Izrada 2D igre za Android u Unity-u

Raguž, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Raguž

IZRADA 2D IGRE ZA ANDROID U UNITY-U

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

LUKA RAGUŽ

Matični broj: 45893/17–R

Jmbag: 0016129635

Studij: Informacijski sustavi

IZRADA 2D IGRE ZA ANDROID U UNITY-U

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, srpanj 2020.

Luka Raguž

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom završnom radu prikazan je proces izrade 2D Android igrice u programskom alatu Unity. Odabrani žanr za igricu je „Platform Game“ koji se također može naći i u drugim oblicima kao što su „Platformer“ ili „Jump 'n' Run“ igre. Na početku završnog rada opisani su alati u kojima je izgrađena igrica, a to su cross-platform game engine Unity i Microsoft Visual Studio. Nakon opisa korištenih alata, detaljno je objašnjen cilj i svrha igrice kao i sama inspiracija i ideja za izradu iste. Nadalje slijedi glavni dio, odnosno opis izrade programskog rješenja. Sam opis podjeljen je u opis izrade kroz programski alat Unity te izradu C# skripti u alatu Microsoft Visual Studio. Izrada igrice dijeli se u par većih cjelina kao što su izrada glavnog protagonista, protivnika, GUI-a (eng. „*Graphical User Interface“*) i dizajniranja levela. Pod kreiranje glavnog protagonista podrazumjeva se izrada kretnji, skoka i napada za istog što je popraćeno određenim animacijama izrađenim kroz Unity. Izrada protivnika se sastoji od kreiranja animacija za kretnje te načina patroliranja od jednog kraja do drugoga. Sljedeći dio pokriven u ovom završnom radu je izrada grafičkog korisničnog sučelja koja se također sastoji od više dijelova. Prvi i glavni dio je sučelje koje omogućava kretnje na mobilnom uređaju bez kojih igra ne bi bila igriva. Manji, ali i dalje jako bitan dio je HUD (eng. „*Heads Up Display*“) koji omogućava korisnicima praćenje ostvarenog rezultata, broja preostalih života i ostalih stvari potrebnih za bolje ostvarenje igrajućeg iskustva. Zadnji dio pokriva opis načina na koji su level-i dizajnirani i sama demonstracija istog. Kraj završnog rada obuhvaća zaključak o radu i izrađenoj igrici te literatura korištena za njenu izradu.

Ključne riječi: Unity, C#, Android, Platformer, Video igra

Sadržaj

1. Uvod

Mobilna tehnologija, kao i gaming industrija od svojih začetaka bilježe kontinuirani rast i razvoj. Dok su u početcima mobilni uređaji služili samo za komunikaciju, danas imaju slične mogućnosti kao i računalo. Jedna od tih mogućnosti je igranje video igara koje su rasprostranjene u velikoj mjeri. U suvremeno doba, skoro svaki član obitelji ima mogućnost posjedovanja mobilnog uređaja što uključuje i djecu koja su ciljana publika za proizvođače video igara. Igrice na računalima su stekle popularnost još i krajem prošlog stoljeća, dok su se mobilne igrice u značajnijoj mjeri krenile proizvoditi tek prije svega desetak godina te je njihova proizvodnja aktualna još i danas te neće prestati. Kako napreduje njihov razvoj, normalno je da je i konkurencija na tržištu velika. Svaka igra može uspijeti i tako svaka osoba ima šansu pridonijeti gaming svijetu ukoliko ima dovoljno znanja i mašte da osmisli inovativnu ideju koja će se svidjeti publici. Svakim danom sve je više igrica, kako kopija, tako i onih sa novim tematikama kojih će uvijek biti jer su one plod ljudske inspiracije kojoj nema kraja.

U ovom radu obrađena je Android igrica koja je žanra „Platformer“. Ovaj žanr je jedan od napoznatijih kategorija računalnih i konzolnih igara, koji se isto tako lako može prenijeti i na mobilni svijet što je u ovom radu slučaj. Ovakve igrice su se krenule proizvoditi još u osamdesetim godinama prošloga stoljeća od kojih se može izdvojiti najpopularnija za koju je svaka osoba čula, a to je „Super Mario“ čiji je tvorac „Nintendo“. Razvoj je krenio sa žanrom „Platformer“, te se „Super Mario“ igrice proizvode još i danas sa istom tematikom, a različitim žanrovima. Što se tiče ovog primjera, sama igrica je dosta jednostavna te nadogradiva. U igrici su prikazane glavne mehanike lika i protivnika te je dizajnirano deset kratkih levela u kojima igrač može iskušati svoje sposobnosti uživajući uz ambijent i vraćanje u retro stil igara. Kroz rad je opisan svaki dio kroz koji je bilo potrebno proći da bi se igra završila. Također će detaljno biti opisani glavni dijelovi skripti koje su omogućile glavne funkcije ove igre.

