kod testiranja rotacija svakako morati mijenjati. Na lijevoj strani u programu nalazi se paleta *Z-order*. U njoj se svaki dio tijela stavlja u određen hijerarhijski položaj s obzirom na druge dijelove. Za taj redoslijed ne postoji striktno pravilo. Primjerice, nije bitno je li glava hijerarhijski na višem položaju od torza budući da se u ovom slučaju ne preklapaju, no dakako je bitno da vrat bude ispod glave. Također, smještaj podlaktice s obzirom na nadlakticu također ovisi o tome kako je sama figura crtana, no najčešće je ona iznad. Cijela stražnja noga svakako mora biti iza torza i prednje noge (u mome slučaju ona će čak biti i iza zdjelice, što također ne mora biti slučaj). Logika iza ove hijerarhije zapravo se temelji na zdravom razumu.

Premještanje spriteova po Z osi vrši se klikom i vučenjem na željenu poziciju, ili se određeni sprite može označiti te strelicama gore i dolje na tipkovnici pomicati po osi, dok tipke lijevo i desno služe da se sprite premjesti na sam vrh ili dno Z redoslijeda. Spriter omogućava da bilo koji sprite na bilo kojem mjestu u bilo kojem vremenu mijenja u hijerarhiji (što je kasnije i iskorišteno

na animaciji napada s mačem), pa ako se Z os mijenja, bitno je to napraviti na keyframeu za željenim redoslijedom te odabrati Edit > Copy Z Order To Other Frames.

Činjenica da je tijelo lika u nepravilnom bočnom položaju, ili „kvazi“ polu profilu, cijelu priču komplicira. Nepravilan bočni položaj zato što noge sa stopalima izgledaju kao da su postavljene potpuno bočno, no razmak između njih daje iluziju perspektive i volumena tijela. Dok su obje noge „bočne“, s rukama to nije slučaj. Ovakav stil i „neobična“ perspektiva, začudo, prilično dobro izgleda i naveliko se koristi u igrama, budući da izgleda mnogo zanimljivije od obične bočne poze, a i dodaje treću dimenziju u prostor. Međutim, kod ovakve poze, zdjelica ne može biti simetrična zbog te, prividne perspektive. Zbog te zdjelice te redoslijeda Z osi, redom; prednja noga, zdjelica, stražnja noga, noge zahtijevaju različitu duljinu a time i smještaj pivot točaka na različitim razinama po horizontali. Slično važi i za ruke. Zbog toga rig i animacija hoda zahtijevaju isprobavanje i gotovo uvijek, neizbježne izmjene. Stvar će biti dodatno zakomplicirana kada se figuri nadoda i odjeća, pa više neće u pitanju biti samo jedna boja, već i linije, a potencijalno i sjene koje će se morati poklapati. Ovo je zapravo nedostatak modularne animacije, koja mnogo bolje funkcionira kod jednostavnih tijela. Konkretno u ovom primjeru, problematične točke su stopala i područje oko zdjelice, što je kasnije na finalnoj verziji ispravljeno.

Kao što je već spomenuto, zamišljeno je kako će lik biti hibridnom poluprofilu, u stilu *egipatskog frontalizma* te će se kretati samo po dvije osi u prostoru. Zato je najpametnije najprije napraviti testni walk cycle, kako bi vidjeli hoće li zamišljena konstrukcija funkcionirati. U ovom slučaju, walk cycle je bio potpuno funkcionalan, a prilikom animiranja lako se vidjelo kako oblik zdjelice treba promijeniti, što je i učinjeno. [2, 4]

3.3. Rig karaktera

Slika 3.12 Finalni izgled karaktera prije dodavanja kostiju, kreirala autorica rada

Na slici je prikazana konačna verzija izgleda lika nacrtanog u photoshopu. Dodan je još i mač koji će se cijelo vrijeme kretati zajedno s tijelom te će služiti u „attack“ animaciji. Osim mača, štitnici za koljena će također biti zasebni spriteovi budući da ne mogu biti u potpunosti vezani za potkoljenice ili natkoljenice.

Jednom kada su svi dijelovi sakupljeni a njihove pivot točke postavljene, rig karaktera počinje tako da se nacrtaju kosti. Kod humanoidnih tijela, najbolje je početi od zdjelice, jer je to glavni dio koji drži težinu tijela. Dodavanje kosti vrlo je jednostavno i intuitivno u Spriteru, a radi se pritiskom na tipku Alt, klikom i vučenjem miša u isto vrijeme. U tom trenutku stvara se kost trokutastog oblika. Ako je prva nacrtana zdjelična kost, kako bi se parenting odmah napravio,

zdjelična kost mora ostati označena kada se izvlači sljedeća, natkoljenična kost. Nakon zdjelice, povlače se kosti natkoljenice, potkoljenice, pete i prstiju. Za sljedeću nogu potrebno je ponovno označiti zdjeličnu kost te izvući ostale kosti noge. Nakon toga, zdjeličnu kost treba ponovno označiti te gore povući kost trupa, vrata, glave. Kosti ruku crtaju se pod hijerarhijom kosti trupa. Pivot točke ranije postavljene u programu neće imati nikakve veze s kosturom te služe samo kao točka iz koje se objekt transformira (na primjer, skalira), te će za kost pivot točka biti ona točke gdje kost počinje, što produljuje tijek rada jer nije moguće testirati rotaciju kosti ako kost nije primijenjena na sprite. To komplicira situaciju jer, ako primijetimo da se kost ne nalazi na istom mjestu, a čitav kostur je već primijenjen na tijelo, kost se mora ponovno odvojiti od spritea. Na primjer, u ovome slučaju primječeno je kako se stražnja natkoljenica ne rotira na dobar način. U tome slučaju mora se obrisati i ta kost, i sve njene „child“ kosti, što zahtijeva dodatno vrijeme, a i činjenica da se kost mora crtati na pamet nije najsretnije rješenje.

Kada je kostur nacrtan, ako kliknemo opet na kost, pojavljuju se strelice pomoću kojih možemo skalirati kosti. Što veće, to bolje, jer ako su kosti pre tanke, kasnije može biti teško označiti je. Kada je kostur nacrtan, slijedi povezivanje slika s kostima. To se izvršava pritiskom na tipku „B“ kada je kost označena, te klikom na sliku, odnosno sprite, ili više njih, koje želimo povezati s tom kosti. Kost koja je niže u hijerarhiji od neke druge, može biti nacrtana bilo gdje i ne mora se nužno spajati s kosti od svoje „parent“ kosti. Kost se može u svrhu editiranja ponovno označiti te kliknuti na sprite od kojeg ga se želi privremeno odvojiti na isti način kao što se sprite i povezao, klik + B. [4]

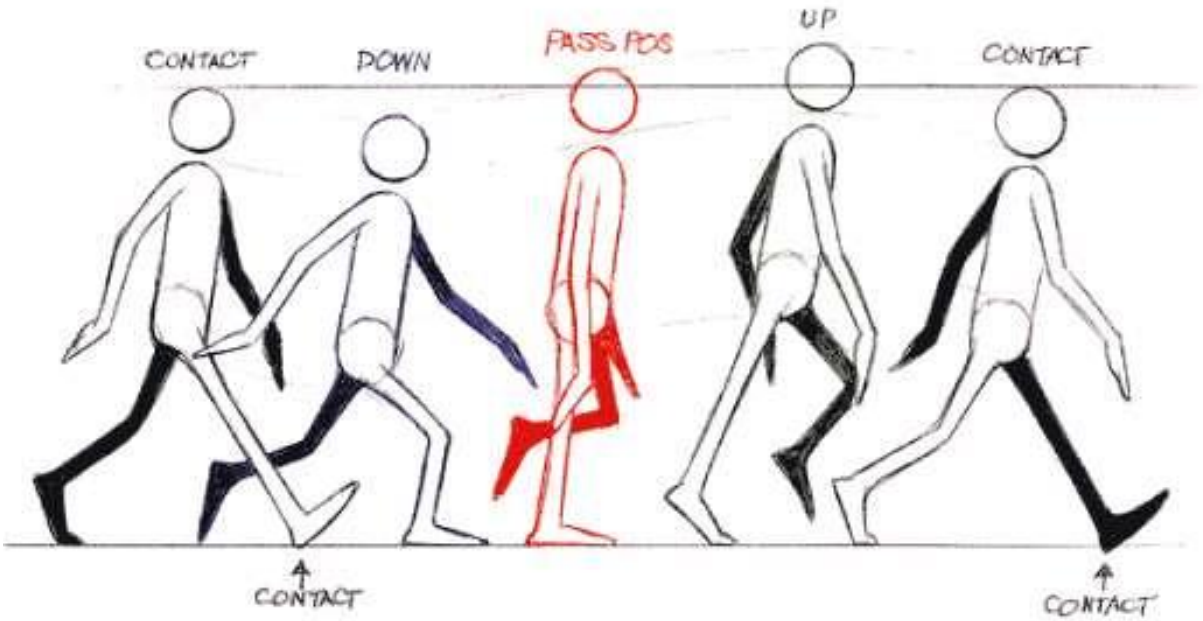
Slika 3.13Povećavanje kostiju ruke, kreirala autorica rada

Nakon mnogo ispravaka i testiranja, dobiven je rig karaktera koji dobro funkcionira, a može se vidjeti na slici ispod. Sljedeći je korak animiranje.

Slika 3.14 Prikaz kostura spojenog na karakter, kreirala autorica rada

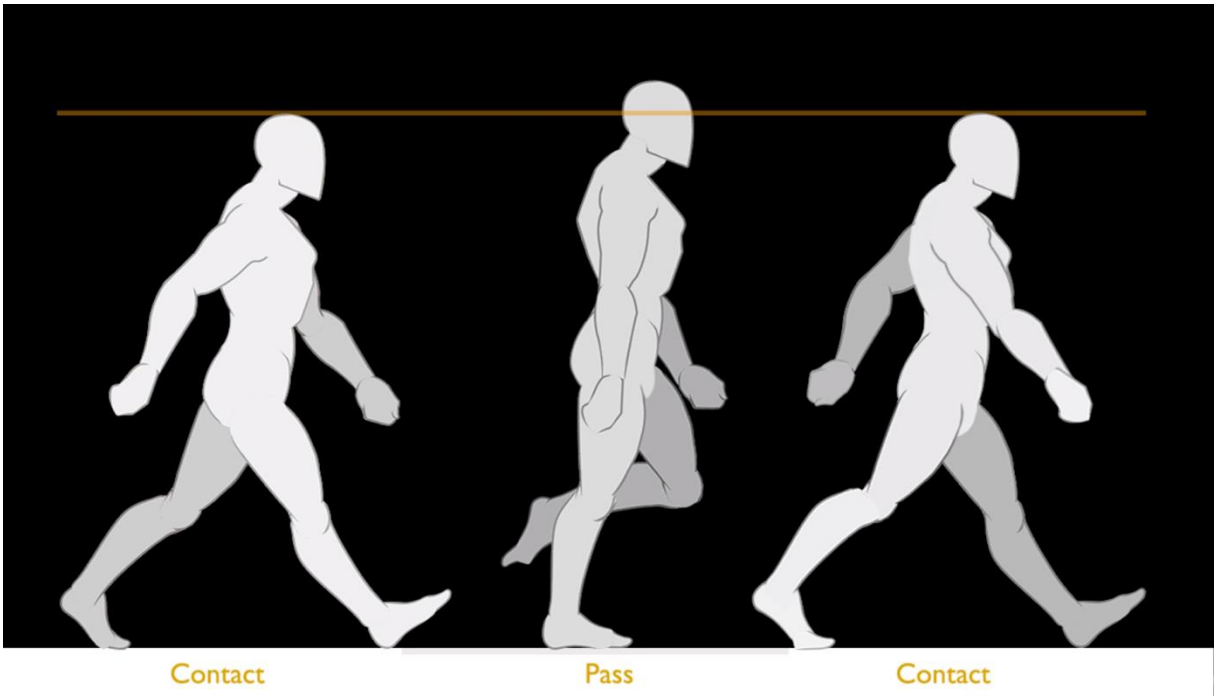
3.4. Walk cycle

Kada pratimo živu akcijsku šetnju, što se također naziva rotoskopija (eng. rotoscoping), lako se vidi što se događa s težinom tijela, no kada dođe do animiranje, neka pravila moraju se promijeniti, a to je prenaglašavanje gornje i donje poze u hodanju. Iz nekog razloga to naglašavanje daje bolji osjećaj stvarnosti. U knjizi Animator's survival kit, Richard Williams zaključio je kako pomaci gore i dolje daju najbolji osjećaj za težinu te je razlomio walk cycle u nekoliko ključnih poza. To su: eng. contact, up, down, pass position. Naime, Williams je rekao kakio dobivamo najbolji osjećaj težine tijela kada se tijelo spusti, noga savije u koljenu a cijelo stopalo na kojem je trenutno težina bude ispruženo na podu. [1]



Slika 3.15 Ključne poze za krug hodanja po crtežu Richarda Williamsa [1]

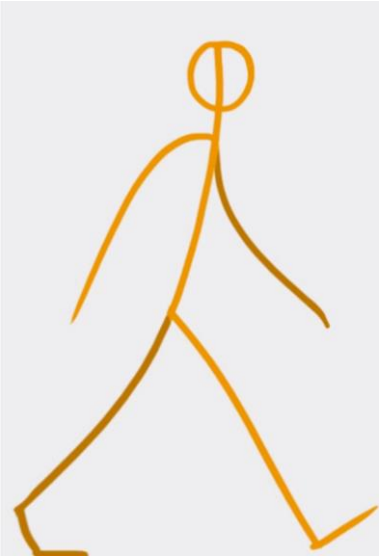
Na slici se vide ključne poze, a u konačnici će biti potrebno ukupno 8 poza, dok će se početna i prva faza u animiranju sastoji od tri ključne poze tijela. Kasnije se one preusmjeravaju na način da se mijenja položaj udova, na primjer, u drugom koraku će prednja ruka u kontakt pozi biti desna, ako je prethodno bila lijeva. Vrijedi i za noge. [1]



Slika 3.16 Dvije kontaktne i prijelazna poza kod realistične figure [3]

Ovdje se na slici vidi promjena razine između prijelazne i kontakt poze, te je uvijek najbolji način za početak hoda uzeti kontakt pozu. U osnovi, bazni walk cycle traje 2 koraka u jednoj sekundi, a 1 sekunda vrijedi 1000 milisekunda u Spriteru. Svaka polusekunda sastojat će se od jednog potpunog koraka. Potrebno je ukupno 8 poza, a prva i zadnja moraju biti identične. Svaka poza udaljena je od druge 125 frame-ova, a frame rate će biti postavljen na 30 sličica po sekundi. [4]

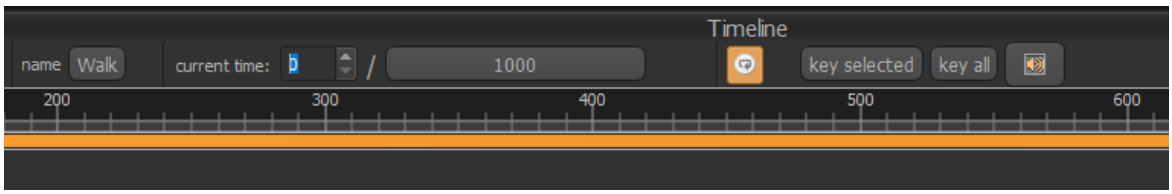
3.4.1. Kontakt pozicija



Slika 3.17 Kontakt pozicija tjala [4]

Iskusni animatori uvijek se oslanjaju na dinamičke poze te *stick figure*. Na slici je primjer dinamičke poze, koja je ujedno i prva poza u animaciji hoda. Može se vidjeti kako je kralješnica savijena što daje dinamiku cijeloj slici, kao i animaciji. Česta greška kod početnika upravo je nedovoljno naginjanje tijela unaprijed što daje dojam statike i pokreti izgledaju neprirodno.

Počinje se na nultom frame-u, s prednjom nogom i stražnjom rukom naprijed, suprotnim udovima otraga, ispruženim prednjim stopalom te savijenim stražnjim. Spriter po defaultu koristi opciju „auto key frame“, što se može promijeniti a to ovisi o preferenciji animatora.



Slika 3.18 Uključena opcija „auto key frame“ u vremenskoj traci (označena narančastom bojom) te prikaz kućice za upisivanje željenog framea, kreirala autorica rada

Kada se prelazi s jednog na drugi keyframe s uključenom opcijom „auto key frame“ bitno je ručno u timelineu upisati broj keyframea na koji se želimo premjestiti, umjesto da se pokazivač na linijskoj crti odvlači na željeno mjesto. To se radi kako bi se izbjegle dodatne i neželjene poze i njihovi key frameovi.Nakon prve poze, na keyframeu broj 500, namješta se suprotna poza. [4]

3.4.2. Pass pozicija

Slika 3.19 Pass poza tijela [4]

Prva prijelazna ili pass pozicija nalazit će se između prve i druge kontakt poze, na frameu broj 250, s ravnom prednjom nogom, rukama gotovo uz tijelo te stražnjom nogom jače savijenom u koljenu. Suprotna poza nalazi ser na frameu 750. [3]

3.4.3. Down pozicija

Slika 3.20 Donja ili „down“ pozicija tijela [3]

Sljedeća je na redu down pozicija.To je poza gdje težina tijela pogađa tlo i vrlo je bitna jer daje vitalnost animaciji. Na stick figuri može se vidjeti jasno. Linija gdje dolazi glava vrlo je nisko s obzirom na prethodnu i sljedeću poziciju. To je pozicija gdje su koljena najviše savijena, a ruke najviše razmaknute od tijela, a čitava poza u duljini je šira od svih ostalih. Ova pozicija dolazi na frame 125 i 625. [4]

3.4.4. High pozicija

Slika 3.21 Visoka ili „high“ pozicija tijela [4]

Zadnja ključna pozicija je visoka pozicija, kod koje glava dolazi do najviše točke, i zato je potrebno na samom početku čitavu figuru podići po y osi. Pozicija je najsličnija onoj prijelaznoj, osim što figura mora biti na prstima, a ruke moraju biti malo razmaknutije. Pozicije dolaze na frameove 375 i 875.

Kada je animacija gotova potrebno je reproducirati je mnogo puta i pažljivo provjeriti ima li u njoj kakvih nedostataka.

Iako za walk animaciju postoji dobar recept, oblik tijela, polufrontalni pristup i različita dužina prednjih i stražnjih udova zahtijevaju blaga odstupanja od pravila i svaku je poziciju bilo potrebno dodatno prilagoditi tijelu kako bi one izgledale prirodno. Na slici ispod vidljiva je svaka ključna poza karaktera. [4]

Slika 3.22 Redoslijed svih 8 ključnih poza u animaciji „walk cycle“, kreirala autorica rada

3.5. Idle animacija

Idle animacija, vrlo tipična u svakoj igri, bila 2D ili 3D, ima svrhu držati igrača kao i lika angažiranog u pokretu, a isto tako omogućava i fluidnost animacije kod prijelaza iz idle u walk ili neku drugu animaciju. Kod pristupa idle animaciji, prva važna stvar je definiranje poze, koja bi idealno trebala biti dinamična, i djelovati kao da je igrač svakog trenutka spreman skočiti, trčati, napraviti nešto, pa generalno nije najbolje ako to bude statična stajaća poza. Umjesto toga, odlučeno je kako će poza biti blago raširenih nogu, savijenih u koljenima, također raširenih ruku, u sličnoj poziciji kao i ruke na prvoj ključnoj pozi u walk animaciji, odnosno kontakt pozi. Lijeva ruka daje dojam kao da je igrač svakog trena spreman u napad, dok desna drži balans lijeve ruke i tjela u cjelini.

Na primjeru idle animacije također će biti korisna inverzna kinematika, koja nije bila primijenjena na animaciji hoda, no ovdje će biti neophodna jer se idle animacije neće se micati s

iste težišne točke, tako da stopala moraju biti zaključana za tlo, a to se najlakše i najbrže postiže inverznom kinematikom. Kada se to ne bi primjenjivalo, stopala bi se trebala za svaki key frame ponovno i ručno odvlačiti na isto mjesto kao i na prethodnom frameu, što je gotovo nemoguće te oduzima previše vremena. U Spriteru je IK manipulacija vrlo jednostavna, te ukoliko, primjerice, želimo utjecati na sve child kosti ruke, te pomak ruke napraviti brzo tako da on izgleda prirodno, potrebno je označiti kost nadlaktice, pritisnuti tipku Z na tipkovnici, i sve child kosti će se automatski označiti i pomak će se ponašati prirodno. Ova opcija moguća je u programu Spriter Essentials, dok Spriter Pro uz to nudi i neke naprednije postavke, kao što je „Lock" opcija, pa ako jednostavno kliknemo desnim klikom i držimo miš na child kosti, pojavit će se mali prozorčić koji omogućava da označimo 3 različite opcije. Potrebno ja na stopalu označiti prvu ikonu, a to je „anchor" kako bi zaključali stopalo za tlo.

Slika 3.23 Dodavanje sidrišne točke na kost stopala, kreirala autorica rada

U ovu svrhu „anchor" je također postavljen i za obje potkoljenice, i sada se potkoljenice pomiču ali vrlo blago. I sada ako pritisnemo tipku shift u isto vrijeme držeći glavnu zdjeličnu kost, vidjet ćemo kako će se stopala držati za tlo, iako će se malo micati jer je algoritam iza ove opcije ipak osmišljen da djeluje organski i prirodno, što ne uključuje apsolutno zaključavanje i držanje spritea na istoj točki . Inače je druga ikona u prozorčiću (inherit) označena po defaultu, te je ključna u IK manipulaciji, jer ona nema smisla ako kosti koje su niže u hijerarhiji ne budu zahvaćene za parent kosti kod bilo kakvog pomaka te ima smisla samo u nekim situacijama. Međutim, ako se i isključi inherit opcija dok je „anchor" uključen, pomakom se cijeli kostur raspada, tako da će ručno podešavanje biti svakako potrebno, iako „anchor" olakšava cijelu radnju.

Kod Idle animacije ključan je pokret zdjelice naprijed – natrag te blago gore-dolje. Važno je ne pretjerati te imati na umu kako idle animacija mora biti vrlo suptilna. Idle će trajati jednu sekundu.

Idle animacija sastoji se od 3 ključna framea, od kojih su prvi i posljednji jednaki, a animacije je blago spuštanje zdjelice i cijele figure prema dolje, savijanje u koljenima te širenje ruku. [4]

Slika 3.24 Redoslijed triju ključnih poza za „idle“ animaciju, kreirala autorica rada

3.6. Jump animacija

Za početak jump animacije, koristit će se prvi frame iz idle animacije. Jump animacija koristi slične principe kao i „bouncing ball“ animacija. Jump animacija će također biti na mjestu, jedino što će se noge odvajati od tla po y osi. Početna poza iz idle animacije bit će dobra jer je za skok potrebna tenzija an nogama. Sljedeći key frame počinjemo sa zdjelicom i zaključavanjem stopala kako bi spustili cijelu figuru u koljenima dolje.

U teoriji, skok animacija trebala bi se sastojati od tri ključne pozicije. Početna ili kontakt pozicija, pozicija spuštanja i gornja pozicija, odnosno pozicija kada je tijelo u zraku. Razmak između prve dvije pozicije mora biti nešto sporiji od samog skoka. Međutim, kasnije kada će u Unityju biti potrebno implementirati istu animaciju, ovo ne bi funkcioniralo iz više razloga. Prvi razlog je skočna sila i gravitacija, a to su funkcije koje je potrebno napraviti u samom Unityju, jer će najviša točka skoka morati biti prilagođena sceni te je svakako poželjno da se to može najprije isprobati i tek onda odlučiti o visini i brzini skoka, a to je lakše kada se najprije testira u igri. Drugi je razlog „jump attack“ animacija, što je u osnovno zamišljeno kao napad u trenutku kada je lik već u zraku, a o tom trenutku napada opet mora odlučivati igrač, tako da se napad izvrši u trenutku kada se pritisne željena komanda. Zbog toga je jump animaciju potrebno napraviti „na mjestu“, odnosno potrebno je animirati skok, ali bez pomaka po y osi. Isto tako, zbog drugog spomenutog razloga, animacija skoka mora se razdijeliti na dvije faze – fazu polijetanja i slijetanja (u daljnjem tekstu „take off“ i „landing“)

Slika 3.25 Redoslijed ključnih poza te brojevi frameova u vremenskoj traci za animaciju polijetanja, kreirala autorica rada

Take off animacija sastoji se od ukupno 5 ključnih frameova, od čega su 4. I 5. Identični. Na slici se može vidjeti redoslijed svakog ključnog framea kao i njihovo mjesto na vremenskoj crti. Prva 3 ključna framea odvijaju se brzo, a zadnja poza, za koju je predviđeno da se izvršava tako dugo dok igrač ne prijeđe u fazu padanja, traje ukupno 545 frameova, što je malo više od pola minute.