Ovaj završni rad sam vidio kao inspiraciju za izradu vlastite igrice te način da naučim nešto korisno uz rad na onom što volim. Kao što gaming industrija billježi rast, tako se razvija i mobilna tehnologija stoga sam se odlučio za spoj ta dva svijeta i izradu 2D Android igrice žanra „Platformer“ u Unity-u što će biti objašnjeno u narednim poglavljima. Također, rad u C#-u danas je aktualan u velikoj mjeri te je svaka novonaučena stvar na tom području jedan korak napred stoga smatram kako će iskustvo na ovom radu biti početak iz kojeg će se dosta napredovati.

2. Metode i tehnike rada

Metode i tehinike korištene za izradu ovog rada su prvotno glavna dva alata korištena za izradu, a to su „Unity“ i „Microsoft Visual Studio“ odnosno jezik C#. Uz glavne alate, korištena su i razna druga pomagala kao što je „Inkscape“, alat za kreiranje sitnih ilustracija koje su potrebne za igru izuzev ilustracija koje su preuzete kao „assets packovi“ i „BFXR“ besplatan alat za izradu zvučnih efekata. Za kraj, izrađen je videozapis u kojem se može vidjeti kratka demonstracija igrice.

Što se tiče pripreme za izradu završnog rada, veliku ulogu je odigrao kolegij „Progamsko Inžženjerstvo“ na kojem sam naučio raditi u Microsoft Visual Studiju i koristiti programski jezik C#. Nadalje, najveći fokus je bio proučavanje kursova na opće poznatoj edukativnoj stranici „Udemy“. Kao pripremu za završni rad proučio sam i završio dva kursa koja su mi uvelika pomogla za ostvarenje programskog rješenja. Prvi kurs se temeljio općenito na rad sa C#-om što je služilo kao podsjetnik gradiva te je također uključivao i upoznavanje sa Unity-em i primjenu C#-a u izradi igrica. Drugi kurs je bio platformerskog karaktera, te se fokusirao na izradu „Platformer“ igara u Unity-ju. Nakon završetka kursova sam imao dobar temelj za početak rada u Unity-ju te sam uz to, proučavao korisna poglavlja na internetu i knjigama koja su pomogla pri rješenju.

Uz sve korištene metode i tehnike, naporan rad i uloženo vrijeme u samostalno proučavanje alata je jedan od najbitnijih stvari koje su rezultirale izradi finalnog programskog rješenja.

3. Opis korištenih alata

Kao što je već navedeno, za izradu ove Android igrice korišteni su razni alati od kojih su glavni Unity i Microsoft Visual Studio. Pomoćni alati korišteni za izradu cijelog rada su Inkscape i BFXR koji su korišteni u maloj mjeri te im nije potrebno dati veliku pažnju u ovom dijelu rada. Za ovaj rad je odabran game engine Unity jer se temelji na strukturiranom objektno orijentiranom jeziku C# s kojim sam bio upoznat i prije početka izrade rada. Za pisanje skripti u C#-u koristi se alat Microsoft Visual Studio, razvijen od strane Microsofta. Ova dva jako moćna alata detaljno su opisana u sljedećim poglavljima.

3.1. Unity

Unity je alat namjenjen za izradu video igara. Ukoliko je nečiji cilj napraviti igricu te svoju maštu pretvoriti u kreacije, Unity je idealan softver za započeti. Izuzetno je jednostavan za korištenje te besplatan ukoliko se ne prekoračuje određeni limit zarade. Unatoč tome što je besplatan, profesionalni je alat koji koriste i neke od najvećih tvrtki u industriji. Neki od popularniji naslova razvijenih u programskom alatu Unity su „Temple Run“, „Angry Birds“, „Superhot“, „Super Mario“, „Subnautica“ koje sam lično i imao priliku odirati. Korištenje ovog softvera programeru omogućavaju da sve ne mora raditi sam. Unity razvojnim timovima može uštedjeti mjesece ili čak godine, što ima velik utjecaj na „indie“ programere koji tako mogu postati konkurencija većim tvrtkama i na isti način, započeti svoju karijeru. [1]

Unity, iako je jednostavan, sadrži ugrađene vježbe i upute za korištenje. Uz besplatne načine učenja moguće je kupiti dodatne sadržaje koje korisnicima omogućavaju brže svladavanje alata, način monetizacije gotovog proizvoda, dodatan rad sa animacijama, kreacije efekata i drugo. Uz navedene načine svladavanja alata, tu je i Unity dokumentacija koja sadrži sve potrebne stvari koje Unity nudi te koja je dostupna svima na korištenje. Unity je programski alat za kreiranje igrica, no za izradu finalnog proizvoda potrebna su i druga znanja kao što su izrada objekata što je sasvim drugo područje za koje je potrebno dodatno znanje. Ovaj Softver ima ugrađenu trgovinu imovine (eng. *Asset Store*) koja sadrži razna pomagala za izradu igara te nudi velik odabir besplatnih stvari za korištenje. [2]

Kao što svaka inovacija ima početak, tako je i Unity morao krenuti od jedne točke. U početku su to bila tri programera koja su se bez puno novca u džepu okupila u podrumu, te započela izradu nečega što će izmjeniti gaming industriju. Jedan od trojice, izvršni direktor (eng. *Chief Executive Officer*, skraćeno CEO) David Helgason je na jednostavan način objasnio što je to Unity. Helgason tvrdi da je Unity „*Skup alata koji se koriste za izradu igara i*

to je tehnologija koja izvršava grafiku, zvuk, fiziku, interakcije i umrežavanje.“ Nicholas Francis i Joachim Ante su druga i treća osoba koje su nam omogućile korištenje ovog softvera. Za razvoj i popularnost nije trebalo mnogo. Desetljeće nakon ove inovacije, nebrojeni broj programera koristio je Unity za izradu mnoštva videoigara kako za računala, konzole, preglednike tako i za mobilne igre. Ovaj softver pomogao je programerima da daju manju posvetu stvaranju temeljnje tehnologije videoigara te da fokus bude na tvorčevu kreativnost i umjetnost što je upravo ono, štoj jednu igru čini zanimljivom. [3]

3.1.1. Opis korištenih komponenti

U sljedećem poglavlju opisani su najbitniji pojmovi vezani uz Unity koji su korišteni za izradu programskog rješenja. Igrica je smještena u dvodimenzionalnom prostoru stoga su prvo objašnjene najbitnije stvari potrebne da bi se realizirala 2D igrica, a zatim slijedi opis korištenja fizike, skriptiranja, animacija i zvuka što je bilo potrebno da bi se realizirao finalni proizvod.

3.1.1.1. 2D komponente

Jedan od najbitnijih pojmova 2D komponenti je tipka 2D način rada na alatnoj traci. Kad je upaljena omogućen je 2D pogled koji postavlja ortografski prikaz ili perspektivnu slobodu (eng. *Perspective-free*). Ovaj način omogućuje da se lakšu vizualizaciju scene i olakšano postavljanje 2D predmeta. [4]

Grafički objekti su još poznati pod nazivom „Sprites“ koji se uređuju u ugrađenom uređivaču od strane Unity-ja. Prikazuju se pomoću „Sprite Renderera“ za razliku od 3D objekata koji koriste „Mesh Renderer“. [4]

Kod korištenja fizike u 2D načinu rada bitno je napomenuti da ovaj motor (eng. *Engine*) ima izgrađene potrebne stvari i komponente za rukovanje 2D fizikom kako bi stvari učinile optimiziranijima. Postoje komponente koje odgovaraju standardnim 3D fizikalnim komponentama kao što su „Rigidbody“ i „Collider“ dok se za 2D način rada koriste „Rigidbody 2D“ i „Collider 2D“. Postoji još fizičkih komponenti, no u ovom dijelu su navedene one koje su najviše korištene u radu. [4]

3.1.1.2. Fizika

Kao što su već ranije bile spomenute neke komponente fizike, u ovom odjeljku će biti ukratko opisane. Naime, riječ je o „Rigidbody 2D“ i „Collider 2D“ komponentama.

„Rigidbody 2D“ je komponenta koji objekt stavlja pod kontrolu fizikalnog motora (eng. *Physics engine*) što znači da se objekt ponaša po zakonima fizike. Ova komponenta nam nudi razne opcije od kojih je prva ta da imamo mogućnost biranja tipa ponašanja objekta. Objekt može biti dinamički, kinematički ili statički. Odabrane opcije definiraju ponašanje kretnje i interakcije sa drugim objektima. Najbitnije stvari korištene kod ove komponente su masa (eng.