Landing animacija trebala bi početi sa istom pozom s kojom take off završava, stoga landing animacija izgleda ovako:

Slika 3.26 Redoslijed ključnih poza te brojevi frameova u vremenskoj traci za animaciju slijetanja, kreirala autorica rada

Landing animacija se sastoji od 6 ključnih frameova, od kojih se prvih 5 izvršava vrlo brzo, dok je šesti opet identičan petome, kao i u primjeru take off animacije. Kasnije je u Unityju cijela animacija malo usporena jer se testiranjem ustanovilo kako se izvršava pre brzo te je između 2 i 3. keyframea dolazilo do čudnog i neprirodnog savijanja u tranziciji animacije, no problem je bio rješiv u game enginu pa samu animaciju u Spriteru nije bilo potrebno mijenjati. Može se vidjeti kako landing počinje s istom pozom s kojom take off završava, a ta poza tek nakon 2. Framea

prelazi u poziciju gdje karakter ravna noge a nakon toga u počučnju pada na tlo, a iz počučnja se diže u prirodnu stajaću poziciju, koja je identična početnoj poziciji idle animacije. [1, 4]

3.7. Attack animacija

Slika 3.27 Redoslijed ključnih poza te brojevi frameova u vremenskoj traci za animaciju napada mačem, kreirala autorica rada

Attack animacija također traje jednu minutu te se sastoji od ukupno 15 ključnih frameova. Za ovu animaciju nije korištena nikakva špranca ni pravilo, već je napravljena intuitivno, višestrukim isprobavanjem i testiranjem. Također je korištena inherit IK metoda kako bi animiranje bilo lakše

a krajnji rezultat izgledao bolje. Prvi frame na nultoj poziciji izgleda kao gard gdje je lik u blagom počučnju, koji se do trećeg framea zadržava i sasvim suptilno još više spušta dok lik u isto vrijeme diže mač, a u trenutku kada mu je mač u ravnini s licem, kralješnica se savija prema natrag. Time se postiže dojam stabilnosti tijela. Stražnja ruka ostaje gotovo na istome mjestu kroz cijelu animaciju. Na 6. Frameu mač dostiže svoj krajnji domet prema natrag te se ruka najviše savija u laktu, a nakon njega mač se zajedno s rukom ponovno ravna, lik se saginje prema naprijed, stražnja ruka u tom se trenutku pokreće sasvim malo prema tijelu, a lik je opet u jačem čučnju. Od 11. do 12. framea on se ponovno ravna u koljenima te podiže mač u skoro početni položaj, a od 13. do 14. framea dolazi do promjene samo u prednjoj ruci u kojoj drži mač, jer se ona vrlo brzo podiže te istu poziciju zadržava do kraja animacije. Zbog broja frameova i pravilnog proračuna vremena te vrlo suptilnih promjena, čitava animacija izgleda kinematički pravilno i fluidno. [4]

3.8. Dead animacija

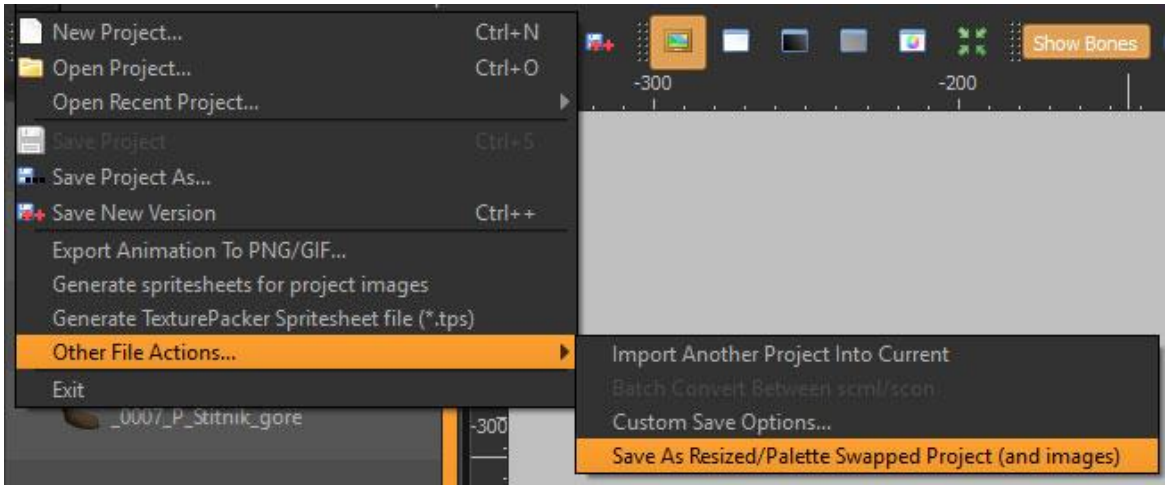
Dead animacija sastoji se od 10 ključnih frameova te nije zamišljena kako bi se ponavljala u loop-u. Zadnja dva framea su identična, a u ovoj animaciji prvi puta je primijenjena „swapping“ metoda za promjenu spriteova.

Slika 3.28 Redoslijed ključnih poza te brojevi frameova u vremenskoj traci za animaciju umiranja, kreirala autorica rada

Dead animacija je pad lika, koji je postignut spuštanjem glavne, zdjelične kosti niže po y osi. Animacija počinje neutralnom pozicijom istom kao i u idle animaciji, nakon koje se kvrči u koljenu, te u rukama, čime se postiže dojam kako gubi energiju. Pad se proteže kroz ukupno 6 ključnih frameova, dok je na 7. Napravljena izmjena cijelog spritea glave. Nova sličica dovučena je na platno, a efekt izmjene postigao se jednostavnim odabirom na novu sprite a zatim desnim klikom na postojeći, odnosno na sličicu glave s otvorenim očima. Izmjena se automatski izvršila, novi sprite se pojavio u hijerarhiji te je automatski povezan s kosti glave i kao takav podložan danjoj manipulaciji. Nakon ove izmjene, glava se zajedno s vratom spustila na tlo. U ovoj animaciji također se mijenjao i redoslijed po hijerarhiji Z – osi. Na slici (BROJ) može se vidjeti kako se mač počeo odvajati od ruke odmah nakon 4. ključnog framea, dok je na 5. udario o tlo te je udaljen po Z osi, tako da je sa samog vrha hijerarhije skočio na dno. Redoslijed spriteova po Z osi može se mijenjati pritiskom na Ctrl i strelice gore dolje, dok Ctrl tipka u kombinaciji s lijevom ili desnom strelicom smješta određeni sprite na samo dno ili vrh hijerarhije. Od 5. do 6. ključnog framea mač je po tlu još malo otklizio te zauzeo položaj ispod glave te iza stražnje ruke karaktera. U ovoj poziciji i završava animacija. [4]

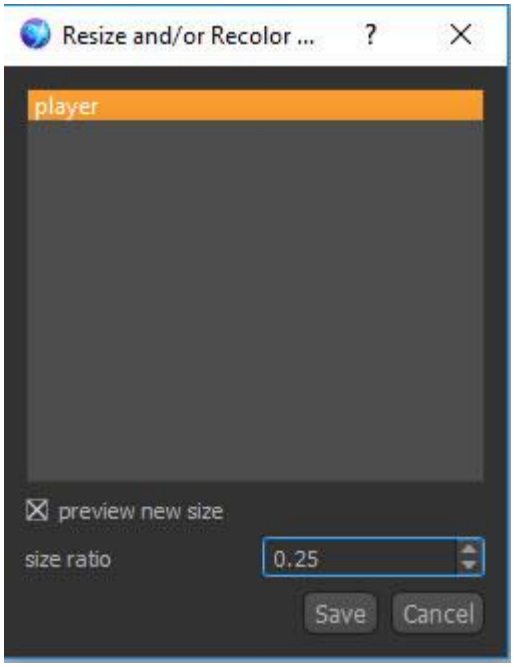
3.9. Dovršavanje i spremanje projekta za game engine

Kada su animacije gotove, projekt je potrebno spremiti u odgovarajućoj veličini. Budući da su svi spriteovi nacrtani u 4K rezoluciji (kako detalji bolje izgledali kada se smanje), cijeli je projekt potrebno spremiti u smanjenoj veličini. To se postiže tako da se u projektu označi File > Other File Actions... > Save As Resized/Palette Swapped Project (and images). [2]



Slika 3.29 Prikaz odabira komanda za spremanja .scml datoteke u Spriteru u smanjenoj verziji, kreirala autorica rada

Nakon toga se otvara prozorčić u kojem možemo označiti postotak smanjenja projekta u decimalnoj vrijednosti. Kako je normalna vrijednost veličine projekta uvijek 1, za četvrtinu veličine projekta upisana je vrijednost 0.25, što je nakon testiranja u Unityju dobro odgovaralo veličini scene.



Slika 3.30 Prozor za spremanje smanjene verzije dokumenta, kreirala autorica rada

Spremanje projekta je zadnja faza rada u Spriteru. Projekt je dovoljno spremiti u zadanom formatu koji nudi Spriter - .scml, budući da postoji plugin koji omogućava Unityju da prepozna te animacije. U suprotnome bi bilo potrebno sve ključne frameove zasebno spremiti te od njih načiniti animaciju u .gif formatu. Dodatna prednost plugina je mogućnost da Unity engine prepozna krivulje brzine ukoliko one postoje a isto tako i kosti i kinematiku iza animacija. [2,3]

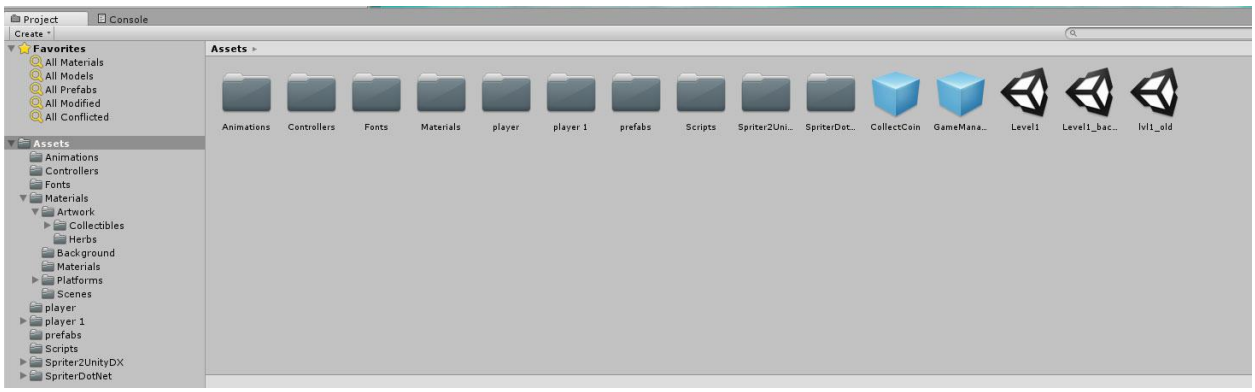
4. Izrada igre u Unity game engine-u

4.1. Što je Unity Editor?

Unity Editor je cross – platform game engine koji je razvila kompanija „Unity Technologies“, prvi puta objavljen u lipnju 2005. godine na Apple Inc.-ovoj konferenciji "Worldwide Developers Conference". Do danas, Unity je razvijen tako da podržava ukupno 27 platforma, a koristi se za 2D i 3D igre ali i kao simulator za stolna računala, kućne konzole, pametne televizore i mobilne uređaje. Posljednja verzija Unity 2018.2.2, izdana je 10. kolovoza 2018. godine. Osim mogućnosti stvaranja igara u 2D i 3D svijetu, Unity nudi primarni API za skriptiranje u C# programskom jeziku, grafičko sučelje sa drag&drop metodom i mnogobrojne plug-inove. Do verzije 5, podržavao je i Boo jezik, a od verzije 2017.1 više ne podržava niti JavaScript jezik. Samo neki od platforma koje engine podržava su: Direct3D za Windows i Cbox One, OpenGL za Linux, MacOS i Windows, OpenGL za Android i iOS, WebGL za web. [10]

U 2D igrama, Unity omogućava uvoz spriteova te napredni 2D "world renderer". Za 3D igre, Unity omogućava specifikaciju za komprimiranje tekstura, Mipmap i postavke razlučivosti posebno za svaku platformu koju podržava. Također podržava i "bump" metodu mapiranja, reflektivno mapiranje, "parallax" mapiranje, "screen space" ambijentalnu okluziju (SSAO), dinamičko sjenčanje koristeći "shadow" mape, "render-to-texture", "full screen" post proces efekte. Također nudi i određene servise za developere; Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build i mnoge druge. Kod 3D kreacija, Unity podržava metode za stvaranje prilagođenih Vertexa, fragmenata ili piksela, teselaciju tekstura, kalkulacije za shadere kao i vlastite CG shadere koje su modificirane verzije Microsoftovih "high - end" metode za shadere razvijenih od strane kompanije NVIDIA . [10]

Kao takav, Unity je vrlo jednostavan i intuitivan za korištenje. Od prvog otvaranja projekta, korisniku se nudi odabir 2D ili 3D opremljenog editora za novi projekt. Novi projekt sprema se na željeno datotečno mjesto te na tome mjestu stvaraju zadane mape kao što su: Assets, Library, obj datoteka s pripadajućim cache, .dll i tekstualnim dokumentima te Project Setting mapu u kojoj se nalaze datoteke .asset ekstenzija a sadržavaju informacije o svim postavkama projekta. Mapa Assets ekvivalentna je „Project“ prozoru u grafičkom sučelju programa. Svaka izmjena u tome prozoru, odnosno svako dodavanje novih animacija, skripti, grafika, fontova i pluginova bit će vidljivo na datotečnom mjestu ili obratno. [3]

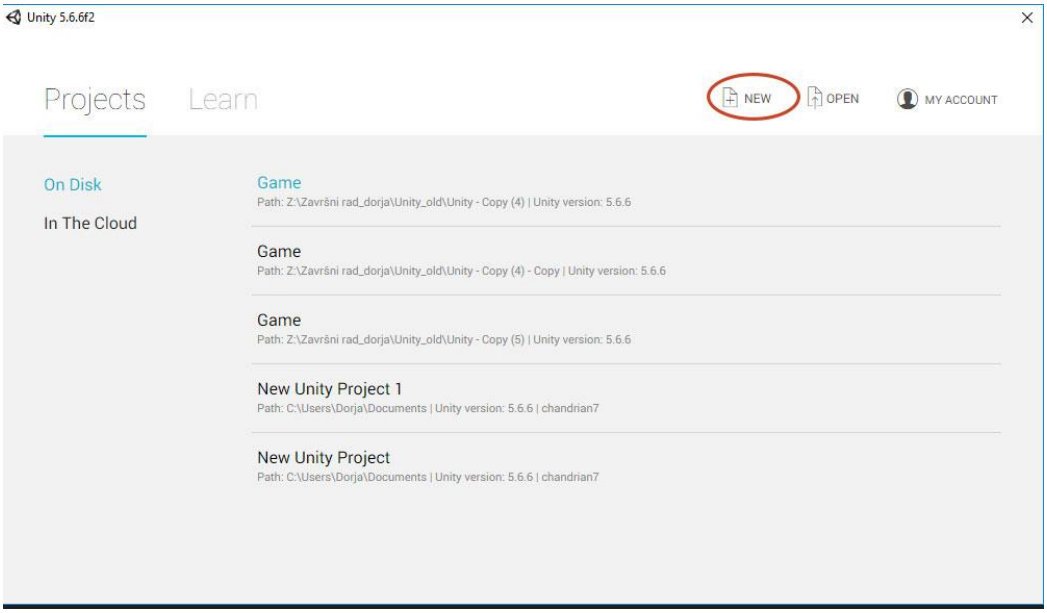


Slika 4.1 "Project" prozor s organizacijom svih uvezenih assets-a, kreirala autorica rada

Na slici je vidljiv prikaz organizacije svega što projekt sadržava u „Project" prozoru u Assets mapi."Asset" je reprezentacija bilo koje stavke koja se može upotrebljavati u igri ili projektu, a može doći iz datoteke spremljene negdje izvana (kao što su primjerice animacije napravljene u Spriteru ili neki 3D model), ili može biti kreirana u samom Editoru (tekstura, skripta, scena, materijal, shader, sprite, animacija ili nešto drugo). [3]

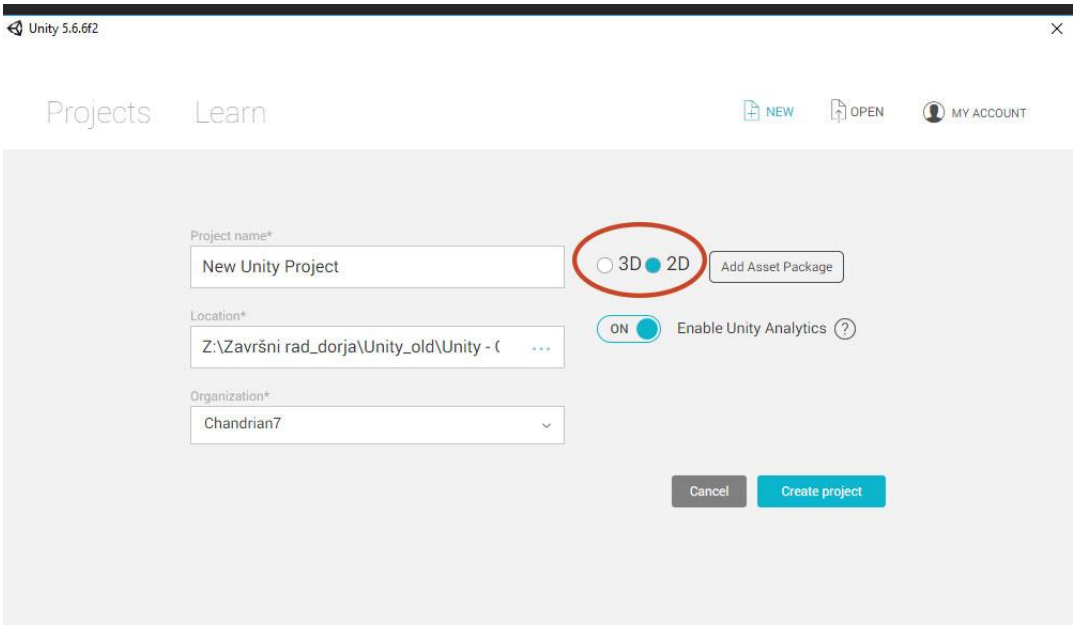
4.2. Izrada novog projekta, kreiranje i organizacije scene u igri

Kada se otvori Unity Editor, pojavljuje se početni prozorčić koji pokazuje već postojeće, zadnje otvarane projekte te nudi opciju „New" za stvaranje novog projekta te „Open" za otvaranje postojećeg.



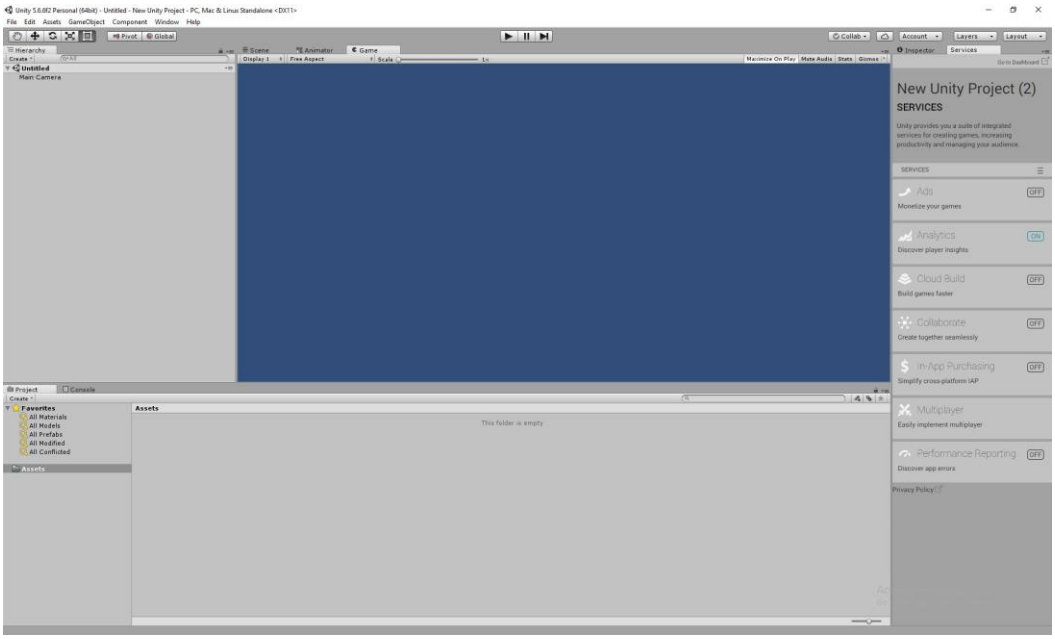
Slika 4.2 Početni prozor Unity Editor te prikaz ikone za kreiranje novog projekta, kreirala autorica rada

Nakon što se odabere opcija za stvaranje novog projekta, u prozoru se odabire ime projekta, lokacija na disku, te je uvijek potrebno odabrati željeni način rada – 2D ili 3D. Projekt se stvara pritiskom na gumbić „Create project“ i otvara se prazan projekt.



Slika 4.3 Odabir imena, lokacije i tipa projekta, kreirala autorica rada

Glavne sekcije grafičkog sučelja Editora su alatna traka s ikonama za brzi odabir, scena, hijerarhijski prozor, „Project“ prozor opisan u poglavlju prije te „Inspector“ prozor, čija će funkcija biti objašnjena u kroz opis izrade projekta. [3]



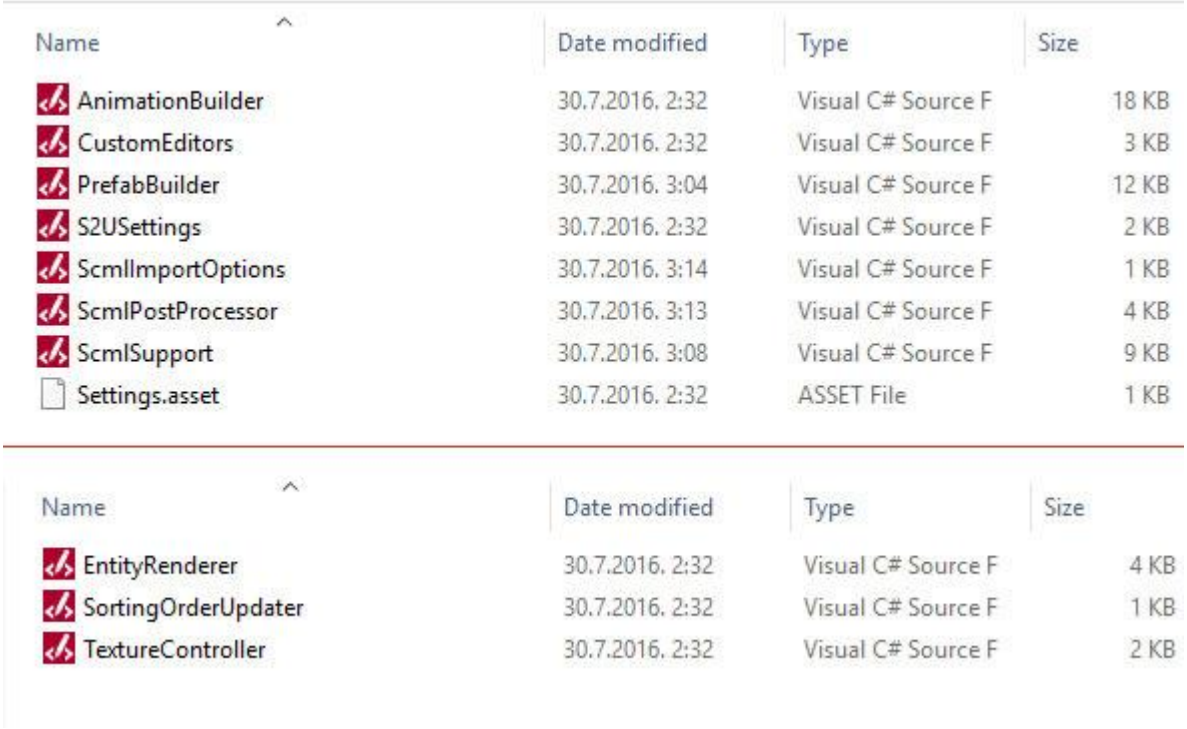
Slika 4.4 Novi, prazan projekt u Unity-ju, kreirala autorica rada

Svi pripremljeni materijali mogu se zatim „drag&drop“ metodom odvući u projektni prozor u zadanu mapu „Assets“, te se potom organiziraju u mape. Načinjene su nove mape za animacije, skripte, platforme, pozadinu, fontovima, pluginovima te za sve ostalo što će se koristiti.

Scena se može graditi odmah ili u toku izrade projekta, no budući da je najbitnije najprije postaviti sve funkcije za igraća, potrebna nam je barem jedna platforma, i svakako „SpriterToUnityDX“ skripta koja služi kao plugin za animacije napravljene u Spriteru.

Centar igre i najbitniji dio je sam karakter, odnosno igrač koji se kontrolira, te ga je zato potrebno izraditi prije neprijatelja i svih ostalih objekata s kojime će biti interaktivan. Budući da je za animacije korišten Spriter, u Asset folder najprije je importan plugin sa skriptama SpriterToUnityDX, a taj plugin omogućava direktnu implementaciju animacija zajedno s njenim bihevioralnim postavkama kao što su ease in i out funkcije, collideri dodani na mač u „Attack“ animaciji, kao i označene postavke za loop u Spriteru i slično.