Mass) koja, kao što sama riječ kaže, daje masu objektu. Nakon što je definirana masa tijelo će se ponašati po zakonima fizike i kretati se brže ukoliko je masa lakša ili sporije ako je masa teža. Također, ovisno o masi, tijela je teže srušiti ako im je masa veća. Sljedeći korišten pojam ove komponente je gravitacijska skala (eng. *Gravity scale*) koja, ovisno o unesenoj količini, daje gravitacijsku silu objektu. Zadnji pojam vrijedan spomena su ograničenja (eng. *Constraints*) koja mogu biti korisna u nekim situacijama ako želimo zamrznuti kretnje ili rotaciju u određenom smjeru. Tada koristimo ograničenja te odabirom određene osi, ograničavamo radnje. [4]

„Collider 2D“ je komponenta koja se koristi za realizaciju fizičkih sudara (eng. *Physical collisions*). Koristi se na način da bi nevidljivi sudarač (eng. *Collider*) trebao biti potpuno istog oblika kao i sam objekt kako bi se, najčešće, omogućile kretnje po istom. Postoje različite vrste ovo komponente koje služe za olakšavanje rada, a napravljene su po određenim geometrijskim likovima: [4]

- Circle Collider 2D
- Box Collider 2D
- Polygon Collider 2D
- Edge Collider 2D
- Capsule Collider 2D
- Composite Collider 2D

3.1.1.3. Skriptiranje

Skriptiranje je jedan od najbitnijih dijelova, ne samo u ovom slučaju, nego i kod izgradnje svake aplikacije. Skriptiranje služi kako bi se događaji u igri odigrali onako kako trebaju i kada trebaju. Uz to, skripte se mogu koristiti kod promjene animacija, stvaranje grafičkih efekata, kontrole fizičkog ponašanja objekata ili impementaciju prilagođenog AI sustava (pr. *Protivnik*). [4]

Za skriptiranje u Unity-ju koristi se standardna skripta Monobehaviour koja ima vlastite tehnike pristupa motoru (eng. *Engine*). U ovom dijelu dan je samo uvod u skriptiranje što će biti prošireno u sljedećim poglavljima kada se budu opisivale skripte korištenje za izradu programskog rješenja.

3.1.1.4. Animiranje

Unity ima ugrađen alat koji pomaže pri izgradnji animacija. Vrlo lak i jednostavan animacijski sustav omogućava programeru da uz određeni set slika, npr. Sprite-ova koji prikazuju kretnju lika kreira i samu animaciju uz odabir broja sličica po sekundi (eng. *Frame per second*). Prilikom kreiranja 2D igrica, za animacije je dovoljno postaviti 8 do 14 isječaka što nije slučaj kod izgradnje složenijih igrica. Unity-jev animacijski sustav također sadržava

Animator kontroler (eng. *Animator controller*) u kojem je moguće napraviti prijelaze između više animacija. [4]

3.1.1.5. Zvuk

Svaka igra bi bila nepotpuna ukoliko ne bi bilo zvučnih efekata. Unity ima dovoljno moćan i fleksibilan audio sustav koji može uvesti većinu formata audio datoteka te se lako mogu reproducirati isti. Kao što je slučaj u stvarnom svijetu, osoba kad čuje zvuk, može reći odakle zvuk dolazi te osjetiti njegovu udaljenost, kao i čuti zvuk na različite načine ovisno o njegovoj okolini. Sve je to također slučaj u Unity-ju te se uz ugrađen audio sustav, skriptiranje i brojan asortiman besplatnih zvukova može doći do lijepih i ugodnih audio efekata. [4]

3.2. Microsoft Visual Studio

Microsoft Visual Studio opće je poznati programski alat koji se koristi za razvoj računarskih programa. Integrirano je razvojno okruženje (eng. *Integrated development environment*, skraćeno IDE) koji podržava različite programske jezike kao što su C, C++, C#, F# koji koriste razne biblioteke [5]. Iako je Unity izvrstan za sastavljanje svijeta igara, u njega se ne može upisivati kod. Odabrani alat za ovaj projekt koji će služiti baš u tu svrhu je Microsoft Visual Studio koji omogućava korištenje poznatih značajki uređivanja i otklanjanja pogrešaka. Visual Studio u ovom slučaju nije samo uređivač teksta (eng. *Text editor*) već također ima duboku integraciju sa Unity-jem koje omogućavaju dolazak do određenog rješanja uz manje vremena te osigurano poboljšanje produktivnosti. [6]

4. Žanr i ideja programskog rješenja

U sljedećem poglavlju dolazimo do opisa žanra igrice, ideje za izgradnju programskog rješenja te cilja i svrhe igrice. Za igricu je odabran žanr „Platformer“ kao što je već spomenuto, ali ne i dovoljno objašnjeno. U prvom dijelu ovog poglavalja opisan je žanr, što je sve potrebno da se igra može zvati „Platformer“ te kako je žanr nastao. U daljnjim dijelovima slijedi opis ideje za igricu, te definiciju njenog cilja i svrhe.