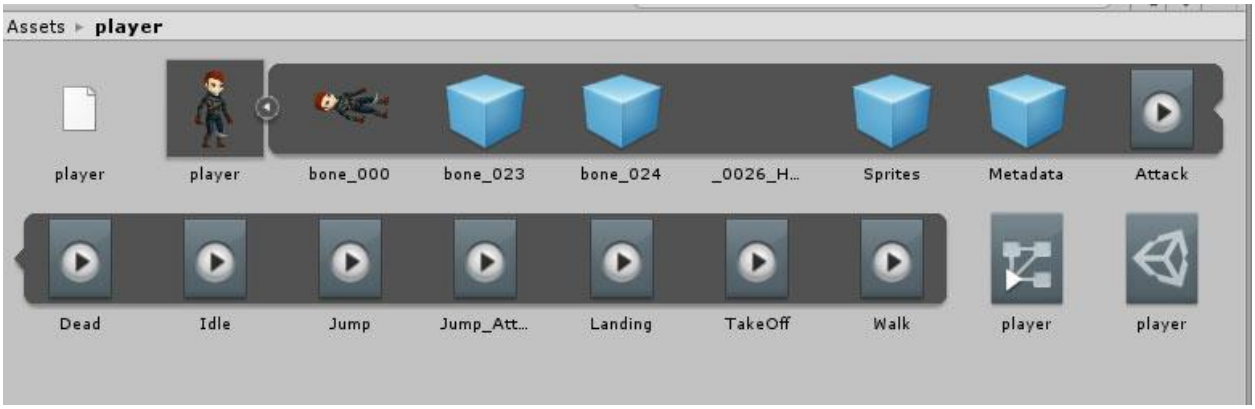
Plugin je dostupan an web adresi https://brashmonkey.com/forum/index.php?/topic/3993-spriter-for-unity-50/ a kompatibilan je samo s inačicom programa Unity 5 ili starijima, pa je stoga cijeli projekt i rađen u starijoj verziji. Nakon što se plugin instalira, cijela mapa odvlači se u projektni prozor u Unity, te sve njene skripte istog trena postaju aktivne. [2]



Slika 4.5 Prikaz svih skripti koje se nalaze u plug-inu SpriterToUnityDX, kreirala autorica rada

Ako otvorimo mapu plugina, možemo vidjeti da se on sastoji od skupa skripti .cs ekstenzija, štop znači da su pisane programskim jezikom C#, a svaka od njih omogućava kompatibilnost i konverziju svih Spriterovih svojstava u formate prepoznatljive Unity-ju. [2]

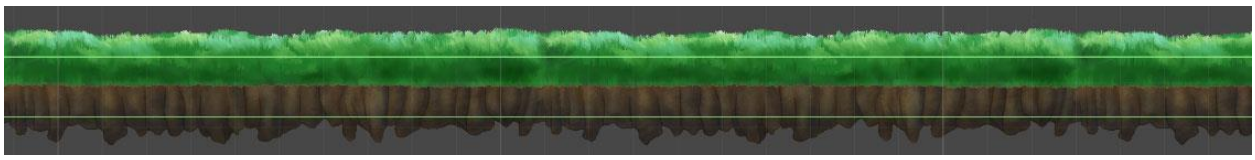
Nakon plug ina, na isti način u projektni prozor potrebno je dodati i samu .scml datoteku u kojoj se nalaze animacije.



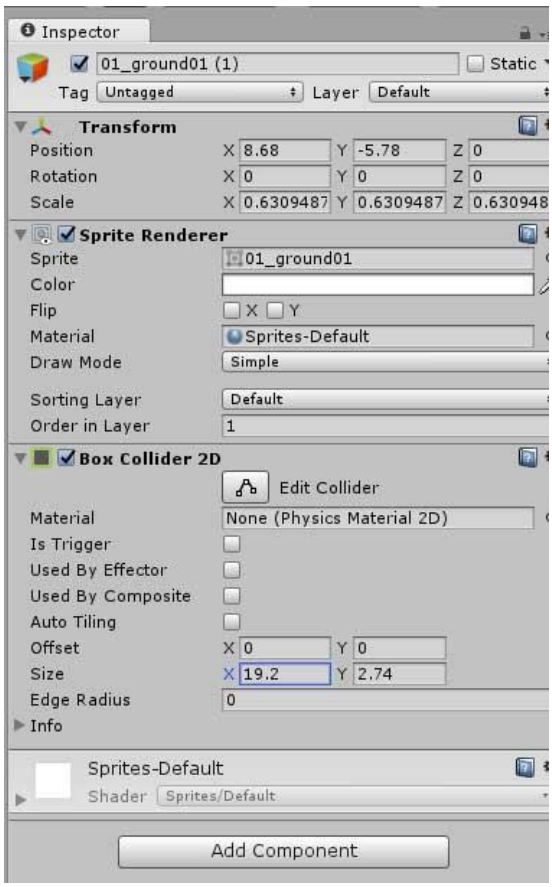
Slika 4.6 Prepoznate animacije .scml datoteke nakon uvoza u Unity, kreirala autorica rada

Ako je thumbnail animiranog lika, te sve njegove animacije i odgovarajući „controller“ objekt, .scml datoteka je prepoznata te se animacije mogu početi implementirati. Prije nego se sam player sa svojom zadanom (idle) animacijom odvuče u scenu, potrebno je najprije u scenu staviti barem jednu platformu, kako bi se hod mogao na njoj testirati.

Za platformu je potrebno dodati granice i fizičko svojstvo „BoxCollider 2D“, jer je bez njega platforma samo obična slika bez ikakve funkcije. Inače je platforma nacrtana u photoshopu na način da ima svoj rubni lijevi i desni dio, a središnji dio nacrtan je kao „seamless“ uzorak te se može ponavljati bezbroejno mnogo puta između dva rubna dijela platforme. Fizičko svojstvo platformi dodaje se tako da se označi središnji dio platforme i u Inspector prozoru odabere se Add Component > Physics 2D > Box Collider. U Inspectoru se odmah pojavljuje Box Collider Settings. Kada je središnja platforma odabrana, opcijom size u Inspectoru vrijednost X osi se poveća tako da box Collider obuhvaća sve tri platforme, dok je na Y osu pod Offset smanjena vrijednost kako bi se područje kolizije malo spustilo, jer bi u suprotnom lik hodao po samoj površini trave, što bi izgledalo nerealno. Dakle sve što dodiruje vrh Box Colider elementa, a da također ima koliziju, stajat će na vrhu tog područja kolizije. [3]



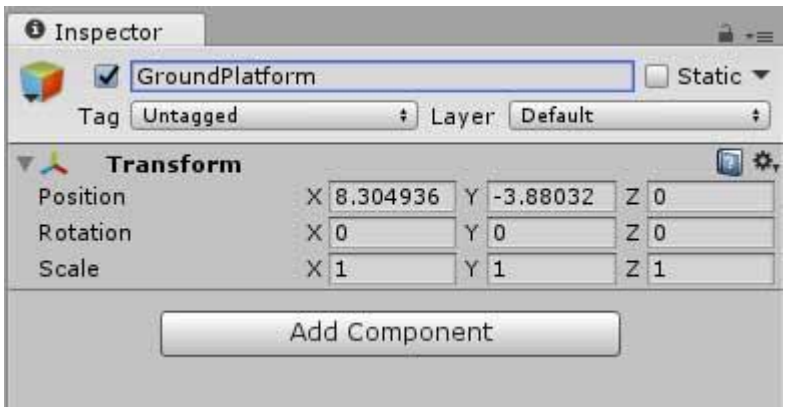
Slika 4.7 „Box Collider 2D“ na slici platforme, kreirala autorica rada



Slika 4.8 „Box Collider 2D“ komponenta sa svojim opcijama u Inspector prozoru, kreirala autorica rada

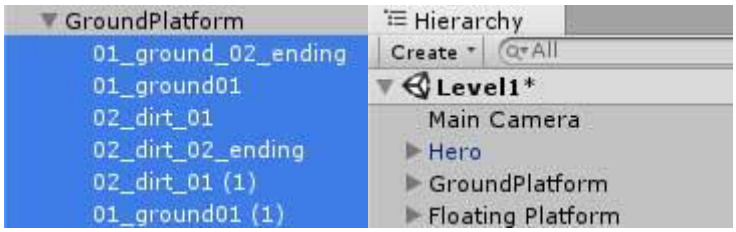
Box Collider je komponenta i može biti dodana baš svakom objektu u sceni, a može postojati i kao nevidljiv objekt, nevezan za nešto drugo. Ono što box Collider radi jest da sprječava da nešto prolazi kroz njega, što također ima dodan „Box Collider“. Dakle on služi kao fizička granica. Box Collider može se također koristiti i kao trigger, što znači da, kada god nešto prolazi kroz ili dodiruje predmet sa Box Colliderom, to će triggerati neku vrstu događaja. Primjerice ako programiramo metak koji dolazi iz pištolja, kada dodirne koliziju neprijatelja, želimo stvoriti štetu neprijatelju a za metak napraviti da nestane. Kada je na platformu dodana kolizija, ona se može ponovno koristiti sa svojom kolizijom nebrojeno mnogo puta kao prefab objekt. [3]

Sljedeći je korak označavanje svih platformi i kreiranje prefaba-od njih, tako ga istu platformu s dodanom kolizijom možemo koristiti opet ako zatreba. Potrebno je označiti sve „tileove“ koji čine tu platformu, što je najlakše napraviti Shift klikom u prozoru hijerarhije, kao i odgovarajući Collision Box, kliknuti na Game Object > Create Empty , te se stvara novi objekt po zadanome nazvan GameObject, a u ovom slučaju taj će se naziv promijeniti u GroundPlatform.



Slika 4.9 Dodjeljivanje naziva objektu platforme, kreirala autorica rada

Radi bolje organizacije, sve tile objekte koji sačinjavaju platformu najbolje je označiti i to Shift klikom u prozoru „Hierarchy“, te dovući u novi objekt „GroundPlatform“, tako da oni postaju djeca toga objekta. Isto ja napravljeno sa platformama u zraku, te su sve smještene u prazan objekt „FloatingPlatforms“.

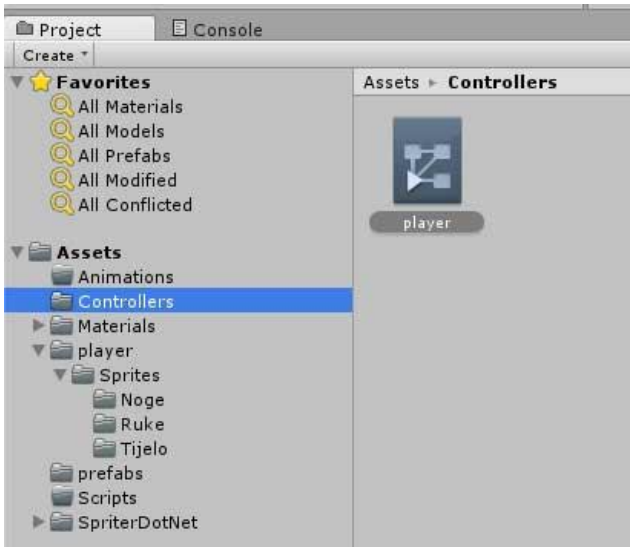


Slika 4.10 Prikaz hijerarhije i organizacije platformi u Unityju, kreirala autorica rada

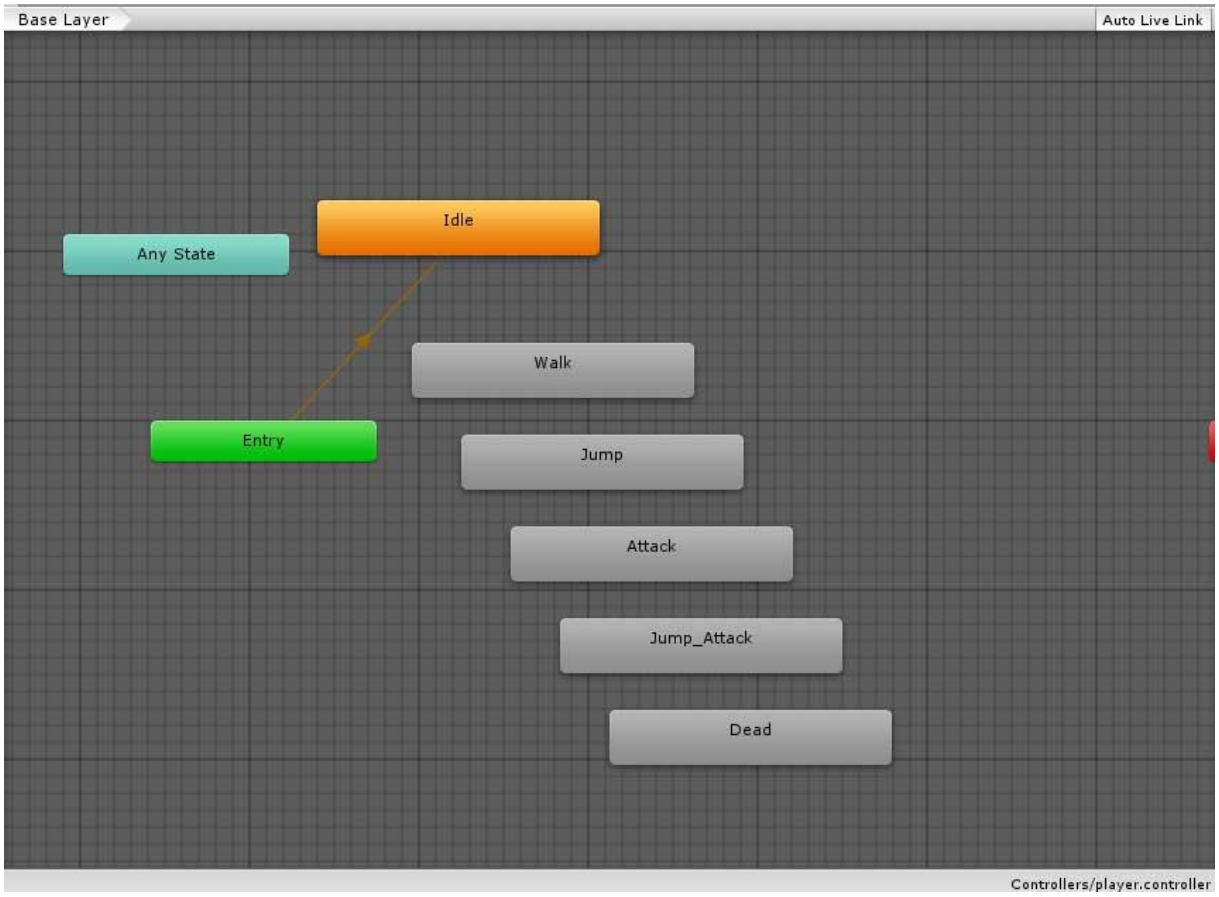
Nakon toga, platforme s kolizijom odvučene su u „prebabs“ podmapu mape „Assets“. [3]

4.3. Implementacija animacije i kreiranje tranzicija

Ako pogledamo sve komponente koje spadaju u „player“ nakon importa .scml datoteke, možemo vidjeti da u paketu s player entitetom dolazi i vrsta objekta koja se zove „controller“. Taj je objekt premješten u posebnu mapu Controllers radi bolje organizacije, te je mapa predviđena za sve controllere ostalih likova, ako će ih biti. Duplim klikom na controller otvara se prozorčić Animator s popisom svih animacija. Zadana animacija je uvijek označena narančastom bojom, a plugin za Spriter SpriterToUnityDX omogućio je automatsku implementaciju svih animacija s prvom animacijom na popisu kao zadanom.



Slika 4.11 Controller komponenta animiranog lika, kreirala autorica rada



Slika 4.12 Izgled animacija u „Animator“ prozoru prije izrade tranzicija, kreirala autorica rada

Animator prozor služi za stvaranje tranzicija iz jedne animacije u drugu, no prije toga potrebno je za samog igraća napraviti rigid body. Rigid body je vrsta objekta koji komunicira s fizikom u Unity-ju. Za dodavanje rigid body komponente potrebno je označiti playera u hijerarhiji, u Inspectoru odabrati Add Component te dodati komponentu „Rigidbody 2D“. S dodanom

komponentom, ako se igra pokrene u ovoj fazi, karakter će početi padati prema dolje. To je zato jer je zadana vrijednost gravitacije 1, a ako je postavimo na 0, karakter će biti na mjestu. Međutim, gravitacija mora biti dodana, ali u ovoj fazi testiranja karaktera još nije neophodna.

Daljnji korak je dodavanje skripte za rigidno tjelo u datoteci Script. To se vrši desnim klikom na prazan prostor u Asset prozoru te odabirom na Create>Script>C# Script.

Duplim klikom na skriptu, ona se otvara u zadanom tekstualnom uređivaču. U skripti najprije treba napraviti referencu za komponentu rigid body. Najprije je u public klasi player deklarirana privatna varijabla „`private rigidbody2D myRigidbody;`“, a zatim je pozvana u void funkciji Start kao „`private void HandleMovement ()` “.

Funkcija `private void HandleMovement ()` dodaje se ispod Update funkcije te je unutar vitičastih zagrada potrebno odrediti velocity instancu za taj rigid body. Velocity je vektorski alat unutar Unityja te se na ovaj način poziva te mu treba pridružiti vrijednost Vector2. Vector2 označava x os. [5]

Nakon što je funkcija napisana, potrebno ju je pozvati u funkciji Update. Funkcija update osvježava se konstantno kroz igru. Na primjer ako napišemo:

```
myRigidbody.velocity = Vector2.left; //x = -1, Y = 0;
```

govorimo programu da na početku igre želimo da se player kreće u lijevo, odnosno na -1 po x osi, i 0 po y osi. Međutim, potrebno je dodati funkcionalnost kontrole, jer se ova radnja izvršava bez pritiska bilo kakve kontrole.Zato se u funkciji Update mora dodati:

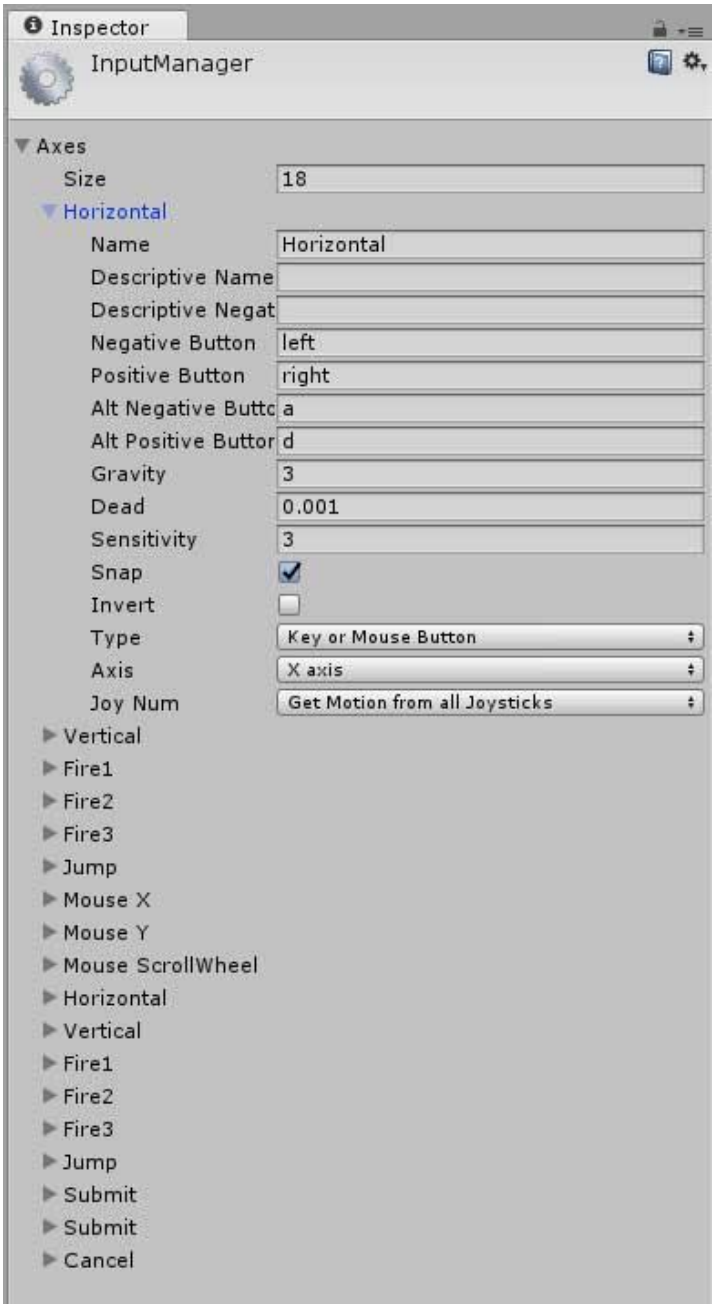
```
float horizontal = Input.GetAxis("Horizontal");
```

Float je vrsta varijable, a u Unityju postoji klasa Input, a unutar nje postoji klasa koja se zove Input.GetAxis, te ona traži neki input po osi, a ovom linijom toj je funkciji pridana vrijednost Horizontal, koja je već ugrađena u Unity, a služi za pomicanje objekata lijevo – desno. Nadalje, „Horizontal“ se treba pozvati u funkciji HandleMovement kako bi obavljao svoju ulogu, i zato u funkciji HandleMovement unutar zagrada dodajemo parametre, na što debugger javlja kako metoda HandleMovement nema argumente pa unutar Update funkcije pod HandleMovement u zagradu upisujemo opet „horizontal“. Kada je horizontal definiran, u prethodno napisanoj liniji u funkciji private Void pod myRigidbody.velocity upisujemo new Vector2 te zagradu. U zagradi se traže osi x i y pa zato prvo pišemo horizontal, a budući da y os ne želimo mijenjati, koristimo zadanu vrijednost y osi.

```
        myRigidbody.velocity = new Vector2(horizontal, myRigidbody.velocity.y)
;
```

Ovime je dodana funkcionalnost da playera možemo pomicati lijevo desno s tipkama a i d ili strelice lijevo desno. Tipke kojima ovaj pokret želimo kontrolirati mogu se lako promijeniti u

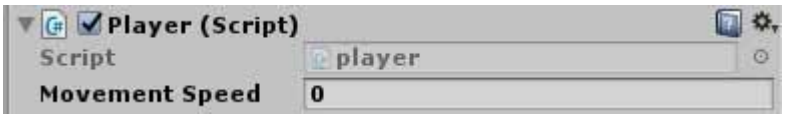
Unityju pod Edit > Project Settings > Input te u inspektoru promijenimo zadane vrijednosti u odgovarajućim poljima, bez potrebe da se isto definira u kodu. [3]



Slika 4.13 Input Settings popis opcija za zadane ulazne kontrole preko tipkovnice u Unityju, kreirala autorica rada

Sljedeći je korak dodavanje brzine igraču, pa u public klasu Player dodajemo novu varijablu private float tipa s nazivom movementSpeed. Nadalje, sve se funkcionalnosti mogu podesiti u inspectoru, pa iznad te varijable upisujemo [SerializeField]. Sada je pokret serijaliziran. Sada, kada odaberemo u hijerarhiji playera, u inspectoru se pojavljuje novo polje „Movement Speed“. Također,kako bi postavku mogli koristiti, potrebno je još pod myRigidBody.velocity, u parametru

komponente Vector2 horizontal pomnožiti s 2. Dakle, odgovarajuća brzina za veličinu playera biti će 5.



Slika 4.14 „Movement Speed" te njena zadana vrijednost u Inspector, kreirala autorica rada

Update funkcija bazirana je zapravo na frame rateu računala, dakle ako naše računalo podržava 100 frameova po sekundi, Update će biti pozvan 100 puta po jednoj sekundi, i animacije će biti nefiksna i možda pre brza. Kako bi se ovo preventiralo, umjesto Update upisuje se FixedUpdate, pa će brzina animacija biti fiksna bez obzira na hardverske mogućnosti računala. [5]

4.3.1. Zrcalno okretanje playera na lijevu io desnu stranu

Za ovu postavku potrebno je definirati private bool varijablu u public klasi Player. Naziv će biti „facingRight". Dakle ova varijabla naznačuje da li je player okrenut lijevo , a varijabla bool tipa moći će imati dvije vrijednosti, točnu ili netočnu, i zato nije potrebno deklarirati još jednu varijablu za lijevu stranu. Ispod funkcije HandleMovement napravljena je zatim još jedna funkcija koja će pozivati ovu varijablu private Void flip, a ta funkcija mora se također pozivati na float varijablu horizontal, pa se unutar zagrade pod parametar upisuje float horizontal. Ako u Unityju označimo Playera, u Inspectoru se može vidjeti, pod Transform sekcijom Scale opcija. Ako na x-u napišemo -1, player će se okrenuti zrcalno u lijevu stranu. No to se mora definirati u skripti, a budući da je veličina karaktera već skalirana dolje i poprima vrijednost 0.09743953, pod private void Flip funkciju unutar vitičastih zagrada potrebno je dodati liniju:

```
if (horizontal = > 0 && !facingRight || horizontal < 0 && facingRight)
```

Ova if tvrdnaj znači sljedeće. Ako je horuizontalna x vrijednost jednaka ili veća od 0 i nije okrenuta lijevo ili je horizontalna x vrijednost manja od 0 i okrenuta je lijevo izvršuje se neki događaj koji se piše unutar vitičastih zagrada, a to je:

```
facingRight = !facingRight;
```

Dakle ako je tvrdnja u if-u ispunjena, tada vrijednost facingRight poprima suprotnu vrijednost od zadane vrijednosti facingRight, i u tom slučaju, player se okreće lijevo. Također se dodaje:

```
Vector3 theScale = transform.localScale
```

Ispod prethodno opisanog, referenciramo se na Scale parametre u Inspectoru, ali je bitno pogoditi samo x os, pa se dodaje također

```
theScale.x *= -1;
```

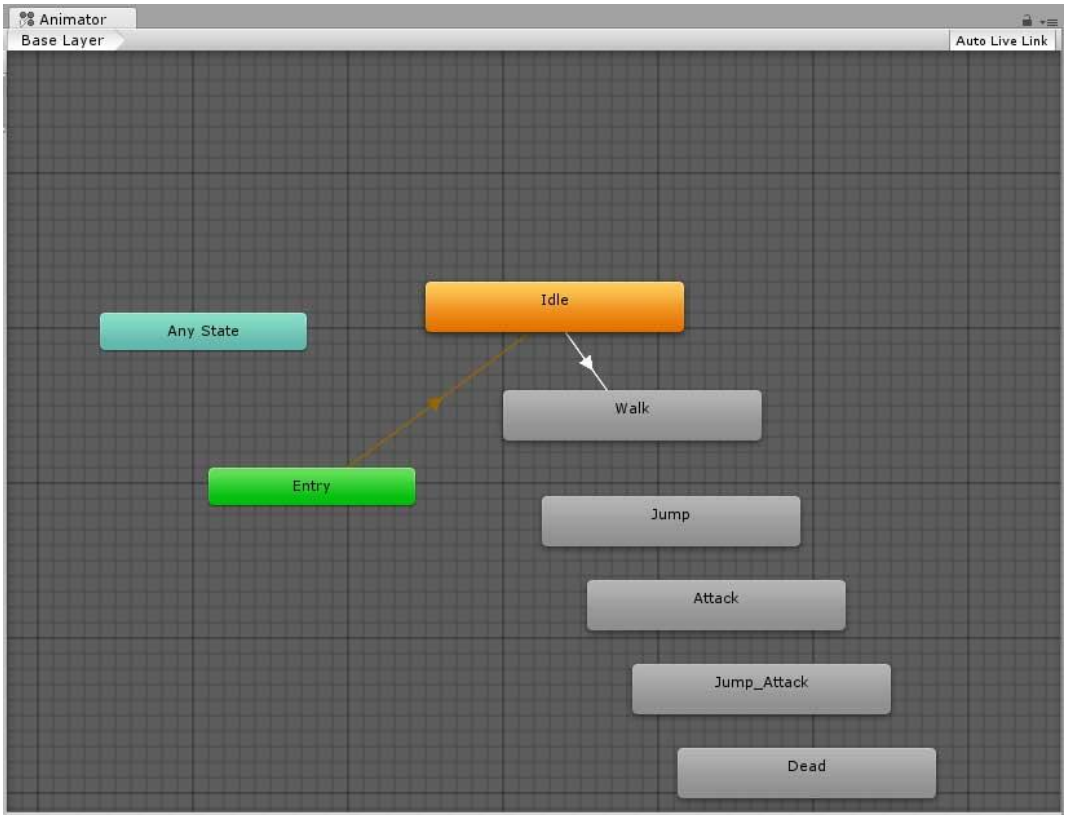
što znači da se varijabla theScale množi s negativnom vrijednošću broja -1 i to samo na x osi, jer svaki broj pomnožen sa negativnom jedinicom rezultira istim tim brojem različitog predznaka. Zadnja linija koja se dodaje u funkciju je:

```
transform.localScale = theScale;
```

Što znači da, kada je vrijednost promijenjena na minus ili plus, stavlja se u localScale varijablu i u konačnici utjeće na poziciju playera. [5]

4.3.2. Implementacija Walk animacije

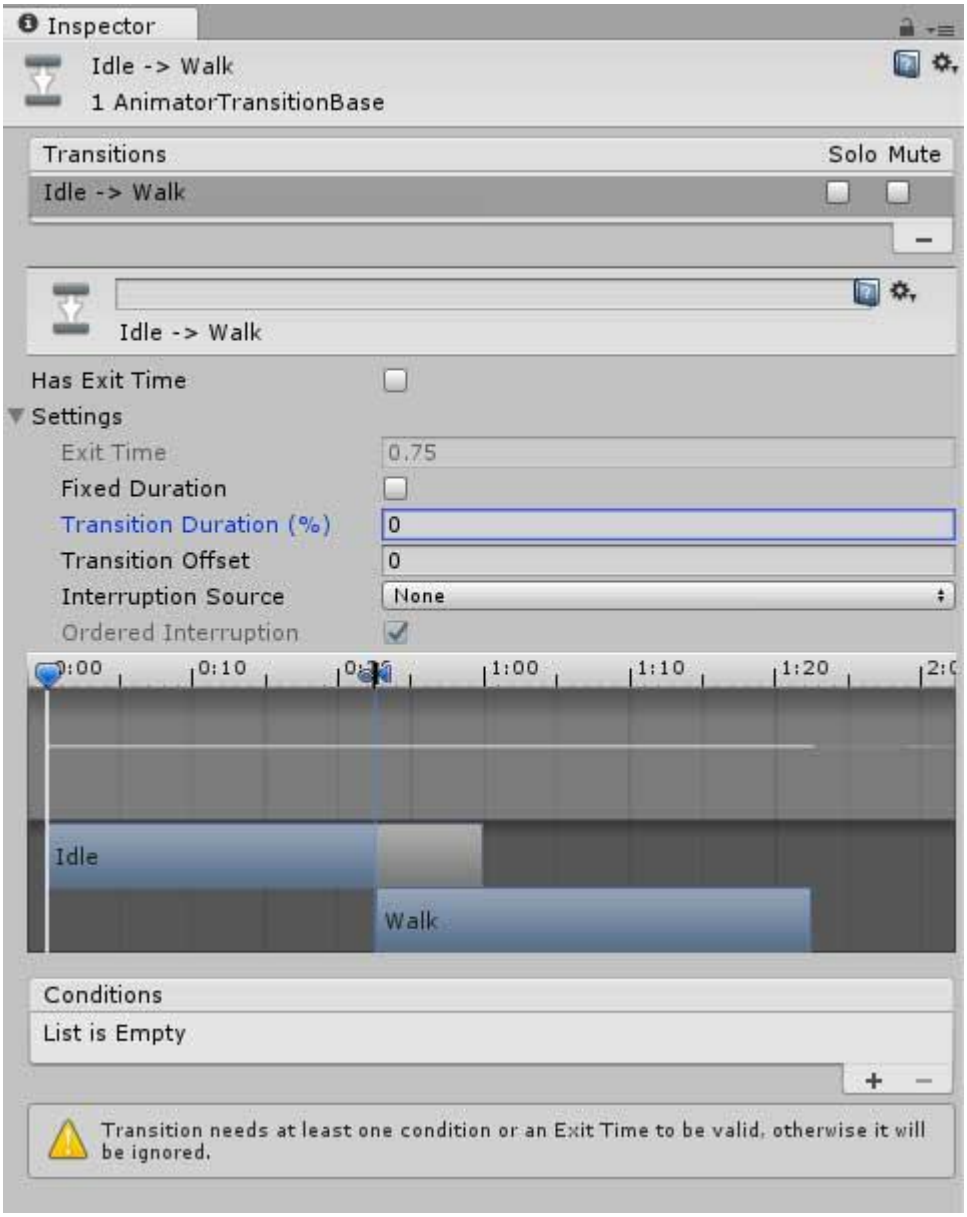
Obzirom da se u Animator prozoru već nalaze sve potrebne animacije, sa zadanom Idle animacijom, potrebno je napraviti tranziciju iz animacije Idle u animaciju Walk.



Slika 4.15 Animator prozor sa tranzicijom iz Idle u Walk animaciju, kreirala autorica rada

Potrebno je samo označiti entitet player u hijerarhiji a zatim desnim klikom na Idle kučicu odabrati „Make Transition“ te kliknuti na Walk, te se pojavljuje strelice. Sada treba dodati uvjete tranzicije.

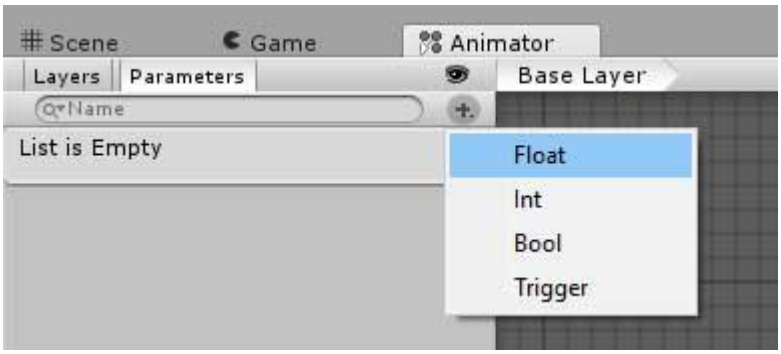
Označivši bijelu strelicu tranzicije, u Inspectoru se otvara novi dijalog s opcijama, gdje treba odznačiti kućicu „Exit Time“ i „Fixed Duration“, a „Transition Duration“ također treba biti na nuli.



Slika 4.16 Prikaz tranzicijskih opcija u Inspectoru, kreirala autorica rada

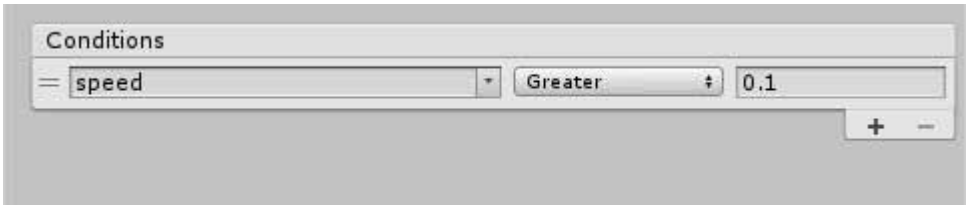
Sada će se tranzicija odvijati odmah po izvršenju zadanog uvjeta. Tranzicijsko vrijeme ima više smisla u 3D igrama kada je potrebno neko vrijeme za gladak prijelaz iz animacije u animaciju. Sada treba dodati funkcionalnost animaciji i uvjete tranzicije. To se radi klikom na plusić u Inspectoru pod sekcijom „Conditions“.

Zatim se u Animator prozoru pod „Parameters“ ponovno klikne na plusić te se dodaje novi Float parametar, jer je u ovoj igri brzina bazirana na float varijablama u skripti. Parametru je dodijeljeno ime „speed“. Bitno je zapamtiti da je parametar „speed“ bez velikih slova kako bi se nadalje takav naziv mogao koristiti u skripti pisanoj jezikom C# koji je osjetljiv na veličinu slova. Ovaj parametar nema nikakve veze s varijablom movementSpeed u skripti „player“.



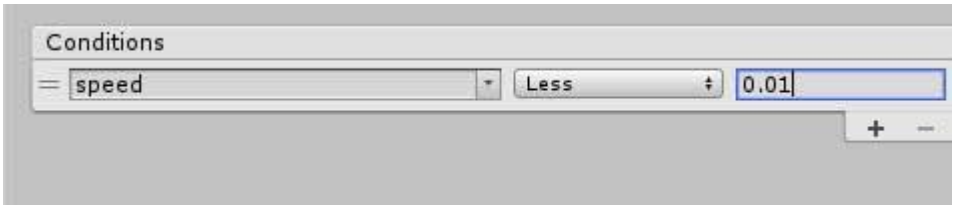
Slika 4.17 Dodavanje uvjeta tipa „float“, kreirala autorica rada

Sa označenom bijelom strelicom, ponovno se u Inspectoru pod Conditions odabire u padajućem izborniku parametar „speed“ koji je upravo stvoren.



Slika 4.18 Postavke za implementaciju float uvjeta „sped“ za tranziciju Idle – Walk, kreirala autorica rada

U Unityju je zatim pod parametrom speed kao uvjet upisan broj 0.01, što znači da kada god brzina animacije po horizontali promijeni brzinu veću od nule, doći će do tranzicije. Međutim, još uvijek ne možemo prijeći iz animacije Walk u animaciju Idle, pa je potrebno na isti način napraviti i reverznu tranziciju u Animatoru. Isto se mora ukloniti Exit time i Fixed Duration. Tranzicija će se odvijati bazirano na parametru speed, ali uvjet neće biti „Greater“ od 0, već „Less“ te upisati 0.01. Jer kada se brzina smanji na manje od 0.01, tranzicija animacija će se opet odviti, ali u obrnutom smjeru, odnosno u idle.



Slika 4.19 Postavke za implementaciju float uvjeta „sped“ za tranziciju Idle – Walk, kreirala autorica rada

Do sada već postoji referentna skripta za komponentu RigidBody2D, no potrebno je dodati još jednu za komponentu Animator. Zato u već postojećoj skripti dodajemo novu varijablu unutar javne klase player:

```
private Animator myAnimator;
```

Zatim se u funkciji Void Start poziva ta varijabla:

```
myAnimator = GetComponent<Animator>();
```

Sada kada je napravljena referenca za Animator funkciju u Unityju, mora se napisati kod za animiranje playera. U funkciji handleMovement dodaje se:

```
myAnimator.SetFloat ("speed", Mathf.Abs (horizontal));
```

Zbog uvijeta da se tranzicija iz Idle u Walk animaciju odvija ako je brzina manja od 0.01, a manje može biti i negativna vrijednost, dodana je ključna riječ Mathf.Abs koja se brine da se uvijek vraća pozitivna vrijednost. [5]

4.3.3. Implementacija Attack animacije

Za Attack animaciju radi se tranzicija iz Walk i Idle animacije. Attack animacija nije namijenjena u ovom slučaju da se događa istovremeno kao walk animaciji, pa će se tranzicija raditi u jednom smjeru za Walk te u dva smjera za Idle, jer kada se attack prestaje odvijati, želimo da se player vrati u stanje idle animacije prvo. Uvjeti za ove tranziciju iz Idlea u Attack kao i iz Walka u Attack će biti parametar tipa trigger te dodijeljenog imena „attack“, a kada se taj uvjet ispuni radnja će se izvršiti. Razlika između boola i triggera je u tome što tvrdnja za bool mora biti istinita ili neistinita, a trigger se jednostavno izvršava svaki puta kada se uvjet ispuni. Za tranziciju iz attacka u idle, u ovom će slučaju biti potreban parametar Exit time, jer se Idle mora početi izvršavati odmah nakon

što player prestane napadati. Exit time za prijelaz iz Attacka u Idle postavljen je na 0.32 sekunde, jer je tako prijelaz izgledao najbolje. Cijela Attack animacija, onako kako je napravljena u Spriteru se dakle neće odvijati jer tako izgleda bolje, a playeru u tranziciji mač gotovo ostaje na mjestu gdje stoji i kada se izvršava Idle. Kada bi Exit time bio veći, player bi vraćao mač gore, pa bi ga opet spuštao na poziciju gdje se nalazi u Idle animaciji, a to ne bi izgledalo dobro.
Tranzicija se treba triggerati iz skripte, pa se stvara nova varijabla:

```
private bool attack;
```

i nova funkcija:

```
    private void HandleAttacks ()
{
     if (attack) {
         myAnimator.SetTrigger ("attack")

     }
```

U nazivu stoji „Attacks“ zato jer će se poslije dodavati uvjet i za Jump Attack animaciju. Sada još treba dodati funkcionalnost za trigger:

```
    private void HandleInput()
{
     if ((Input.GetKeyDown(KeyCode.W)) || (Input.GetKeyDown(KeyCode.UpArro
w)) )
     {
         jump = true;
     }
```

U ovoj funkciji nalazi se if tvrdnja koja traži pritisak Alt tipke ili lijevi klik miša, i ako se ta tipka pritisne, izvršava se napad. Budući da je ova funkcija tipa void, neće se izvršavati ako je ne pozovemo u Update funkciji, jer tvrdnje unutar Update funkcije traže i izvršavaju svake sekunde, pa se iznad dodaje nova Update funkcija u kojoj se poziva funkcija HandleInput:

```
    void Update ()
{
     HandleInput();
}
```

HandleAttack se također mora pozivati i u FixedUpdate funkciji, samo preko jedne linije kako bi animacija zadržala vremensku konzistentnost:

```
HandleAttacks ();
```

Kako se animacija ne bi izvršavala u petlji nakon pritiska na Alt, skroz na dnu glavne funkcije dodaje se funkcija:

```
    private void ResetValues ()
{
    attack = false;
}
```

te se zatim poziva ponovno u FixedUpdate funkciji.

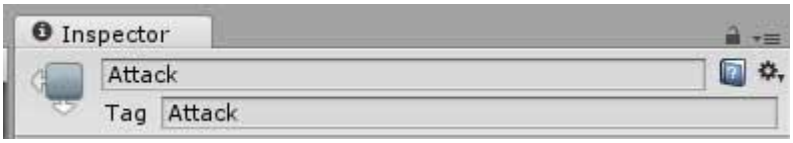
Sada je Attack animacija funkcionalna, međutim, ako držimo tipku a ili d, te pritisnemo Alt, obje animacije se izvršavaju u isto vrijeme,a player se skliže te u isto vrijeme obavlja attack, pa se unutar HandleMovement funkcije dodaje nova if tvrdnja:

```
        if (!this.myAnimator.GetCurrentAnimatorStateInfo(0).IsTag("Attack"
))
    {
 myRigidbody.velocity = new Vector2(horizontal * movementSpeed, myRigidbody
.velocity.y);
```

Unutar Animatora, postoje layeri na kojima se odvijaju animacije, a za playera postoji samo jedan layer zadanog naziva „0“, i zato kod StateInfo u zagradi stoji nula, kako bi tvrdnja dohvatila layer 0.

Nakon toga samo treba u Unityju u polje Tag upisati „Attack“, pa ako animacija nije labelirana s oznakom „Attack“, player se može pomicati. Dakle kada se izvršava radnja napada, player se ne može micati, ali i dalje može napadati ako se najprije pritisnu kontrole za pokret i naknadno tipka Alt. To se događa jer unutar if tvrdnje i dalje postoji velocity, pa se mora dodati reset funkcionalnost za velocity, pa unutar if tvrdnja u Handle Attack dodajemo još jedan uvjet:

```
myRigidbody.velocity = Vector2.zero;
```



Slika 4.20 Postavljanje „Tag“ komponente za „Attack“ animaciju, kreirala autorica rada

Testiranjem igre sa dodanom funkcijom za napada, uočeno je da sve funkcionira na željeni način. [5]

4.3.4. Jump funkcionalnost i collideri

Slika 4.21 Smještaj „Box Collider 2D“ komponente na playera, kreirala autorica rada

Kako bi player mogao biti u interakciji s platformama i drugim objektima, potrebno ih je povezati s objektima koji se zovu Collideri. Collider se dodaje oznakom na Add Component > CoxCollider2D. Oko objekata i playera stvara se pravokutnik kojemu se može podesiti veličina i pozicija Također, uvjet kolizije je gravitacija, pa se u Inspectoru s označenim playerom treba gravitacija postaviti na vrijednost 1, umjesto zadane vrijednosti 0.

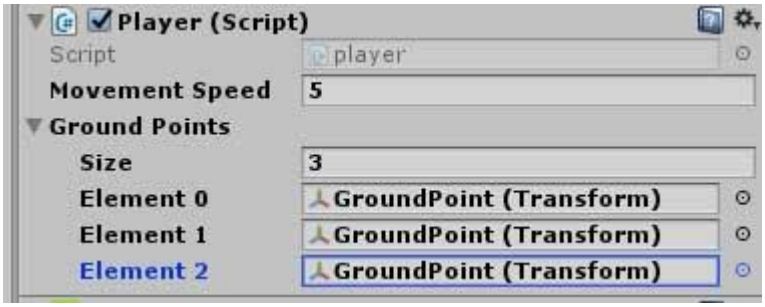
S dodanim colliderom na platformu i playera, kod testiranja igre može se vidjeti da nakon kratkog hoda, player počinje padati na tlo. Unutar Rigid Body > Constraints sekcije u Inspectoru, s označenim playerom zato moramo označiti kućicu Freeze Position Z, kako ne bi dolazilo do rotacije na Z osi. Pozicija se također može zamrznuti na X i Y osi, no kada bi to napravili, player se ne bi mogao micati po tim osima.

S dodanom fizikom, možemo napraviti jump animaciju, kako bi player imao na što pasti nakon što skoči, jer bi u suprotnosti padao u beskonačnost.

Kako bi prepoznao skok, Unity mora znati kada je igrač na tlu a kada je u zraku, pa se playeru dodaje „grounding point“ desnim klikom u prozoru hijerarhije na Create Empty. Novu prazan objekt naziva GameObject zatim se odvuče na playera u hijerarhiji kako bi mu bio primijenjen, a točka se u sceni pozicionira na mjesto ispod njegovih nogu. Prva točka se pozicionira na dno njegove lijeva, koja se zatim duplicira se stavlja na dno desne noge. Radnja se treba još jednom ponoviti te se nova točka odvući u središte. Sva tri objekta preimenovana su u naziv „GroundPoint“. Sada playeru treba referenca na te točke, pa u skripti na vrhu ispod ostalih varijabli dodajemo novi niz:

```
private Transform[] groundPoints;
```

Kako bi vidjeli ovo u Inspectoru dodajemo [SerializeField] iznad. Sada u Inspectoru imamo novu sekciju GroundPoint, te kao veličinu pod „Size“ upisujemo broj 3 budući da imamo tri točke. Istog trenutka stvaraju se nova 3 polja, a zatim na svako od njih iz hijerarhije odvlačimo jednu po jednu točku za svako polje, u to polje.



Slika 4.22 Elementi komponente „Ground Points“ u Inspectoru, kreirala autorica rada

Također će nam trebati i:

```
[SerializeField]
private float GroundRadius;
[SerializeField]
private LayerMask whatIsGround;
```

Ground Radius će se koristiti u sljedećoj funkciji, a u Unityju će njena vrijednost biti postavljena na 0.2. U skripti se dodaje nova funkciju koja provjerava stoji li igrač na tlu. [4]

Ako je brzina na y osi jednaka ili manja od nule, tada igrač stoji na tlu.

```
    private bool IsGrounded()
    {
        if (myRigidbody.velocity.y <= 0)
        {
            foreach (Transform point in groundPoints)
            {
                Collider2D[] colliders = Physics2D.OverlapCircleAll (point.po
sition, GroundRadius, whatIsGround);

                for (int i = 0; i <colliders.Length; i++)
                {
                    if (colliders [i].gameObject != gameObject)
                    {
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

Tvrdnja:

```
if (colliders[i].gameObject != gameObject)
```

provjerava je li objekt s kojim collider kolizira player, odnosno, nešto što nije gameObject, a to je player. Ako se ispuni uvjet, vraća se vrijednost true, a ako ne onda se vraća vrijednost false. Da bi funkcija radila, treba deklarirati dvije bool varijable i jedna float varijabla:

```
private bool isGrounded;

private bool jump;

private float jumpForce;
```

Potonja treba biti serijalizirana kako bi je se mogla dohvatiti iz Unityja, a isGrounded varijablu potrebno je pozvati u FixedUpdate funkciji:

```
IsGrounded = IsGrounded ();
```

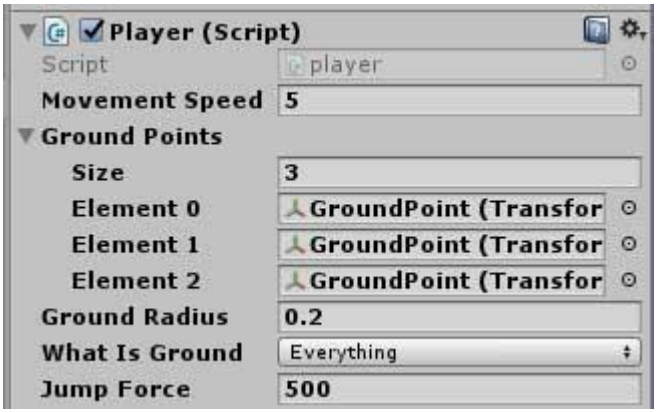
Dakle vrijednost false ili true iz IsGrounded funkcije će se pridružiti IsGrounded varijabli nakon izvršenja te funkcije. Funkcionalnost se zatim dodaje u funkciju HandleMovement preko if tvrdnje:

```
if (IsGrounded && jump)
{
 IsGrounded = false;
    myRigidbody.AddForce (new Vector2 (0, jumpForce));

}
```

Uvjet u zagradama indicira da, ako je igrač stoji na tlu, a pritisnuta je kontrola za skok, player više nije na tlu, već u zraku, a druga se poziva na jumpForce varijablu te se rigidnom tijelu na y osi dodaje nova sila jumpForce, dok x završava vrijednost 0. Unutar HandleInput funkcije dodaje se kontrola za skok, a to će biti slovo „W“ ili strelica gore:

```
if ((Input.GetKeyDown(KeyCode.W)) || (Input.GetKeyDown(KeyCode.UpArrow)) )
```

Dakle, sve varijable koje su serijalizirane pojavile su se u inspektoru, pa se tako na lakši način mogu manipulirati. Po zadanome na „What Is Ground“ bilo je označeno „Nothing“ pa je potrebno označiti „Everything“, te Jump Force postaviti na optimalnu vrijednost kako skok nebi bio pre velik ili pre malen.



Slika 4.23 Korištene vrijednosti za „Ground Radius“ i skočnu silu

Sada player skače, no ono što se događa jest da, ako držimo tipku za skok te istovremeno tipku za pomicanje lijevo desno, možemo vidjeti da on hoda u zraku, a također, za skok još uvijek nije primijenjena animacija napravljena u Spriteru.Kako bi spriječili hodanje po zraku dodajemo novu, serijaliziranu bool varijablu:

```
    private bool airControl;
```

pa se i sljedeću liniju u funkciji HandleMovement:

```
    myRigidbody.velocity = new Vector2(horizontal * movementSpeed, myRigid
body.velocity.y);
```

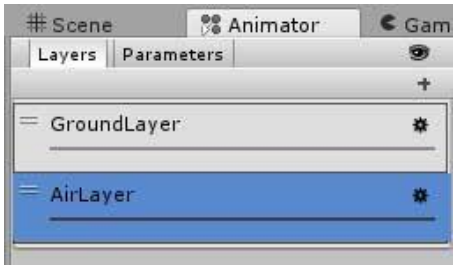
dodaju novi parametri za ispunjavanje uvjeta pokreta, nakom čega tvrdnja izgleda ovako:

```
if (!this.myAnimator.GetCurrentAnimatorStateInfo(0).IsTag("Attack") && (isGro
unded || airControl))
        {
            myRigidbody.velocity = new Vector2(horizontal * movementSpeed, my
Rigidbody.velocity.y);
        }
        if (isGrounded && jump)
        {
            isInAir = true;
            isGrounded = false;
            myAnimator.SetBool ("jump", true);
            myRigidbody.AddForce (new Vector2 (0, jumpForce));
        } myAnimator.SetFloat ("speed", Mathf.Abs (horizontal));
```

Dakle pomak po x osi se događa ako se ne izvodi animacija labelirana s tagom „Attack“, i ako je on prizemljen ili na njega djeluje varijabla „airControl“. Sada ga ne možemo pomicati kada je u zraku, no ako pritisnemo prvo strelice za pomak a zatim „W“ on i dalje hoda po zraku, a to će se popraviti kada se na skok primjeni sama animacija skoka. [5]

4.3.5. Implementacija Jump animacije

Jump Animacija ne može se primijeniti samo tako na skok, jer je animaciju najprije potrebno razdijeliti na dvije zasebne animacije. To su već spomenute TakeOff animacija i Landing animacija. U tu svrhu potrebno se koristiti layerima u Animator prozoru te načiniti dva layera.

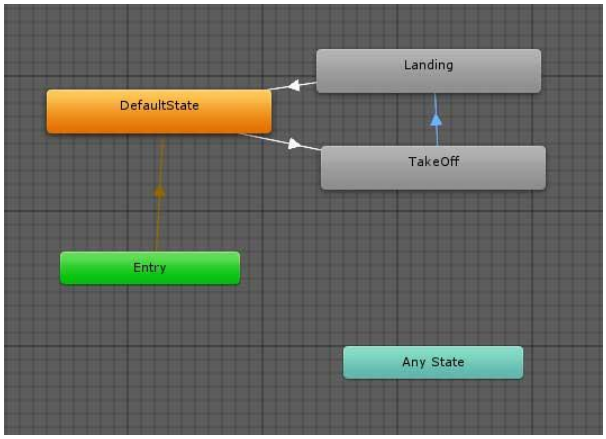


Slika 4.24 Layeri u Animatoru, kreirala autorica rada

GroundLayer služit će za kontrolu svih animacija koje se odvijaju kada je player prizemljen, dok će u suprotnosti AirLayer kontrolirati pokrete u zraku. U našem slučaju to su animacije Jump i JumpAttack. Sa označenim AirLayer-om, u Animator prozor dodajemo TakeOff i Landing animacije. Budući da je GroundLayer nulti po redu, prvi se izvršava po zadanim postavkama. Layeri koriste komponentu „Weight“ kako bi animacije layeri sa svojim animacijama prelazili iz jednog u drugi. Weight poprima vrijednosti od 0 do 1, a što je vrijednost, primjerice, TakeOff animacije veća, manje će na nju utjecati Idle animacija koja je zadana animacija u GroundingLayer-u. Ovo se mora učiniti putem koda na način da, kada se pritisne „W“ ili strelica gore, animacija prelazi iz idlea ili Walka u TakeOff animaciju, a vrijednost koju će Weight poprimiti indicira stupanj stapanja jedne animacije u drugu. Loop Time mora biti isključen za obje animacija skoka. [5]

Za tranzicije je potrebno dodati dva nova parametra, jednog tipa bool pod nazivom „land“, a drugog tipa trigger pod nazivom „jump“, te načiniti praznu kućicu desnim klikom na prazno polje u Animatoru te odabirom na Create>Empty State. Dodijeljen joj je naziv DefaultState te je desnim klikom na opcijama odabrano „Set as Default State“, kako bi se prva izvršavala. Radi se tranzicija

kao na slici tako da sa zadanog stanja player može ući u TakeOff, pa u Landing, i onda ponovno u DefaultState.



Slika 4.25 Dijagramski prikaz animacija u Animatoru za „Air“ layer, kreirala autorica rada

Za sve strelice (tranzicije) u Inspectoru je potrebno odznačiti sve postavku Exit Time te ostale parametre postaviti na nulu, te se moraju dodati uvjeti. Tranziciji Landing-DefaultState dodaje se uvjet „land“ koji se postavlja na „false“, TakeOff-Landing također se postavlja uvjet „land“ ali se postavlja na „true“, a tranziciji DefaultState – TakeOff dodijeljuje se trigger uvjet „jump“. Daljnje funkcionalnosti mogu se dodati jedino u skripti. Kako bi se kodom moglo utjecati na layere, dodaje se nova funkcija :

```
    private void HandleLayers()
{
    if (!isGrounded) {
        myAnimator.SetLayerWeight (1, 1);
    } else
    {
        myAnimator.SetLayerWeight (1, 0);
    }
}
```

Kada player nije prizemljen, odnosno kada je u zraku, skripta govori layeru broj 1 (što je AirLayer), da promijeni Weight vrijednost u 1, a promjena te vrijednosti u 1 znači da se će se animacije s prethodnog layera prestati izvršavati. Inače (else), dakle ako player nije u zraku, „Weight“ vrijednost prvog layera prelazi u nulu i to omogućava svim animacijama na layeru GroundLayer da se mogu izvršiti. HandleLayers funkcija se zatim poziva u FixedUpdate funkciji.

Kako player ne bi idlea-o prilikom skoka slijedi triggeranje „jump“ parametra kako bi se TakeOff mogao izvršiti pritiskom na već zadane kontrole „W“ ili strelica gore, pa se u unutar vitičastih zagrada u funkciji HanfleMovement tvrdnji:

```
if (isGrounded && jump)
```

dodaje

```
myAnimator.SetTrigger (jump);
```

Za ispunjenje uvjeta kod tranzicije Landing – DefaultState, treba provjeriti točnost „land“ bool parametra, pa u posljednjoj if tvrdnji u funkciji IsGropunded dodajemo:

```
myAnimator.SetBool ("land", false);
```

a u funkciji HandleMovement :

```
if(myRigidbody.velocity.y <0)
{
    myAnimator.SetBool ("land", true);
}
```

[5]

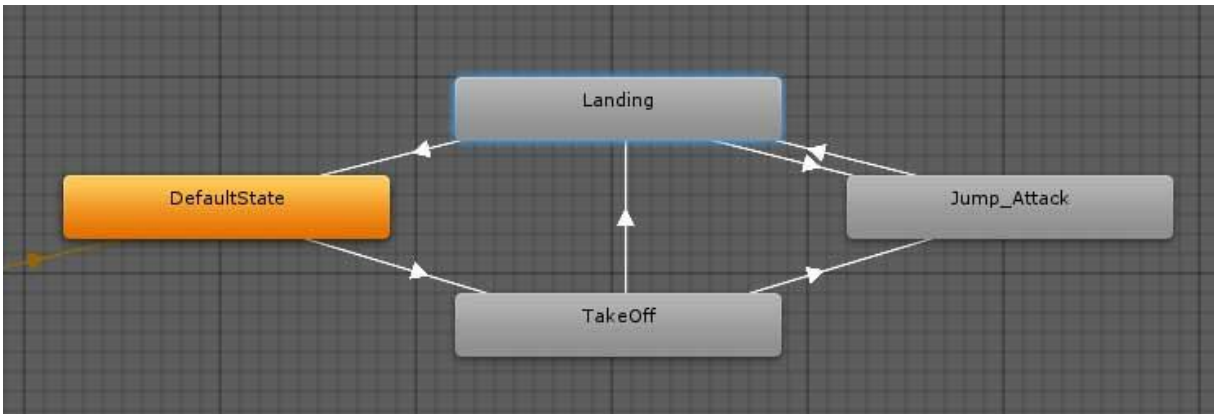
4.3.6. Implementacija Jump Attack animacije

U ovom trenutku u igri, ako pritisnemo kontrolu za napad kada je player u zraku, animacija skoka će se prestati izvoditi te će player odmah prijeći u „landing“ poziciju i početi padati, pa se u funkciji HandleMovement unutar funkcije HandleAttack dodaje još jedan uvjet u if tvrdnju, a to je IsGrounded

```
if (attack && isGrounded && !this.myAnimator.GetCurrentAnimatorStateInfo(0
).IsTag("Attack"))
```

Dakle uz već ranije opisane uvjete, Attack animacija može se izvršavati samo kada je player prizemljen, odnosno kada se ne izvodi funkcija IsGrounded, to jest kada ona vraća rezultat „true“, a to se događa samo kada je player prizemljen i nije u zraku.

Sada nije moguće prekinuti jump animaciju pritiskom na kontrolu za napad pa igrač neće pasti, no još treba dodati mogućnost da se može napasti u zraku a da se „Jump“ animacija i dalje izvodi. Najprije treba JumpAttack animaciju dodati u AirLayer u Animatoru.



Slika 4.26 Dijagramski prikaz animacija i njihovih tranzicija u Air layeru s dodanom „Jump_Attack“ animacijom, iz koje je moguće prijeći u „Landing“ u oba smjera, kreirala autorica rada

Tranzicije je potrebno napraviti iz TakeOff animacije u Jump_Attack animaciju u jednom smjeru, te iz Jump_Attack u Landing u oba smjera, te dodati novi parametar koji će se koristiti kao uvjet, tipa bool i naziva „jumpAttack“. Za prijelaz iz TakOff animacije u Jump_Attack potrebno je opet isključiti Exit Time i ostale opcije, te kao uvjet označiti parametar jumpAttack te ga postaviti na „true“. Za prijelaz iz Landing animacije u Jump_Attack treba postaviti identične postavke i isti uvjet također postaviti na „true“, dok se za reverzibilni prijelaz iz Jump_Attack u Landing ništa ne treba mijenjati, dakle Exit Time mora ostati uključen. Također, za prijelaz iz DefaultState u TakeOff potrebno je postaviti uvjet jumpAttack na false kako bi se preventiralo da animacija iz Jump_Attacka dođe natrag na TakeOff.

U skripti se zatim dodaje nova bool varijabla:

```
private bool jumpAttack;
```

te se u ResetValues funkciji poziva:

```
jumpAttack = false;
```

kako bi se animacija sa svojim uvjetima resetirala svaki puta kada je jumpAttack izvrši.

U HandleInput funkciji zatim treba dodati „jumpAttack = true;“ u već postojeću if tvrdnju za kontrolu napada pritiskom na Alt ili lijevi klik miša.

```
            if ((Input.GetKeyDown (KeyCode.LeftAlt)) || (Input.GetMouseButtonD
own(0)) )
        {
            attack = true;
            jumpAttack = true;
        }
```

U HandleAttack funkciju dodaje se nova tvrdnja:

```
            if (jumpAttack && !isGrounded && !this.myAnimator.GetCurrentAnimat
orStateInfo (1).IsName ("Jump_Attack"))
        {
            myAnimator.SetBool (jumpAttack = true);
        }
```

Dakle ako je pozvan jumpAttack i player nije prizemljen i Jump_Attack animacija se već ne izvodi na prvom layeru, jumpAttack uvjetu se dodaje istinita vrijednost, odnosno animacija kojoj je taj uvjet istinit može se početi izvoditi. No potrebno je još uvjeta pa se dodaje još jedna if tvrdnja u istoj funkciji.

```
            if (!jumpAttack && !this.myAnimator.GetCurrentAnimatorStateInfo (1
).IsName ("Jump_Attack"))
        {
            myAnimator.SetBool (jumpAttack = false);
        }
```

Ako je jumpAttack uvjet neispunjen, odnosno vrijednost mu je postavljena na „false“, i Jump_Attack animacija se već ne izvodi, jumpAttack uvjetu se dodijeljuje vrijednost false, te se u tom slučaju ona neće izvoditi.

Naknadno je brzina Jump_Attack animacije smanjena u pola (0.5), te je Exit Time za tranziciju Jump_Attack-Landin postavljen na vrijednost 0.1, jer se animacija odvijala pre brzo a pokret napada je kasnio na pritisak kontrole.

S implementacijom Jump Attack animacije komponenta „player“ je potpuno funkcionalna te slijedi gradnja scene i dodavanje ostalih funkcionalnosti kao što je pratnja kamere, gubitak ili pobjeda, kreiranje levela, mogućnost sakupljana novčića i ostalo. [5]

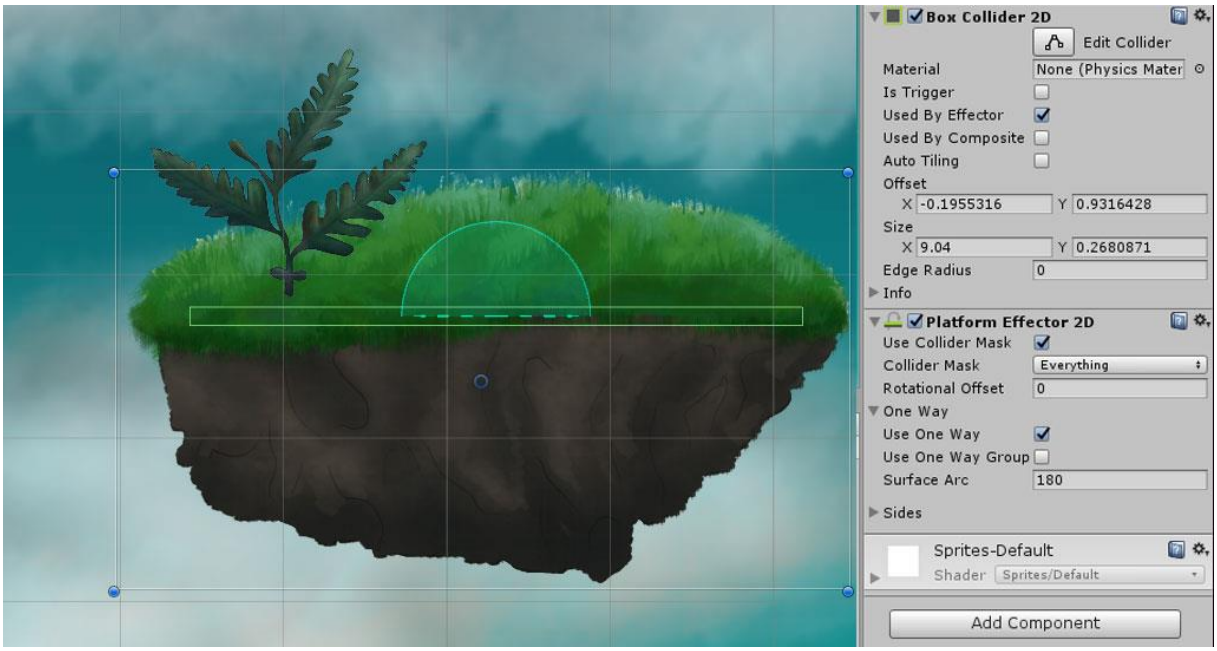
4.4. Platforme

Kako bi player mogao biti u interakciji s platformama, platformi se dodaje „BoxCollider2D“. To općenito vrijedi za bilo koji objekt s kojim želimo kolidirati. U side scrollerima vrlo je česta,

ustvari podrazumijevajuća pojava da platforme imaju mogućnosti da se skoči kroz njih odozdo ili, ako je smještena nisko, prođe kroz nju, i zato se u Unityju mogu dodati dva „Box Collidera 2D“ objekta i programirati ih tako da jedan drugome bude trigger, a lakši način bi bio jednostavno dodati jedan Box Collider 2D objekt te jedan „Platform Effector 2D“.

„Platform Effector“ dodajemo tako da za željenu platformu u Inspectoru označimo „Add Component“ te u tražilicu upišemo „Platform Effector“ te odaberemo isti. Na prethodno primijenjenom Box Collideru zatim označujemo opciju „Used By Effector“, a za efektor mora biti označena opcija „Used One Way“.

Ovaj efektor vrlo je korisna, već gotova komponenta koju daje Unity te ne zahtijeva nikakvo dodatno programiranje u ovom slučaju.



Slika 4.27 Izgled postavki „Box Collider 2D“ i „Platform Effector 2D“ komponenti u Inspectoru

Platforme su često ponavljajući objekti pa je poželjno od svake napraviti „prefab“ kako bi istu sa dodanim efektorom mogli koristiti više puta. [3]

4.5. Kamera

4.5.1. Implementacija prateće kamere

Kamera će biti postavljena tako da slijedi igrača, pa joj je umjesto zadanog naziva dodijeljeno ime „FollowCamera“. Funkcionalnost se dakako mora dodati u skripti. Najprije je u C# skripti dodana varijabla:

```
private Transform target;
```

kojom se želi dohvatiti Transform komponenta targetiranog objekta.

U Start funkciju potrebno je dodati

```
target = GameObject.FindWithTag ("Player").transform;
```

Ovom varijablom referenciramo se na komponentu u igri koja ima naziv „Player“ i njegovu komponentu za transformaciju. Sljedeće se u Update funkciji upisuje

```
transform.position = new Vector3 (target.position.x, target.posit
ion.y, transform.position.z);
```

Ne želimo mijenjati Z os na kojoj je kamera, i zato je treći parametar Vector.3 objekta postavljen na transform.position.z, dok su x i y os targetirane.

Međutim, postoje neke situacije kada nije poželjno da kamera slijedi igrača, već da se prestane micati, primjerice na kraju levela, ne želimo da se otkrije rub platforme ili nekog drugog vizualnog elementa. Zato treba dodati granice pa se deklariraju 4 nove varijable:

```
public float boundsTop = 0.0f;
public float boundsBottom = 0.0f;
public float boundsLeft = 0.0f;
public float boundsRight = 0.0f;
```

Sve su public klase kako bi bile dostupne iz editora. U Update funkciji potrebno je referencirati pristup Mathf frameworku te pozvati „bounds“ varijable za x i y parametre na Vectoru 3. To znači da pozicija kamere nikada neće ići niže na x osi od lijeve ili više od granice desne strane. Isto vrijedi i za y os. Sada void Update funkcija izgleda ovako:

```
void Update () {

transform.position = new Vector3 (Mathf.Clamp(target.position.x,bound
sLeft, boundsRight), Mathf.Clamp(target.position.y, boundsBottom, boundsTop),
 transform.position.z);
    }
```

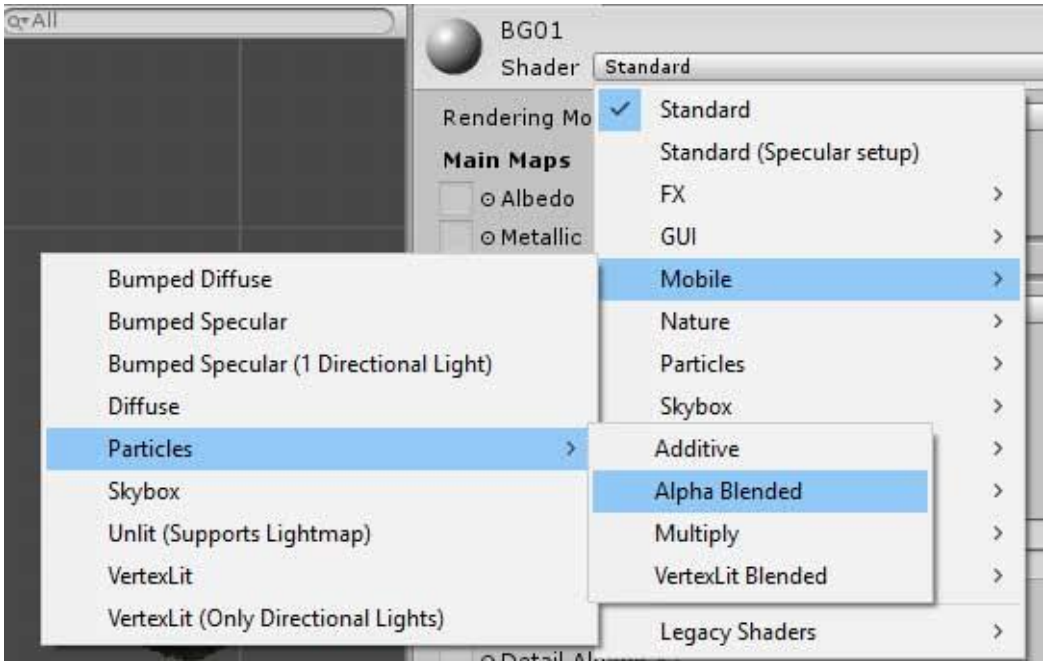
U Unityju je lako izmjeriti krajnju vrijednost pozicije kamere tako da jednostavno kameru odvučemo gdje želimo da kamera stane. To se može napraviti preko Inspectora u sekciji Transform > Position. U ovom slučaju na x osi ta je vrijednost 103. Nakon što smo izmjerili domet kamere na x osi, pozicija se resetira na nulu, te se u polje boundsRight u Inspectoru upisuje ta vrijednost. Isto se postupa i s lijevom, gornjom i donjom granicom. [3]

4.5.2. Parallax pozadine

Parallax efekt je pojava kada se dvije različite komponente, recimo slike, kreću različitim brzinama, a ta se praksa koristi za pozadine u igrama kako bi imale veću dubinu. U primjeru iz stvarnog života, ako se vozimo u autu, možemo primijetiti kako se red drveća koji nam je blizu, „kreće“ brzo, dok se zgrada u daljini „kreće“ sporije, te nam treba više vremena kako bi ju prestigli. Podrazumijeva se, dakako da se ti objekti ne kreću već stoje na mjestu, ali ono što mi vidimo je pokretna slika. Dubina scenografije dolazi još više do izražaja ako koristimo više slojeva pozadine, na primjer, nebo, planine, oblaci i slično. Pomoću parallax efekta stoga možemo simulirati iluziju tako da izgleda kao da su svi ti grafički elementi na određenoj distanci, bliže ili dalje prvom planu igre. Nebo i oblaci izgledat će kao da se jedva uopće i kreću, zato što su jako daleko.

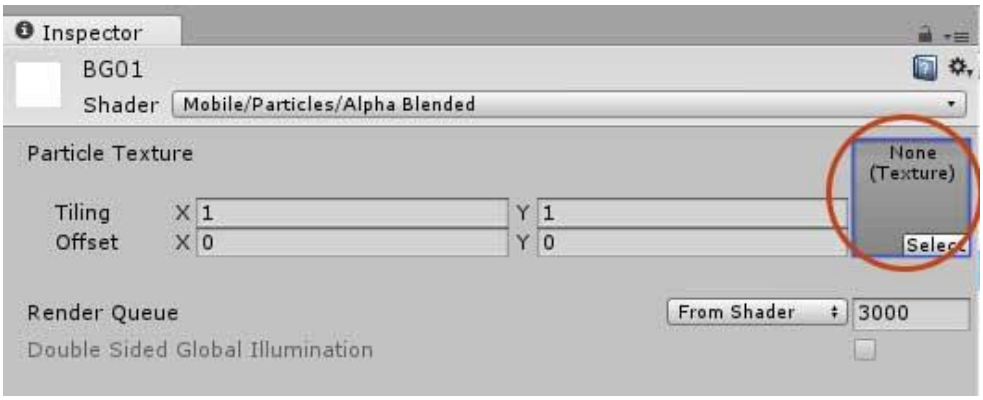
Slika neba koja će se koristiti kao pozadina je u .png formatu je ju je potrebno najprije odvući u Asset mapu pod Backgrounds. Prije nego je odvučemo u scenu, potrebno je označiti da bude tipa „Texture“ ili „Default“, ovisi o inačici Unity Editora. Po zadanome svaka je slika tipa „Sprite (2D and UI). Ovdje će se stoga dogoditi kombinacija 2D i 3D objekata. Da bi tekstura mogla biti primijenjena, potrebno je sljedeće napraviti materijal tako da se u željenoj mapi odabere desni klik>Create>Material, kome se dodjeljuje željeno ime (BG01) te odabiru željena svojstva u Inspectoru.

U sekciji „Shader“ u Inspectoru zatim na padajućem izborniku odabiremo Mobile>Particle>Alpha Blender



Slika 4.28 Putanja odabira tipa teksture „Alpha Blended“ za teksturu pozadine, kreirala autorica rada

Sljedeće što je potrebno je u Inspector s odabranim Shaderom odvući željenu sliku u „Texture“ kućicu.



Slika 4.29 Mjesto označeno crvenom bojom je kućica u koju je potrebno dovući teksturu, kreirala autorica rada

Isti postupak s kreiranjem materijala i primjenom slike za teksturu ponavlja se za svaki pozadinski element za koji ćemo primjenjivati parallax efekt. Napravljeni Shaderi sada se smatraju kao 3D objekti.

Da bi teksturu primijenili na pozadinu stvaramo novi 3D objekt u glavnom izborniku Game Object>3D Object>Quad. Stvara se mali pravokutni objekt u središtu scene kome mijenjamo veličinu tako da pokriva malo veće područje od same scene jer će se ta kućica mijenjati ovisno o rezoluciji zaslona gdje će se igra reproducirati. Quad objektu zatim mijenjamo ime kako bi se lakše mogli snalaziti te isključujemo opciju Mesh Collider u Inspectoru jer neće biti potrebna nikakva kolizija bilo kojeg objekta s pozadinom. U sekciji Mesh Rendere pod Materials > Element 0 zatim označavamo neposredno prije napravljen Shader sa slikom teksture, te se ta tekstura odmah primjenjuje na Quad element u pozadini. Quad objekt nalazi se ispred same scene, a budući da je taj objekt 3D tipa, ne možemo ga smjestiti u neki od layera prethodno napravljenih, pa mu se mijenja pozicija na Z osi. Kada je shader s teksturom primijenjen na Quad objekt, sada promijenjenog naziva „Layer01“, odvući na kameru „Follow Camera“, tako da kamera postane roditelj toga objekta. Sada će se Layer01 micati istovremeno s kamerom.

Zatim je napravljeno još 9 shadera, a na svaki primijenjena po jedna slika oblaka kao tekstura.

Slika 4.30 Izgled oblaka nacrtanih u Photoshopu, kreirala autorica rada

Svaki od ovih oblaka spremljen je kao zasebna sličica u .png formatu s transparentnom pozadinom. Potom je napravljen novi Quad objekt, no ovaj put se u sekciji „Materials“ pod „Size“ upisala brojka 9, budući da je 9 oblaka, te je svaki od prethodno napravljenih Shadera odvučen u Inspector kao zaseban element, te prilagođene „Offset“ i „Tile“ vrijednosti na obje osi kako bi svaki oblak bio pozicioniran na željeno mjesto. Isti layer, naziva „Layer02“ ponovno se odvukao u „CameraFollow“ kako bi „Layer02“ postao dijete kamere, no ovaj puta je primijenjena brzina preko skripte za primjenu parallax efekta.

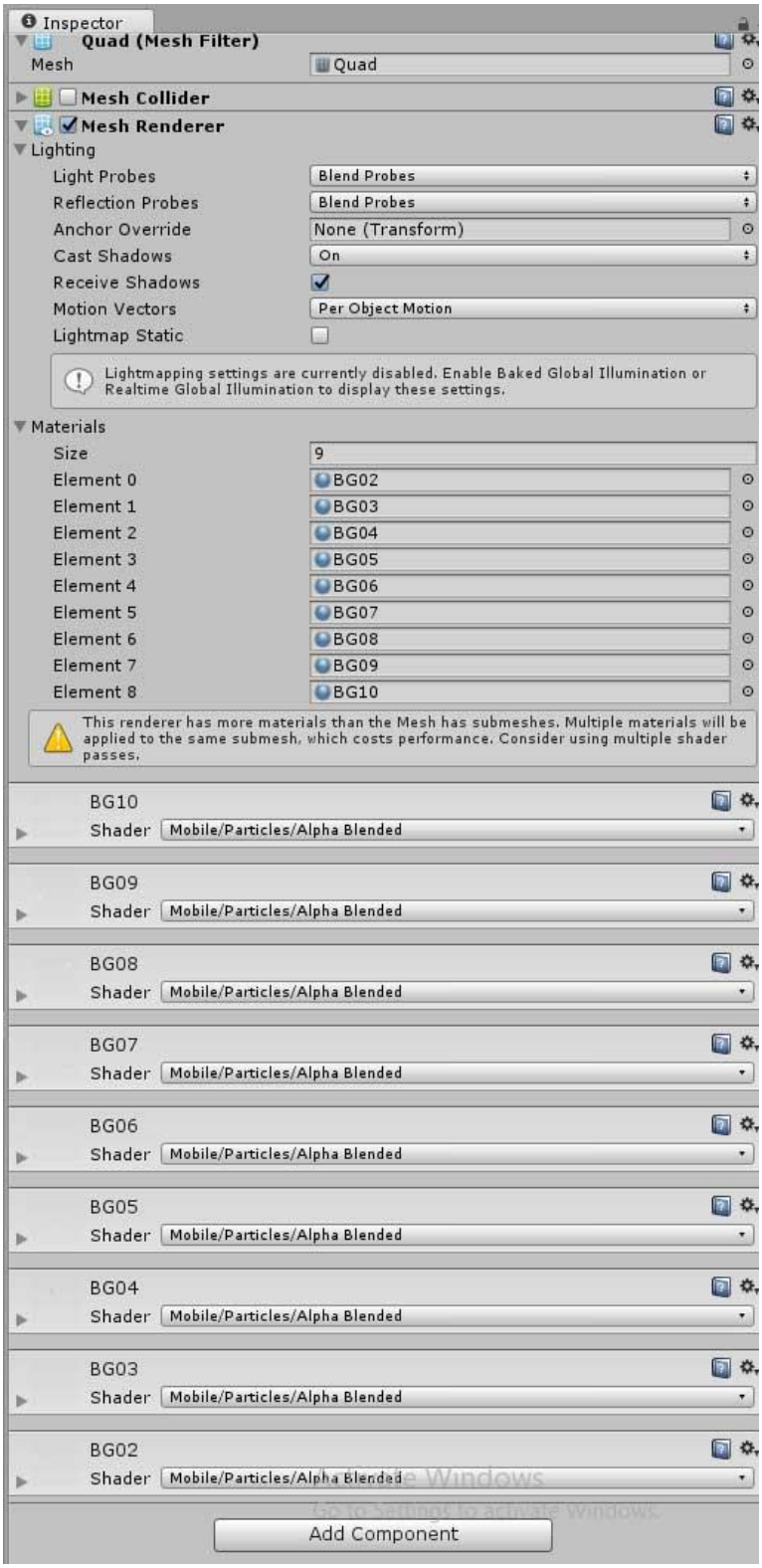
Za parallax se može napraviti nova skripta s referencom na skriptu za kameru, ili se mogu dodati funkcionalnosti za parallax u samu „CameraFollow2D“ skriptu. U glavnoj klasi deklariraju se svije javne varijable tipa „Vector2“ imena „movementFromBase“ i „baseLocation“ kojem se dodjeljuje modifikator „new“ sa zadanim vrijednostima nule po x i Y osi. New mora biti kako bi se sakrili članovi iz naslijeđene klase

```
public Vector2 movementFromBase = new Vector2 (0.0f, 0.0f);
public Vector2 baseLocation = new Vector2 (0.0f, 0.0f);
```

Još dvije varijable koje je potrebno deklarirati su:

```
public float[] layerSpeeds;
public GameObject[] layers;
```

Ove varijable referencirati će se na brzinu layera pozadine sa slikama i njihovu brzinu. [3]



Slika 4.31 Svi elementi teksture u Inspectoru, kreirala autorica rada

U Start funkciju potrebno je dodati:

```
baseLocation = new Vector2 (transform.position.x, transform.position.y);
```

čime varijabli dajemo zadane vrijednosti na početku igre, odnosno kada se igrač još nije počeo kretati, a kamera za njime. U tom trenutku, i samo jednom (jer se Start funkcija izvršava samo jednom, na početku igre) vrijednosti tranzicije na x i y osi su na nuli.

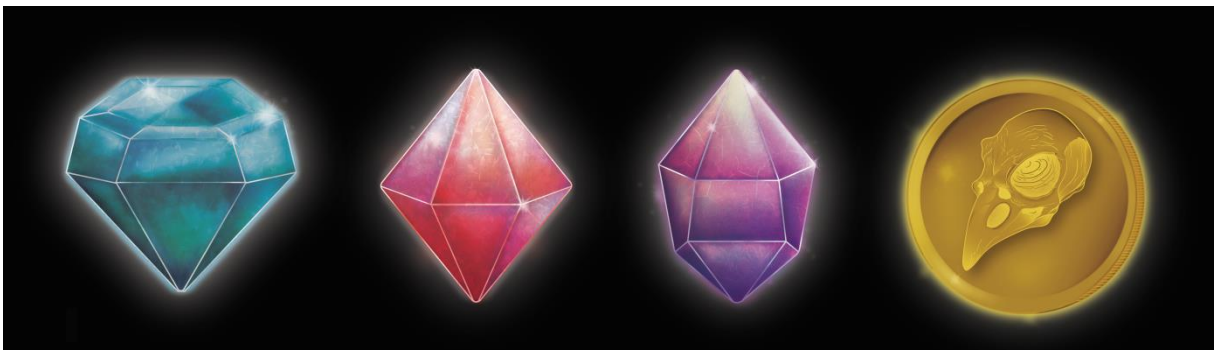
Update funkciju je potrebno nadograditi:

```
transform.position = new Vector3 (Mathf.Clamp(target.position.x,boundsLeft, b
oundsRight), Mathf.Clamp(target.position.y, boundsBottom, boundsTop), transfo
rm.position.z);
        movementFromBase = new Vector2 (transform.position.x - baseLocation.x
, transform.position.y - baseLocation.y);

        if (layers.Length > 0)
        {
            int i = 0;

            foreach (GameObject layer in layers)
            {
                var material = layer.GetComponent<Renderer> ().material;
                material.SetTextureOffset ("_MainTex", new Vector2((movementF
romBase.x * 0.015f) * layerSpeeds[i], 0.0f));
                i++;
```

Ovime svakom layeru vezanim za kameru, čija je dužina veća od nule, a u Unityju je objekt tipa „material“, dodjeljujemo ofsetni pomak bazne pozicije pomnožene za decimalnu vrijednost 0.015, što u konačnici govori „Rendereru“ kojom brzinom će renderirati slike. Takva decimalna vrijednost će zapravo usporiti kretanje oblaka, oni će se i dalje pomicati s kamerom, no uz odgodu. [3]

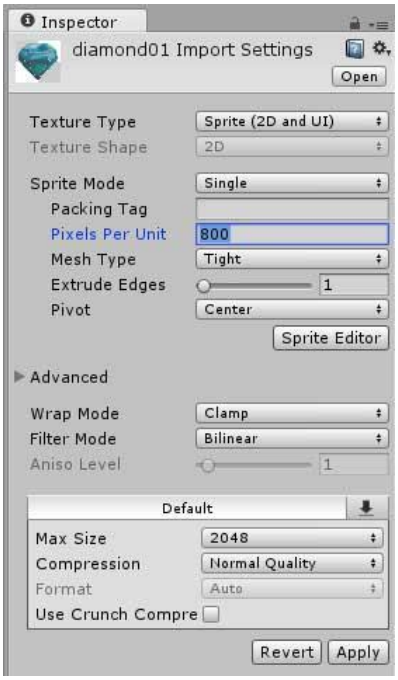
4.6. Kreiranje i implementacija objekata za sakupljanje

Kako je to uobičajeno u side scrolling igrama, igrač često sakuplja neke predmete, a u ovoj igri sakupljat će novčiće i dijamante.



Slika 4.32 Izgled svih objekata namijenjenih za sakupljanje u igri, kreirala autorica rada

Ovako izgledaju svi objekti koje će igrač sakupljati. Svaki je zasebno spremljen u .png formatu sa transparentnom pozadinom te zatim odvučen u mapu „Collectibles“. Isti su odvučeni na scenu, te im je bilo potrebno smanjiti vrijednost „Pixels per unit“, budući da su nacrtani u visokoj rezoluciji pa su zazimali pre veliku površinu na sceni. Zadana vrijednost je 100, a povećanjem te vrijednosti dijamant se smanjuje.



Slika 4.33 Pixels Per Unit opcija u Inspectoru za smanjivanje grafičkog elementa, kreirala autorica rada

U ovom slučaju želimo dodati funkciju da, kada igrač dotakne jedan od dijamanata ili novčić, on nestane sa scene te on bude „pokupljen“. Novčiću se stoga dodaje nova komponenta u Inspectoru, Add Component > Physics2D > Circle Collider. Radius novog kružnog collidera prilagođava se veličini novčića, u ovom slučaju ga je trebalo smanjiti kako glow efekt ne bi bio zahvaćen. Budući da želimo da se nešto dogodi u trenutku interakcje igrača i novčića, označavamo opciju „Is Trigger“ u Inspectoru. Kod novčića RigidBody2D ne treba dodavati jer ne želimo da padnu na tlo ili se miču na bilo koji način. Sljedeće što treba dodati je naravno nova skripta „CollectCoin“.

U ovoj funkciji nije potrebna Start niti Update funkcija tako da se obje brišu te se dodaje void funkcija koja se referira na trigger označen na Collideru novčića u sceni.

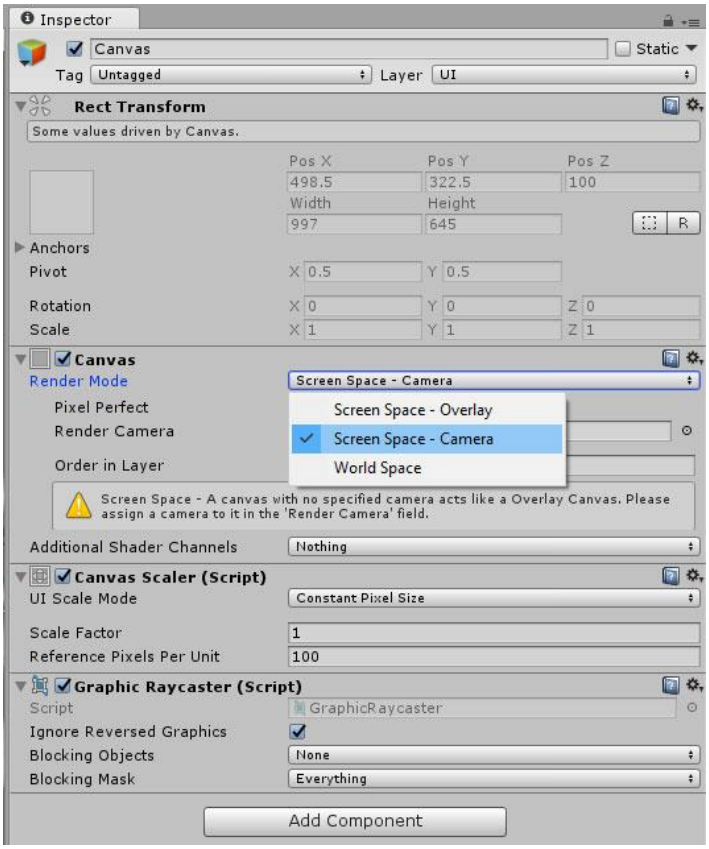
```
   public class CollectCoin : MonoBehaviour {

    void OnTriggerEnter2D(Collider2D target)
    {
        if (target.gameObject.tag == "Player")
        {
            Destroy (gameObject);
        }
    }
}
```

Funkciji void dodan je objekt tipa Collider2D naziva target, a ako je taj Tarter komponenta „GameObject“ koji je tagiran oznakom „Player“ tek u tom slučaju poziva se naredba „Destroy“, za taj isti „GameObject“. Dakle samo collider box koji ima definitan tag „Player“ će biti triger za ovu if tvrdnju.

U ovom trenutku objekti samo nestaju sa scene u trenutku kolizije s playerom, no cilj je sakupljene novčiće i dijamante negdje zapisivati, tako da ta faza slijedi nakon implementacije korisničkog sučelja u obliku brojača. [5]

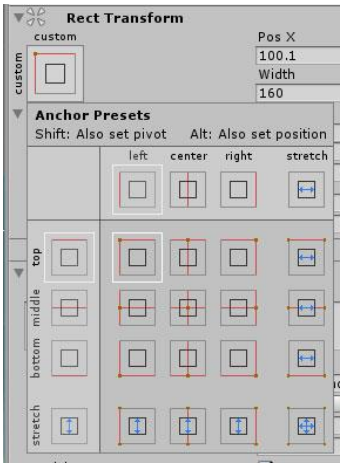
4.7. Dizajn brojača

Ključan dio igre je sustav menija i korisničko sučelje, jer bez toga bi bilo nemoguće pokrenuti igru. Grafičko sučelje dodati će se kameri koje će sučelju biti parent objekt. Dodaje se kao Game Object > UI > Canvas. Novo platno odvlači se u objekt Follow Camera kako bi postao dijete kamere. U Inspectoru Canvas komponenta ima nekoliko svojstva, no ključno je da svaki puta kada dodajemo UI komponentu, to radimo na objektu Canvas. Pod „Render Mode“ potrebno je označiti opciju „Screen Space - Camera“, kako bi platno zauzimalo istu veličinu i omjer kao i vidno polje kamere kao zaslonsko područje s UI elementima, pa je potrebno još kameru „FollowCamera“ iz hijerarhije odvući i pustiti u polje Render Camera u Inspectoru.



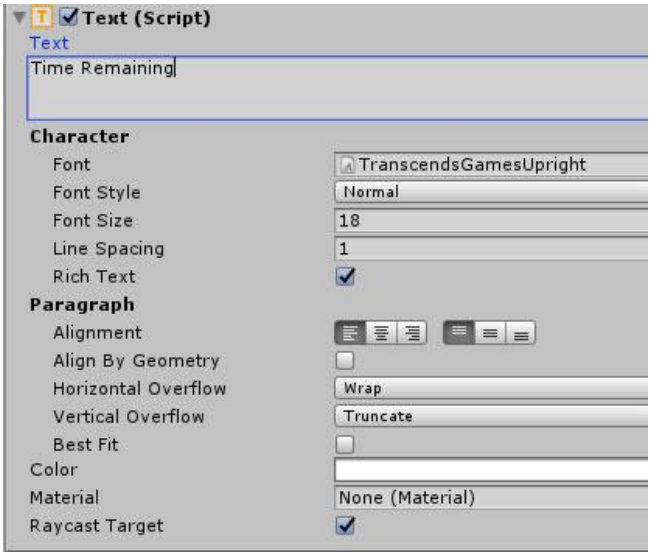
Slika 4.34 Screen Space – Camera Render Mode, kreirala autorica rada

Ova igra će imati vremenski brojač u obliku tekstualne oznake na zaslonu koja će reprezentirati preostalo vrijeme da se dostigne kraj levela. Dodaje se tekst Game Object > UI > Text, i taj je objekt automatski dodan na prethodno napravljen Canvas. Pod polje „Character“ može se odvući željeni font (prethodno instaliran na računalo te dodan pod „Assets“), jer je zadani i jedini font u Unityju Arial. Kućicu s tekstom potrebno je odvući u lijevi gornji ugao platna (W). Za tekst je u Inspectoru odabran lijevo gornje područje, tako da se teks pojavljuje u tom području bez obzira na veličinu zaslona. Drugim riječima, kretanje ovog teksta uvijek će biti relativno s obzirom na gornji lijevi ugao ekrana.



Slika 4.35 Opcije za poravnavanje teksta u tekstualnoj kućici, kreirala autorica rada

Tekstualnom objektu je zatim boja promijenjena u bijelu, malo povećan font te promijenjen sam tekst koji će biti ispisan.



Slika 4.36 Postavke za uređenje teksta tekstualnog UI elementa u Inspectoru

Sljedeće što je potrebno je dodati prazan Game Object imena „Timer“ te mu dodijeliti C# skriptu koja će omogućiti odbrojavanje. Ova skripta bit će u interakciji s UI elementima, pa je u imenski prostor skripte potrebno dodati framework i deklarirati neke javne varijable:

```
using UnityEngine.UI;
    public class Timer : MonoBehaviour {
    public float valueRemaining = 10;
    public float valueMaximum = 120;
    public float valueConsumptionRate = 1f;
```

Ove vrijednosti će se poslije promijeniti kada se igra bude testirala kako brojač ne bi odbrojavao pre malo ili previše vremena. Prva varijabla ima zadanu vrijednost 10 koja će biti promjenjiva, druga pokazuje maksimalan limit vremena što je 120 sekundi, a varijabla „ValueConsumptionRate“ označava da se odbrojavanje vrši jednom u sekundi.

Sljedeće je potrebno napraviti vezu između tekstualnog objekta i onoga što ćemo kasnije definirati u Update funkciji:

```
public Text timerText;
public bool gameInProgress = true;
```

Druga varijabla može poslužiti u slučaju da se timer pauzira ako se i igra pauzira, ili da se pauzira ako se bilo koji željeni događaj ispunjava. Iz skripte je uklonjena Start funkcija jer nije potrebna, a u Update se dodaje tvrdnja:

```
      void Update () {
       if (gameInProgress)
       {
           valueRemaining -= Time.delatTime * valueConsumptionRate;
           timerText.text = "Time Remaining: " + Mathf.Round(valueRemaining.
ToString ();
       }
```

Dakle ako je igra pokrenuta, od vrijednosti vremena koje je preostalo oduzima se jedna vremenska jedinica pomnožena sa „ValueConsumptionRate“, odnosno jednom sekundom. U isto vrijeme, „deltaTime“ se referencira na vrijeme u sekundama koje je potrebno da se završi zadnji frame i služi samo za čitanje (Read Only). Drugim riječima, deltaTime popravlja kašnjenje frameova, pa se Update funkcija izvršava svaki frame bez obzira na snagu računala jer zbog toga frame rate ne mora uvijek biti konzistentno, pa tako deltaTime sprječava kašnjenje ili pre brzo izvršavanje radnje po frame-u. Vrijednost „deltaTime“ je pomnožena s varijablom „valueConsumptionRate“. Kada bi željeli da se odbrojavanje vrši, primejrice, svake dvije sekunde, tu bi varijablu postavili umjesto na 1f, na vrijednost 0.5.

Drugi uvjet referencira se na sam ispis teksta na platnu te osigurava da se tekstualnoj komponenti „Time Remaining: “ dodaje funkcija Mathf.Round kojoj se dodjeljuje vrijednost preostalog vremena. [6]

Mathf funkciji dodan je Round zato što se Round odnosi na prave sekunde. Kada ne bi bilo te ključne riječi, sve ove vrijednosti bi se odnosile na milisekunde umjesto ne sekunde. Posljednji sufiks „ToString“ obavlja konverziju funkcije u niz. To je potrebno napraviti zato što je String u objektno orijentiranom programiranju objekt koji služi samo za čitanje te ima tekstualne znakove,

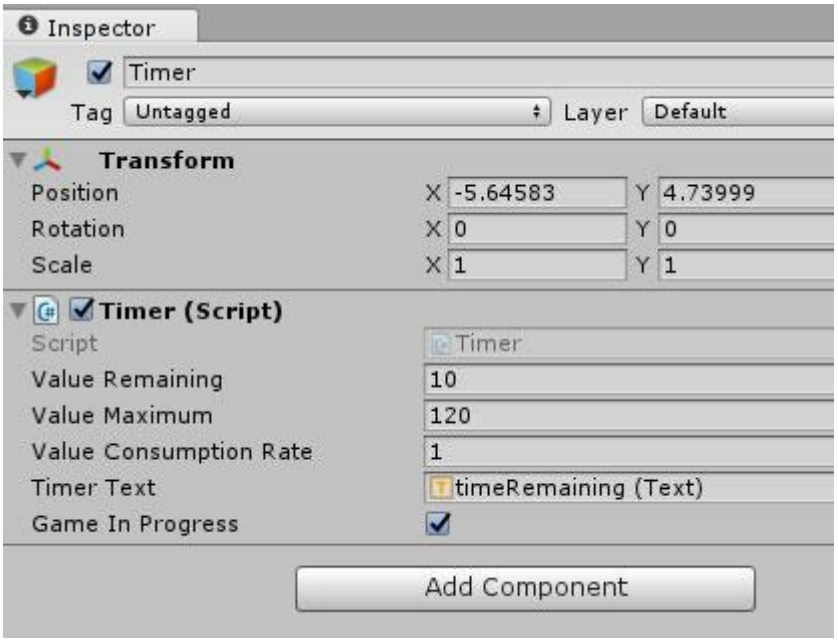
odnosno služi se skupom „Char“. Budući da smo se koristili matematikom kako bi izračunali preostalu vrijednost, ta se vrijednost može pretvoriti u tekst jedino putem stringa, a samim time i ispisivati na zaslonu. Potrebno je ugnijezditi još jednu if tvrdnju unutar postojeće.

```
if (valueRemaining < 0)
{
    gameInProgress = false;
       endGameWithFailure ();
 }
```

Kada vrijeme istekne, bool varijabla „gameInProgress“ postane „false“, odnosno igra se gubi. Ispod void funkcije dodana je još jedna, za sada prazna funkcija koja označava kraj igre.

```
    void endGameWithFailure(){

}
```

Kako bi se skripta mogla izvršavati, potrebno je još u Unityju praznom objektu Timer na sekciju Timer Text iz hijerarhije odvući timeRemaining UI element. [6]



Slika 4.37 Promjenjive vrijednost varijabli definiranih u skripti „Timer“ u Inspectoru

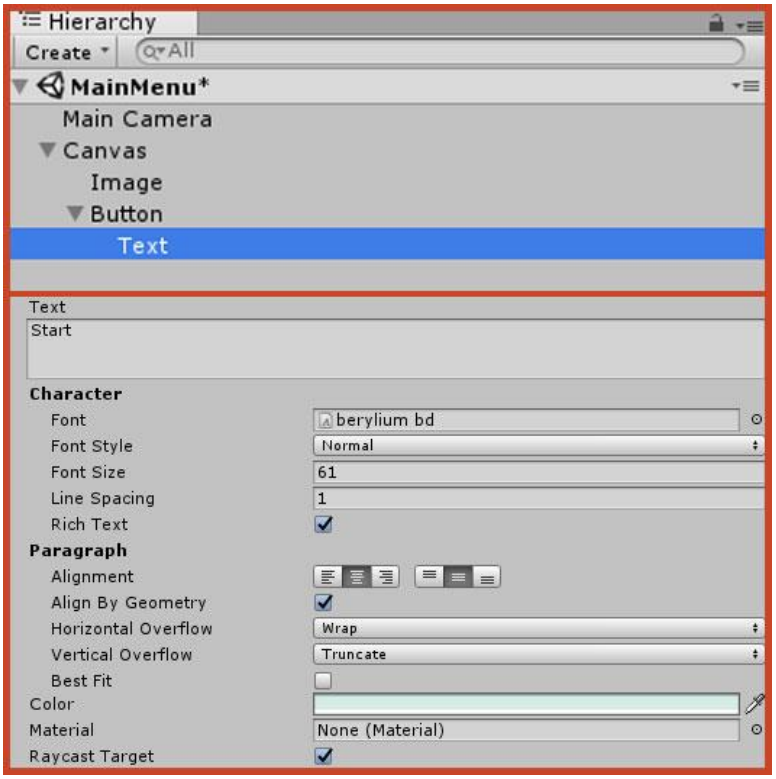
4.8. Kreiranje glavnog menija

Glavni meni s korisničkim sučeljem nalazit će se na zasebnoj sceni, tako da je u Assets mapu najprije potrebno dodati novu scenu pod desnim klikom > Create > Scene. Sceni je dodijeljen

naziv „MainMenu“ te dodan objekt „Canvas“ kojem je Render Mode opet odabran kao „Screen Space – Camera“ a u „Render Camera“ dodaje se glavna kamera kao i prije.

Sljedeće što nam treba je pozadinska slika pa se dodaje Game Object > UI > Image, te se novom objektu iz „assets“ mape odvlači željena pozadinska slika u Inspector pod sekciju „Source Image“ te se prilagođava njena veličina tako da malo izlazi iz vidnog polja kamere.

Glavni meni mora imati gumbe tako da dodajemo novi objekt namijenjen za tu svrhu, Game Object > UI > Button. Umjesto običnog pravokutnika u nekoj boji, u Inspectoru je dodan crtež kamena u sekciji „Source Image“. Komponenta „Button“ dolazi s tekstom, i zato se u hijerarhiji unutar gumba nalazi i tekst. S označenim tekstom, u Inspector odvlačimo željeni font, prilagođavamo veličinu i boju fonta te pod „Text“ upisujemo riječ „Start“.



Slika 4.38 Opcije tekstualnog UI elementa u Inspectoru, kreirala autorica rada

Isti gumbić je dva puta dupliciram te su tekstovi promijenjeni i konačna scena menija izgleda ovako:

Slika 4.39 Izgled glavnog menija s 3 UI elementa, kreirala autorica rada

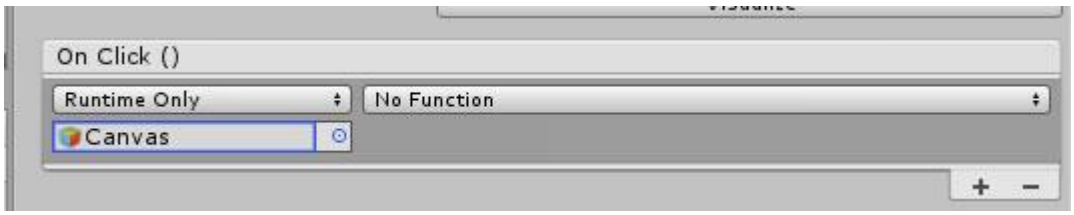
Kada pritisnemo neki od gumbića, želimo da se nešto događa, pa je sljedeći korak dodavanje skripte „Select“. U ovoj skripti neće biti potrebne zadane Start i Update funkcije pa se odmah na početku brišu. U imenski prostor skripte dodaje se klasa koja omogućuje pristup metodama za manipulaciju scenama:

```
using UnityEngine.SceneManagement;
```

U glavnu klasu potrebno je dodati nekoliko linija za prepoznavanje levela po prefiksu ID-a:

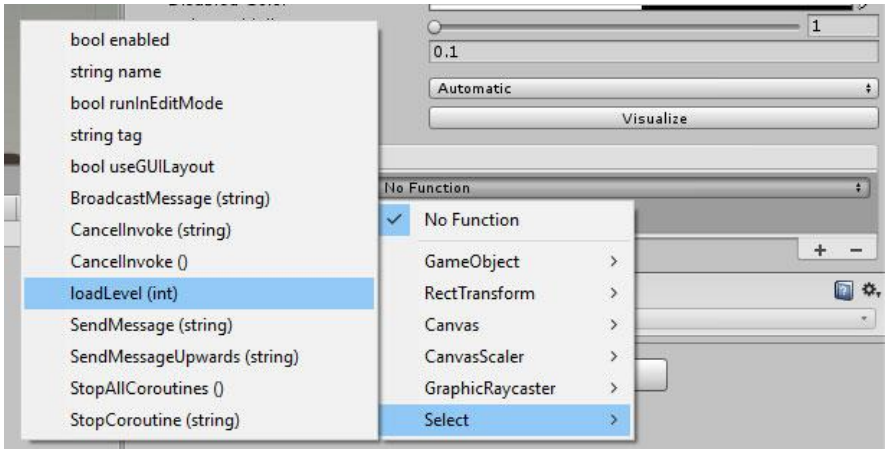
```
public string levelPrefix = "Level";

public void loadLevel (int levelID)
{
    SceneManager.LoadScene (levelPrefix + levelID);
}
```

Iz varijable public void loadLevel pristupit ćemo SceneManagement frameworku.
Sljedeće u Unityju treba označiti gumbić „Start“ u sceni „Main Menu“i u Inspectoru u sekciji „On Click ()“ dodati uvjet događaja putem plusića dolje desno. Odabire se objekt „Canvas“.



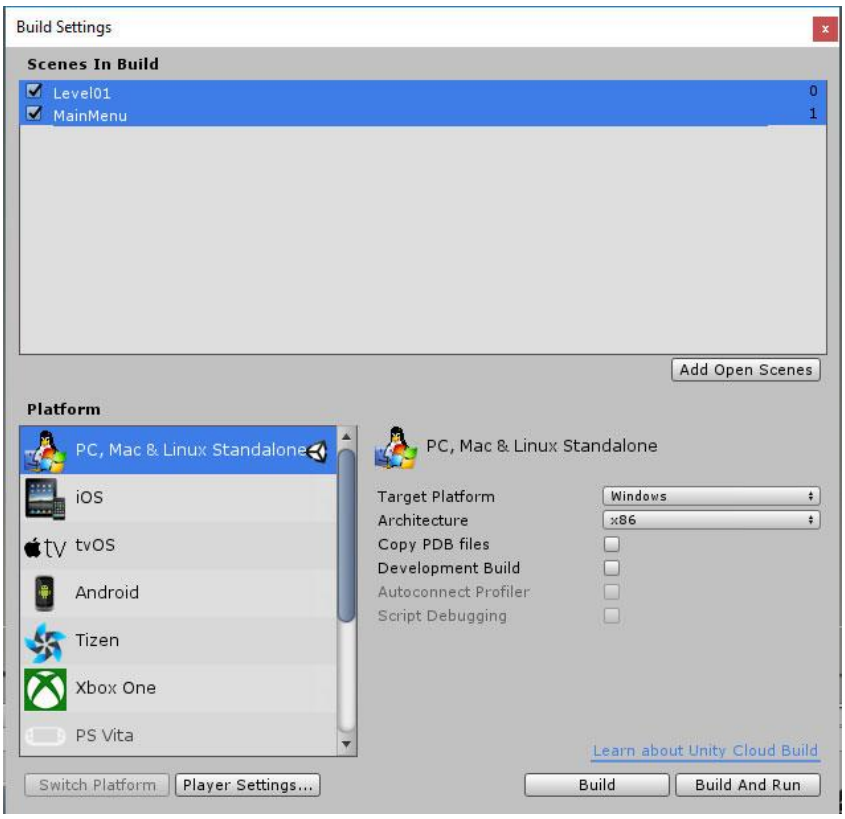
Slika 4.40 Prazna „On Click()“ funkcija UI elementa, kreirala autorica rada

Skriptu „Select“ potrebno je dodijeliti objektu Canvas, a zatim gumbu Start pod „On Click ()“ u polju gdje piše „No Function“ odabrati funkciju Select > loadLevel (int).



Slika 4.41 Dodavanje funkcije „loadLevel“ kao uvjet promjene scene na pritisak Start gumbića, kreirala autorica rada

Zatim, kako bi gumb funkcionirao potrebno je podesiti „Build Settings“. Odabire se File > Build Settings. Otvara se prozor „Build Settings“ gdje je potrebno u „Scene In Build“ području odvući željene scene te se prozor na iksiću zatvara i gumb Start funkcionira. [6]



Slika 4.42 "Build Settings" prozor s učitanim scenama, kreirala autorica rada

4.9. Kreiranje „game manager" objekta

Sve do sada napravljeno je prilično jednostavno. Međutim, kada u igri postoje mnogo različitih događaja i ishoda, potreban nam je način da sve to nekako spojimo u cjelinu. To posebno vrijedi za funkcije kao što su spremanje informacija i zadržavanja dosljednosti za iste. Game Manager je ono što nam je potrebno, a toe nešto što upravlja svim tim događajima. To je nešto kao konačni automat, odnosno sustav koji se sastoji od konačnog broja stanja, prijelaza između tih stanja te akcija koje obavlja. Zato je potrebno stvoriti novi Game Object te mu dodati C# skriptu koja će imati naziv „PersistentManager". Prvo se dodaju neke klase koje će biti potrebne u imenski prostor.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using UnityEngine.SceneManagement;
```

```
using System.Runtime.Serialization.Formatters.Binary;
```

klasa služi za serijalizaciju, odnosno za kriptiranje i dekriptiranje a:

```
using System.IO;
```

je uobičajen framework koji koristi input i output sistem, odnosno služi za spremanje i sakupljanje datoteka. SceneManagement obješnjen je prethodno.

PersistantManager je objekt kod kojeg će sve pripadajuće funkcionalnosti ostajati aktivne tako dugo dok je i sama igra aktivna. Na primjer, ako i kada prelazimo iz levela u level, ili iz scene u scenu, ovaj Game Object neće biti uništen ili resetiran u toj tranziciji. Što je također bitno jest inicijalizacija novih objekata u igri kad ih želimo uništiti i zadržati podatke o tome u game manageru. Sve to postići ćemo tako da koristimo singleton shemom. Singleton shema ili uzorak je matematički koncept čiji se principi koriste u softverskom inženjerstvu a bazira se na ograničavanju instanciranja klase na jedan objekt. Koristi se kada je potreban točno jedan objekt za koordiniranje radnji u sustavu. To je jednostavan način za uspostavljanje game objekata. Počinjemo s dodavanjem varijable:

```
public static PersistentManager dataStore;
```

Ova varijabla reprezentira sebe samu te je naziva dataStore.

Kako bi se spriječilo da se funkcija poziva, želimo djelovati sa „Awake“ funkcijom, i zato se briše funkcija Start. Awake funkcija, za razliku od Start funkcije djeluje prije nego se igra pokrene te se poziva samo jednom i traje cijelo vrijeme.

```
void Awake()
  {
      if (dataStore == null) {
          DontDestroyOnLoad (gameObject);
          dataStore = this;
      } else if (dataStore = !this) {
          Destroy (gameObject);
      }
  }
```

U ovoj funkciji provjeravamo je li dataStore objekt prisutan. Ako ga nema (ako je jednak nuli), označava se sa DontDestroyOnLoad, te će se trenutni gameObject propustiti i postaviti vrijednost game objekta na „this“. Inače, ako već postoji dataStore i nije jednak tom objektu, objekt će biti uništen. To će biti slučaj kada imamo tranziciju u novu scenu, jer ova funkcija pokušava pogoditi Awake metodu, ali tamo je već jedna koja nije jednaka trenutnom stanju u sataStore varijabli. Ovdje se osigurala dosljednost određenog game objekta. Znajući da će objekt preživjeti prelaskom u drugu scenu, moramo odabrati koje stvari želimo zadržati.

Radimo public instance varijablu koja će pokrivati bilo što za što želimo da klasa zadržava i ispod dataStore varijable dodaje se:

```
    // Public instance variables
public int currentLevelID;

//Persisted variables
public int coinsCollected;
public int highestLevelCompleted;
```

U prvoj varijabli postavlja se pitanje želimo li zadržati trenutni ID levela kao integer, a on je public kako bi ostale klase imale pristup podacima koje predstavlja, a podaci koje će predstavljati su broj sakupljenih novčića (coinsCollect varijabla) i najveći level koji je završen. Zatim iz skripte CoinCollect možemo pristupiti Game Manageru te reći da smo pokupili novčić pa u void funkciju iznad if tvrdnje dodajemo:

```
PersistentManager.dataStore.coinsCollected += 1;
```

Ovom tvrdnjom dodajemo broj 1 trenutnoj vrijednosti pohranjenoj u dataStore varijablu u PersistentManager skripti te vrijednost dodjeljujemo varijabli „public int coinsCollected“. Samo kada koristimo singleton shemu možemo pristupiti varijablama na ovakav način. U svim drugim slučajevima potrebno je koristiti GetComponent metodu. Sada se zbrajaju svi pokupljeni novčići, međutim, još uvijek još treba postići spremanje i loadanje podataka kako bi se oni zadržali svaki puta kada igramo igru. [6]

4.9.1. Spremanje i učitavanje podataka

Postoji mnogo načina za spremanje i učitavanje podataka, a svaki od njih ima različitu jačinu s obzirom na to kako ćemo ih koristiti i koliko podataka za skladištenje postoji. Najuobičajeniji način, ali ne ujedno i najlakši bit će ovdje demonstriran. Najbolje je koristiti ovaj način jer podržava enkripciju pa se rezultat neće moći hakirati. U PersistantManager skripti dodaje se nova klasu koja će pohranjivati dva integera, odnosno cijela broja. Jedna od njih predstavlja ukupan broj sakupljenih novčića, a drugi je najviši level postignut.

```
   class GameData
{
    public int coinsCollectedTotal;
    public int highestLevel;
}
```

Slijede dvije public funkcije, Save i Load. U oba slučaja, funkcije će biti napravljene u tri dijela – dio gdje otvaramo datoteku (file), dio gdje spremamo ili učitavamo bilo koje podatke te dio gdje zatvaramo datoteku.

```
    //Saving
public void Save {

BinaryFormatter bt = new BinaryFormatter ();
FileStream file = File.Create (Application.persistentDataPath + "/GameDat
a.dat");

GameData data = new GameData ();
data.coinsCollectedTotal = coinsCollected;
data.highestLevel = highestLevelCompleted;

bf.Serialize (file, data);
file.Close ();
}
```

Koristimo varijablu BinaryFormatter imena „bf“ te je skladištimo u novu funkciju istog imena. Sljedeća linija služi kao pristup i spremanje datoteci i zato se koristi FileStream varijabla imena „file“ u koju se sprema funkcija File.Create koja koristi sve navedeno u zagradi što predstavlja datotečni put u određeni direktorij. Application.persistentDataPath zapravo pronalazi put za pohranu na bilo kojem tipu OS-a. U slučaju da se ne koristi, manualno je potrebno programirati gdje želimo da se podaci spremaju. + „/GameData.dat“ je samo ime koje je dodijeljeno. U drugom djelu funkcije nalazi se varijabla GameData u koju je pohranjena istoimena funkcija, a bazira se na prethodno deklariranoj klasi. Oba svojstva klase (coinsCollectedTotal i highestLevel) pohranjuju se u novu funkciju. Podaci za sakupljene novčiće i dostignuti level zatim se spremaju u varijable koje reprezentiraju njihov ukupan broj. U trećem dijelu funkcije brojevi iz binarnog formata koji iz varijabli „file“ i „ data“pretvaraju se u format čitljiv za Unity. Zato je potrebna riječ „Serialize“. Na kraju funkcije, datoteka se zatvara. Funkcija „Load“ izgleda ovako:

```
    //Loading

    public void Load {
     if (File.Exists (Application.persistentDataPath + "/GameData.dat")) {
         BinaryFormatter bt = new BinaryFormatter ();
         FileStream file = File.Open (Application.persistentDataPath + "/G
ameData.dat", FileMode.Open);
         GameData data = (GameData)bf.Deserialize (file);
         file.Close ();

         coinsCollected = data.gemsCollectedTotal;
         highestLevelCompleted = data.highestLevel;
         }
     }
```

Load funkcija funkcionira na sličan način, no najprije provjerava da li datoteka „GameData.dat“ postoji na putu u direktorij. Opet se koristi ugrađeni konstruktori BinaryFormatter i FileStream koji služe da serijalizaciju i deserijalizaciju, odnosno konverziju u format čitljiv Unityju te ponovnu konverziju u prvobitni, binarni oblik. Ove dvije metode gotovo uvijek se koriste za spremanje i učitavanje podataka. Ako datoteka postoji (.Open), otvara se datoteka u „open mode“. Stvara se novi objekt GameData imena data te se u njega pohranjuje deserijalizirana vrijednost datoteke i onda zatvara.

Nakon što su napravljene Save i Load funkcije, u Awake funkciju potrebno je dodati unutar if tvrdnje Load ();. Dakle ako je taj game objekt aktivan, , učitat će pohranjene podatke. [6]

4.10. Kreiranje postavki događaja za pobjedu i poraz

Ono što slijedi je odlučiti što će se dogoditi ako dobijemo ili izgubimo igru. Generalno, igra se gubi ako padnemo s ruba i umremo, ako vrijeme istekne ili smo ubijeni od strane neprijatelja ili nečeg drugog u igri što uzrokuje štetu igraču. Igra se dobiva ako stignemo do konačnog cilja na kraju levela.

U ovu svrhu u skriptu Persistent Manager dodat će se još dvije funkcije ispod „Awake“ funkcije, jedna za pobjedu a druga za poraz. Obje funkcije će biti tipa „public“ i neće ništa vraćati - „void“: Dodaje se if tvrdnja koja provjerava ID trenutnog levela te se poziva na cjelobrojne varijable ranije deklarirane u skripti. Zatim se taj podatak sprema, jer kada level završi s gubitkom, ne želimo ništa spremati. Ispod se dodaje Debug.Log komanda koja prima tekstualnu poruku za pobjedu igre, a ispod je ista komanda koja dodaje ime scene poruci tako da se zna koji level je završio ishodom pobjede (ta poruka pojavljuje se samo u konzoli). Zatim se poziva SceneManager koji otvara glavni meni. Ova funkcija dakle prati najveći level dostignut, a u ovom slučaju neće biti potrebno no funkcionalnost je stavljena kako bi se kasnije igra nadogradila s više levela, a uobičajeno je da su progresivni leveli zaključani dok se prethodni ne prijeđu. Na primjer, uobičajeno bi bilo dodati tvrdnju koja provjerava ID levela, primjerice nekog gumba + 1) jer želimo otključati level koji dolazi kasnije), pa se istome može postaviti jednostavna void funkcija s boolean varijablom za otključavanje levela. Funkcija za gubitak je ista samo što ne sadržava if tvrdnju te ima drugačiju poruku.

```
public void endGameWin(){
 if (currentLevelID < highestLevelCompleted) {
     highestLevelCompleted = currentLevelID;
 }
 Save();
```

```
        Debug.Log ("Game over - Win");
        Debug.Log (SceneManager.GetActiveScene ().name );
        SceneManager.LoadScene ("MainMenu");
    }

    public void endGameLoss(){
        Debug.Log ("Game over - Loss");
        Debug.Log (SceneManager.GetActiveScene ().name );
        SceneManager.LoadScene ("MainMenu");
    }
```

4.11. Dovršavanje brojčanika

Kada se pisala skripta „Timer“ na samom dnu napisana je funkcija endGameWithFailure() koju je tek sada, kada imamo mogućnost za pobjedu i poraz moguće implementirati u igri. Najprije treba pozvati PersistantManager skriptu i podatke koje sprema u funkciju endGameLoss(). Sada, kada vrijeme istekne, igra prestaje i scena se mijenja u glavni meni. U nastavku slijedi cijela glavna klasa skripte „Timer“:

```
public class Timer : MonoBehaviour {

    public static Timer timer;

    public float valueRemaining;
    public float valueMaximum = 50;
    public float valueConsumptionRate = 1f;
    public bool isFirst = true;

    public Text timerText;
    public bool gameInProgress = true;


    void Awake()
    {
        if (timer == null) {
            timer = this;

        } else if (timer != this) {
        }
    }

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

        if (gameInProgress)
        {

            if (this.isFirst) {
                valueRemaining = 10;
                isFirst = false;
            }
```

```
            valueRemaining -= Time.deltaTime * valueConsumptionRate;
            timerText.text = "Time Remaining: " + Mathf.Round(valueRemaining)
.ToString ();

            if (valueRemaining < 0) {
                PersistentManager.dataStore.coinsCollected = 0;
                PersistentManager.dataStore.diamondsCollected = 0;
                gameInProgress = false;
                endGameWithFailure ();
            }

        }

    }

    public void endGameWithFailure(){
        PersistentManager.dataStore.endGameLoss();
    }
}
```

Ono što je novo je funkcija „Awake“ s else if tvrdnjom, koja provjerava na samom početku igre postoji li varijabla „timer“, te ako postoji, u njega se sprema trenutačna (this) instanca klase. Kako bi se spriječilo da se odbrojavanje nastavi u slučaju gubitka igre na bilo koji drugi način osim isteka vremena, u Update funkciji dodana je tvrdnja koja provjerava je li igra pokrenuta (gameInProgress), te ako je, vrijednost preostalog vremena se resetira na predviđenu vrijednost za to, a to je 10, bilo da je igra pokrenuta prvi puta ili igramo ispočetka u slučaju gubitka. [6]

4.12. Kreiranje brojača za sakupljene objekte

Osim što svaki sakupljeni dijamant dodaje vremensku jedinicu od 10, a novčić od 5 na preostalo vrijeme, ideja je također bila kreirati brojčanik koji zbraja količinu sakupljenih dijamanata i novčića. Za svaki od njih, napravljena su nova dva UI tekstualna objekta s natpisima „Coins: “ i „Diamonds: “. Za svaki objekt, postoje po dvije skripte. Za novčić su to „CollectCoin“ i „CollectCoinHandler“ skripte. Ovako izgleda čitava „CollectCoin“ skripta:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;

public class CollectCoin : MonoBehaviour {


    void OnTriggerEnter2D(Collider2D target)
    {
        PersistentManager.dataStore.coinsCollected += 1;
```

```
        if (Timer.timer.valueRemaining + 5 > Timer.timer.valueMaximum) {
            Timer.timer.valueRemaining = 50;
        } else {
            Timer.timer.valueRemaining+=5;
        }



        if (target.gameObject.tag == "Player")
        {
            Destroy (gameObject);
        }
    }
}
```

Potonji dio gdje se objekt uništava već je objašnjen. Također se može vidjeti logika iza dodavanja vremena za skupljeni objekt, što je riješeno preko „Timer“ game objekta u igri te instanci „timer“ i „valueRemaining“, te jednostavnih if else petlji.

„CollectCoinHandler“ klasa služi za povezivanje sakupljenoga te ispisivanje u UI objektu u sceni. U funkciji Awake provjerava se trenutno stanje sakupljenih novčića, dok se u Update funkciji podaci iz PersistantManagera povlače svakog framea, provjerava se sakupljeno te se pretvara u string, odnosno brojke se pretvaraju u tekst i ispisuju u tekstualnom UI objektu u sceni i to samo ako je bool dunkcija gameInProgress zadovoljena. U suprotnom igra završava.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
namespace AssemblyCSharp


{
    public class CollectCoinHandler : MonoBehaviour
    {
        public static CollectCoinHandler collectCoinHandler;
        public Text coinText;
        public bool gameInProgress = true;

        void Awake()
        {
            if (collectCoinHandler == null) {
                collectCoinHandler = this;

            } else if (collectCoinHandler != this) {
            }
        }

        // Update is called once per frame
```

```
        public void Update () {

            if (gameInProgress)
            {
                coinText.text = PersistentManager.dataStore.coinsCollected.To
String ();
            }

        }

        public void endGameWithFailure(){
            PersistentManager.dataStore.endGameLoss();
        }
    }
}
```

Inače, u PersistantManager skripti, deklarirane su dvije varijable, jedna za sakupljene novčića, druga za dijamante. Dodane su dvije funkcije, „endGameWin“ i „endGameLoss“ u kojima je tim varijablama dodijeljena vrijednost nula, kako bi se svakim novim pokretanjem levela one resetirale. U Save i Load funkcijama ključni dijelovi su:

```
    FileStream file = File.Create (Application.persistentDataPath + "/GameDat
a.dat");

    GameData data = new GameData();

    data.coinsCollectedTotal = coinsCollected;
```

Funkcija traži put pohrane gdje će pohraniti podatke, te ukupnom zbroju sakupljenih novčića dodjeljuje vrijednost koja se sprema u varijablu „coinsCollected“, a ta je vrijednost uvijek inkrementirana za jedan svaki puta kada dođe do kolizije playera i objekta.

U Load funkciji pozivaju se iste varijable, samo što je se u njoj ne stvara, već otvara postojeći put pohrane podataka te se oni deserijaliziraju:

```
if (File.Exists(Application.persistentDataPath + "/GameData.dat")) {
            BinaryFormatter bf = new BinaryFormatter();
            FileStream file = File.Open (Application.persistentDataPath + "/G
ameData.dat", FileMode.Open);
            GameData data = (GameData)bf.Deserialize (file);
            file.Close ();
```

Varijable se nalaze u klasi GameData na dnu skripte:

```
[Serializable]
class GameData
```

```
{
    public int coinsCollectedTotal;
    public int diamondsCollectedTotal;
    public int highestLevel;
}
```

[6]

4.13. Kreiranje kraja levela i mrtve zone

U ovom trenutku, player može sakupljati coinove, kretati se, skakati, postoji glavni meni a vrijeme se odbrojava, no još ne postoji kraj igre, a ako padnemo s ruba platforme, igra zapravo ne završi te se level ne resetira a igra traje tako dugo dok vrijeme ne istekne, što nije najelegantniji način da se igra dovrši, pa je potrebno napraviti mrtvu zonu tako da, kada player ode sa scene, igra završava s gubitkom. Na kraju levela također će se nalaziti drvo, pa kada player dođe do te točke, level je dovršen. Stvaramo zato novu skriptu „EndGameAtTouch“. U skripti je potrebno ukloniti Start i Update funkcije te stvoriti javnu bool varijablu, koja će stoga biti da ili ne pitanje.

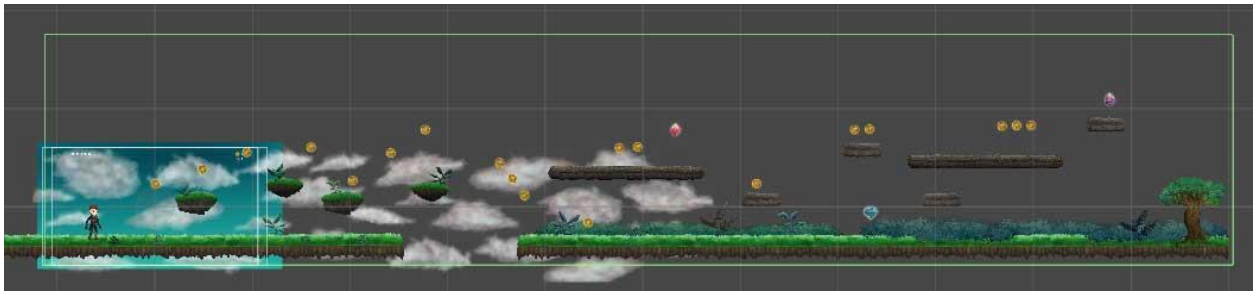
```
public bool endWithWin;
```

Kada dođemo do drva, želimo da igra završi ishodom pobjede, odnosno da varijabla poprimi vrijednost „true“, a ako padnemo s ruba levela, ishod varijable će biti „false“: Zatim se stvara nova void funkcija koja prihvaća Collider2D. Inače, „OnTriggerEnter2D“ postavlja se kada drugi objekt dotakne collider koji je tipa trigger na određenom objektu, i funkcionira samo za 2D fiziku. Cijela funkcija izgleda ovako:

```
    void OnTriggerExit2D(Collider2D target) {
     if (target.gameObject.tag == "Player") {

         if (endWithWin == true) {
             PersistentManager.dataStore.endGameWin ();
         } else {
             PersistentManager.dataStore.endGameLoss ();
     }
  }
 }
```

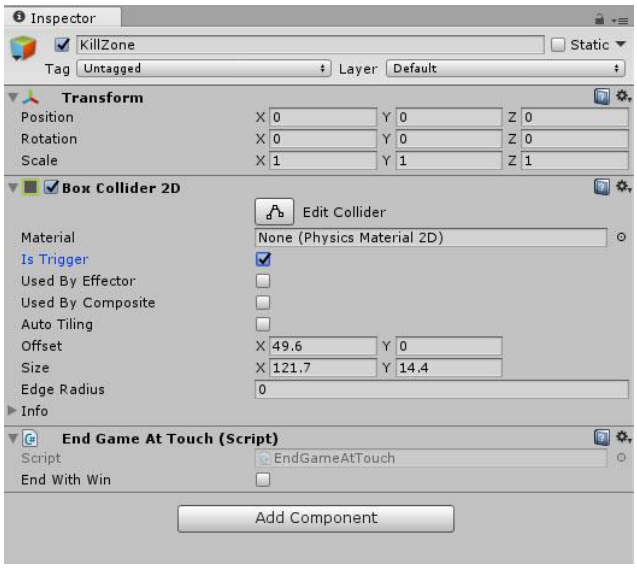
Funkcija ima if tvrdnju koja traži objekt koji je tagiran s nazivom „Player“, te ako je endWithWin zadovoljen, u funkciji iz PersistantManager skripte „endWithWin“ dodaje se vrijednost „true“, a u suprotnom, vrijednost „true“ dodaje se endGameLoss funkciji.

Sljedeće u Unityju stvaramo novi prazan objekt imena „KillZone“. Objektu se dodaje komponenta „BoxCollider2D“, te se isti pojavljuje u sceni te ga je potrebno razvući po osima kroz cijeli level.



Slika 4.43 Collider čije će granice biti trigger za ulazak u mrtvu zonu, kreirala autorica rada

Zeleni rub reprezentira granice 2D Collidera. Sve što je izvan ovih granica, smatra se mrtvom zonom a funkcionirat će samo ako player padne u provaliju između dvije prizemljene platforme. Potrebno je dodati još jednu komponentu, a to je skripta „EndGameAtTouch“. Box Collider komponenti potrebno je označiti kućicu „Is Trigger“, a u skripti se ostavlja neoznačeno svojstvo „End With Win“.

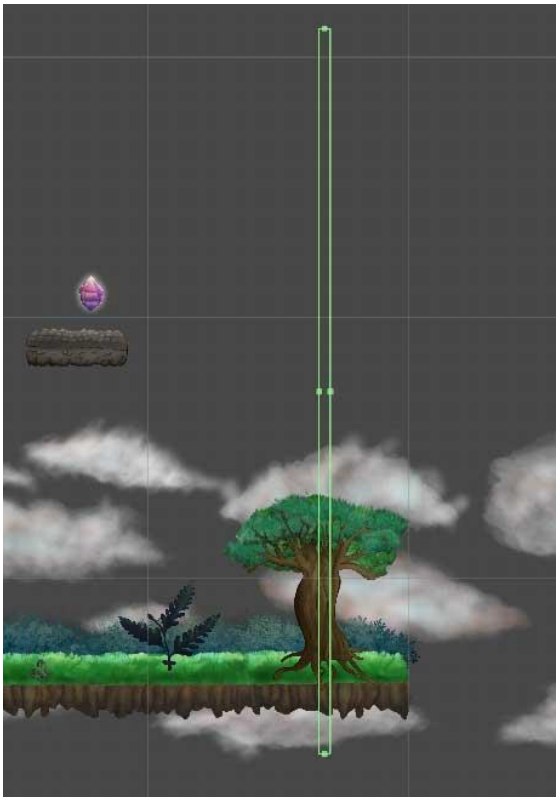


Slika 4.44 „Kill Zone“ game objekt s dodanom skriptom i komponentom „Box Collider 2D“ te označenom opcijom „Is Trigger“, kreirala autorica rada

U ovom trenutku, ako testiramo igru i skočimo u provaliju, igra završava te se scena mijenja u glavni meni.

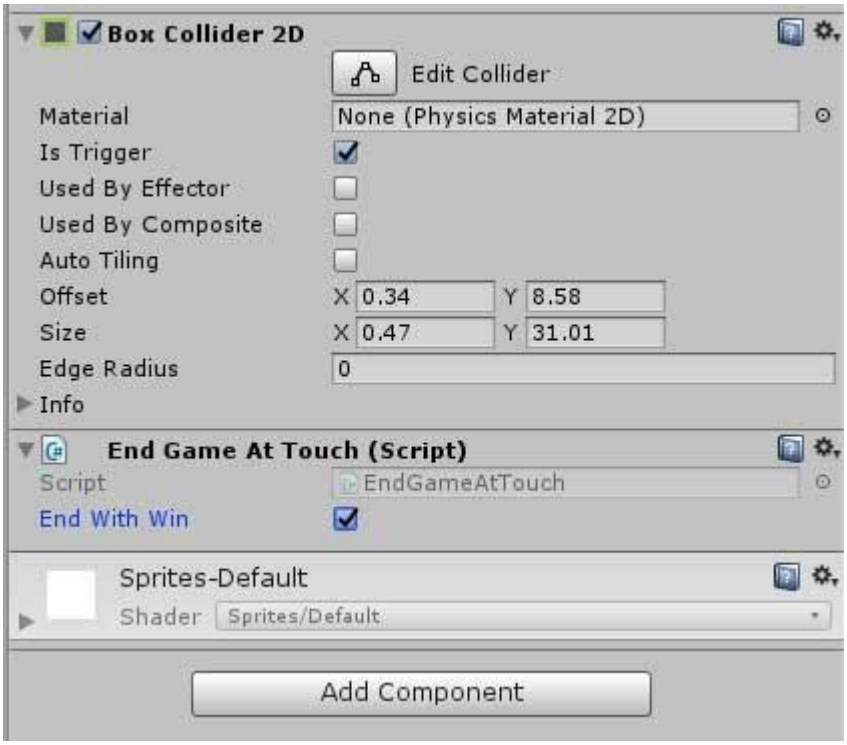
Sada je još potrebno dodati svojstvo pobjede u igri. Da bi pobijedili u igri, potrebno je doći do kraja levela te player mora ući u koliziju s objektom drva. Dakako, .png slika je najprije postavljena na željeno mjesto u sceni te joj je dodijeljen naziv „LevelEnd“ te dodan Box Collider 2D čija je veličina sužena na X osi i produljena na Y osi, u slučaju da player skoči s platforme koja je blizu

kraja pa fizički ne dodirne drvo, ili u slučaju da preskoči drvo.. Također, u Inspectoru je za Box Collider 2D komponentu označeno svojstvo „Is Trigger“, budući da će collider služiti kao trigger za neki drugi događaj, što je u ovom slučaju pobjeda levela. [5]



Slika 4.45 Collider koji triggera kraj levela, kreirala autorica rada

Na objekt „LevelEnd“ dodaje se skripta „EndGameAtTouch“ u kojoj je samo potrebno označiti mogućnost „End With Win“, prethodno definiraju u skripti,. I sada igra ima mogućnost pobjede.

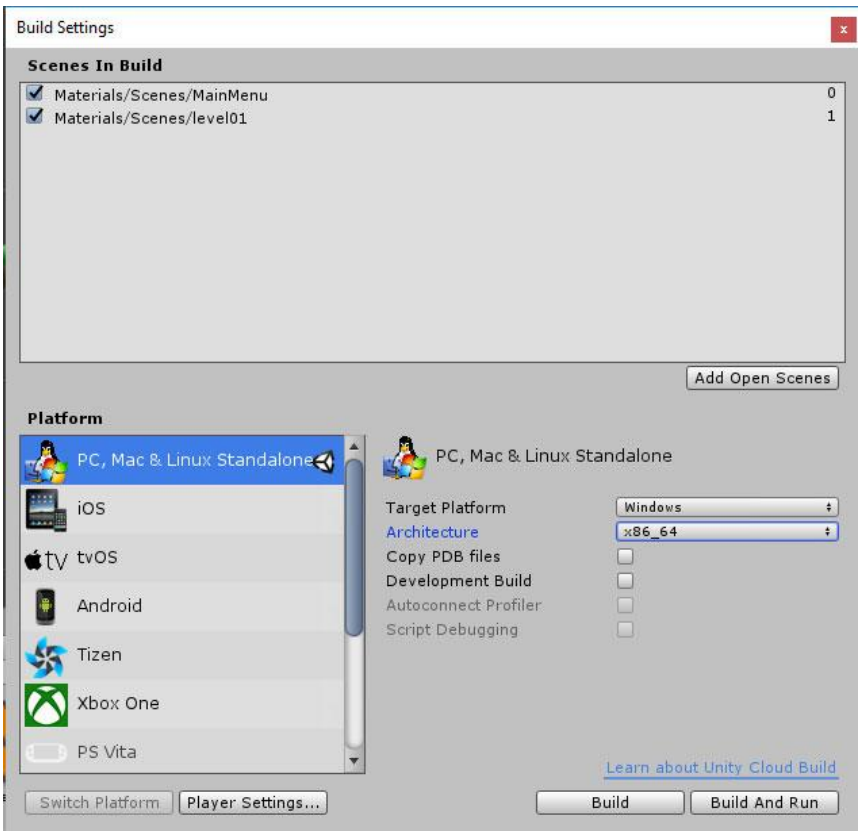


Slika 4.46 Označavanje „End With Win" funkcije, kreirala autorica rada

S mogućnosti pobjede ili poraza, ovaj jedini level u igri postaje smislen i funkcionalan i može se zaključiti kako je prva verzija prvog levela gotova, a igra je spremna za export. [5,6]

4.14. Priprema igre za distribuciju

Zadnja faza u izradi igre je export igre, bilo da je želimo negdje distribuirati na određenoj platformi ili poslati na beta testere. U izborniku File potrebno je označiti opciju „Build Settings". Otvara se prozorčić u koji moramo odvući sve scene, a redoslijed mora biti onakav kakvim želimo da se scene pojavljuju. U ovom slučaju prva scena će biti glavni meni.



Slika 4.47 Build Settings prozor s odabranom „PC, Mac & Linux Standalone“ platformom s ciljanom Windows platformom i 64 bitnom arhitekturom, kreirala autorica rada

Dolje lijevo može se vidjeti popis svih mogućih platformi za koje možemo izgraditi igru. Najuobičajeniji za testiranje je prvi na popisu „PC, Mac, Linux Standalone“. Igra se može izgraditi samo na platformi na kojoj se trenutno nalazimo, i zato s desne strane nije moguće označiti Mac OS ili Linux ukoliko se radnja vrši na Windowsima. Ispod „Target Platform“ je opcija za odabir arhitekture, a nudi opcije za 32 i 64 bitnu arhitekturu, te samo za 64 bitnu arhitekturu.

Sa željenim odabranim opcijama i uvezenim scenama, odabire se „Player Settings“ i u Inspectoru se pojavljuje popis svih ostalih mogućnosti. U prvoj sekciji bira se ime kompanije, ime finalnog produkta, zadana ikona aplikacije i zadana slika kursora.

Slika 4.48 Player Settings za buildanje igre, kreirala autorica rada

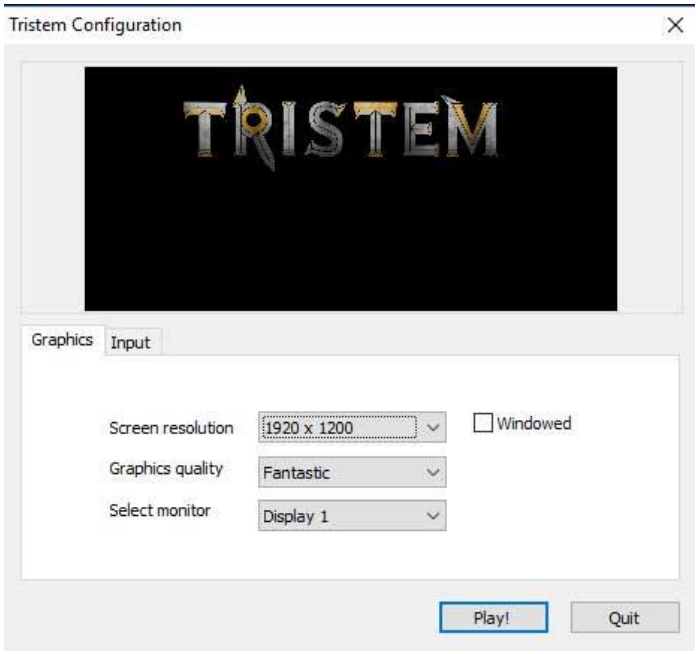
U drugoj sekciji imamo postavke rezolucije i samostalne player opcije. U rezolucijskim opcijama pod sekcijom „Aspect ratio“ možemo odabrati sve omjere slike s kojima želimo da igra bude kompatibilna. Ako odaberemo sve, slika će biti prilagodljiva svim standardnim omjerima zaslona. U opcijama „FullScreen Mode“ imamo „D3D9 Fullscreen Mode“ što je zapravo Direct X9 API na starijim Microsoftovim platformama. „D3D11“ je novija inačica API-ja koji je podržan od Windows Viste pa na svim novijim Windows operativnim sustavima. Ono što nude „Full Screen“ modovi su dvije opcije; „FullScreen Mode“ i „Exclusive Mode“. Fullscreen Windows označava prozor koji se proteže po cijelom zaslonu i podržava brze izmjene fokusa (alt - tab), dok „Exclusive mode“ to ne podržava. Prva opcija zauzima malo više memorije budući da zahtijeva dodatno renderiranje radne površine. Obično korisnici preferiraju prvu mogućnost, osobito kada se radi o radnim aplikacijama, no niti igre nisu iznimka. Međutim, ekskluzivni način rada može biti bolji kod zahtjevnijih igara kada ako profiliranje otkriva da se igra približava granicama VRAM-a, pa se u nekim slučajevima ekskluzivnim načinom može zadržati kvaliteta performansa. Od Unity verzije 2017.0, D3d9 API više nije podržan. [3]

Ispod se nalazi sekcija „Splash Image“ koja je u suštini slika ili video koji se pojavljuje na samom početku pokretanja aplikacije. Po zadanome to je Unityjev logo i natpis „Made with Unity“. Ako koristimo besplatnu verziju Unity-ja, ovo se ne može mijenjati. Međutim, želimo li zaista distribuirati i prodati igru na neku platformu, poželjno je na ovo mjesto staviti vlastiti logo i ime kompanije kao i ime i logo te igre, a to je moguće samo uz licencu.

Posljednja je sekcija „Other Settings“ na mnogim naprednim mogućnostima kao što su prostori boja, napredne rendering opcije, podrživost na novim tehnologijama kao što su VR, GPU skinning te opcije optimizacije igre. Kod ove jednostavne igre 2D igre dovoljno je sve opcije ostaviti na zadanome.

Možemo također igru izgraditi za web, odabirom na WebGL platformu, Xbox, Playstation, Android, iOS, Windows Store, Samsung TV ili Facebook, no za mnoge od ovih potrebno je instalirati dodatne programe te u nekim slučajevima i posebne dozvole i licence.

Kada su postavke podešene po želji, odabire se „Build and Run“ u prozoru „Build Settings“, a zatim se otvara novi dijaloški okvir u kojem odabiremo datotečno mjesto i ime izvršne datoteke aplikacije. Odabirom na „Save“ igra se počinje kompajlirati, te se nakon nekoliko trenutaka otvara novi prozor za konfiguraciju aplikacije. [3]



Slika 4.49 Konfiguracijski prozor za odabir rezolucije i kvalitete prije pokretanja igre

Ovdje odabiremo željenu rezoluciju zaslona, kvalitetu grafike, monitor, te označavamo kućicu „Windowed“ ako želimo da prozor igre bude podložan smanjivanju. Pritiskom na „Play!“ možemo pokrenuti igru te je testirati na računalu.

5. Zaključak

Proces razvoja side scroller igre možemo razdijeliti na puno radnih faza, u što spada planiranje, dizajn koncepta i algoritama te slijeda željenih događaja, dizajn scenografije, crtanje, animiranje, implementiranje animacija i tranzicija, programiranje svake moguće željene interakcije i funkcije u igri, spremanje i pohranu podataka, mnogobrojne izmjene, višestruko testiranje iza svakog novog koraka, te u konačnici izgradnja te igre u format čitljiv određenoj platformi.

Izrada svake, pa čak i najjednostavnije igre u bilo kojem game engine-u proces je koji zahtijeva specifična znanja iz mnogobrojnih područja; dizajna, animacije, programiranja i skriptiranja. Upravo zbog toga je u svakoj ozbiljnoj game industriji svaka grana ovog posla podijeljena na stručnjake iz određenih područja.

Dva ključna dijela ovog projekta su izrada karakterne modularne animacije te izrada igre koristeći te animacije kao središnju točku i centar projekta, i zato je za animiranje potrebno mnogo vremena, što je opće poznato za bilo koju vrstu animacije, pa tako i za animaciju namijenjenu igrama. Principi animiranja nisu se promijenili još od zlatnog doba animacije, jer to su ujedno i pravila fizike i kinematike pa možemo zaključiti da je jedino što se mijenja je tehnologija izrade animacije, ali i video igara, jer svaki game engine omogućava posredovanje putem grafičkog sučelja, što uvelike olakšava učenje, kao i činjenica vrlo velike zajednice ljudi koji koriste Unity Editor i C# programski jezik koji je također zbog svoje rasprostranjenosti i univerzalnosti preferiran nad ostalim jezicima.

Ono što se također može zaključiti jest da ovaj proces nije linearan, već više nalikuje regresijskoj analizi gdje se u fazi produkcije uvijek vraćamo iz implementacije na testiranje i ispravljanje te obratno.

U Varaždinu: _____

Potpis: _____

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, DORJA HATLAK *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom KREIRANJE MODULARNE ANIMACIJE U PROGRAMSKOM OKRUŽENJU SPRITER PRO TE IMPLEMENTACIJA RJEŠENJA U 2D RAČUNALNOJ IGRI *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Dorja Hatlak
(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, DORJA HATLAK *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom KREIRANJE MODULARNE ANIMACIJE U PROGRAMSKOM OKRUŽENJU SPRITER PRO TE IMPLEMENTACIJA RJEŠENJA U 2D RAČUNALNOJ IGRI *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Dorja Hatlak
(vlastoručni potpis)

6. Literatura

[1] Richard Williams: The Animator's Survival Kit, Farrar, Straus and Giroux, New York, 2009.

[2] Spriter Pro User's Manual Version 1.4, dostupno na: http://www.brashmonkey.com/spriter_manual/index.htm (on-line verzija korisničkih uputstva ne sadrži informacija o datumu posljednjeg ažuriranja sadržaja kao i nastanka istog)

[3] Unity 5.6 web dokumentacija, dostupno na: https://docs.unity3d.com/560/Documentation/Manual/UnityManual.html (Sadržaj nastao 31. ožujka, 2017 godine)

[4] Game Art Co: 2D Animation for Games, Udemy kurs, dostupno na: https://www.udemy.com/2d-animation-for-games/learn/v4/content (Zadnje ažurirano u listopadu 2017. godine)

[5] Neil North: Starting 2D Game Development in Unity with C#, Udemy kurs, dostupno na: https://www.udemy.com/2d-game-development-unity/ (Zadnje ažurirano u studenom 2017. godine)

[6] Ben Tristem, GameDev.tv by Ben Tristem, Rick Davidson: Complete C# Unity Developer 2D - Learn to Code Making Games, Udemy kurs, dostupno na: https://www.udemy.com/unitycourse/ (Zadnje ažurirano u rujnu 2018. godine)

[7] Weebneel: 20 Different Types of Animation Techniques and Styles, dostupno na: http://webneel.com/different-types-of-animation-styles (Članak ne sadrži informacije o datumu nastanka)

[8] D.Vusić, Z.Sabati, A.Bernik, 3D modeliranje u primjerima 1., Sveučilište Sjever, Varaždin, 2015.

[9] D.Vusić, A.Bernik, R.Geček 3D modeliranje u primjerima 2., Sveučilište Sjever, Varaždin, 2016.

[10] Sue Blackman: Beginning 3D Game Development with Unity : All-in-one, multi-platform game development, aPress, Berlin, Germany, 2014

[11] Steven Kent: Ultimate History Video Games, Random House USA Inc.,New York, 2011

7. Popis slika