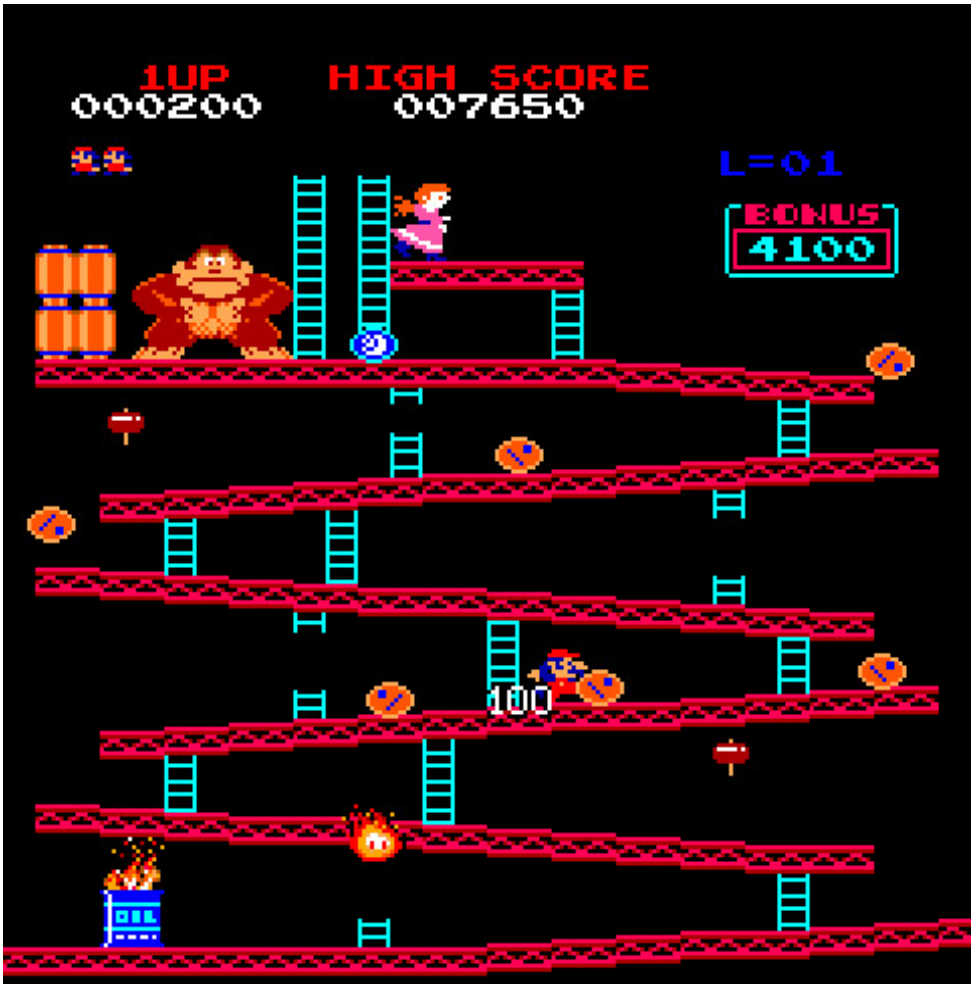
4.1. Žanr igrice (Platformer)

Platformna video igra (eng. *Platformer*) je podžanr akcijskih video igara koje tradicionalno sadrže dvodimenzionalnu grafiku. U platformnim igricama igrači najčešće kontroliraju likove koji se kreću i skaču po različitim platformama na zaslonu. To su uobičajno igrice koje imaju bočnu pomičnu kameru koja u našem slučaju, prati lika, no to može jako zavisiti do igre jer postoje i platformne igrice koje sadrže jednu nepomičnu kameru (eng. *Single screen movement).* To je vrsta igrice koju je prilično jednostavno stvoriti i odličan su način za upuštanje u svijet dizajna video igara. [7]

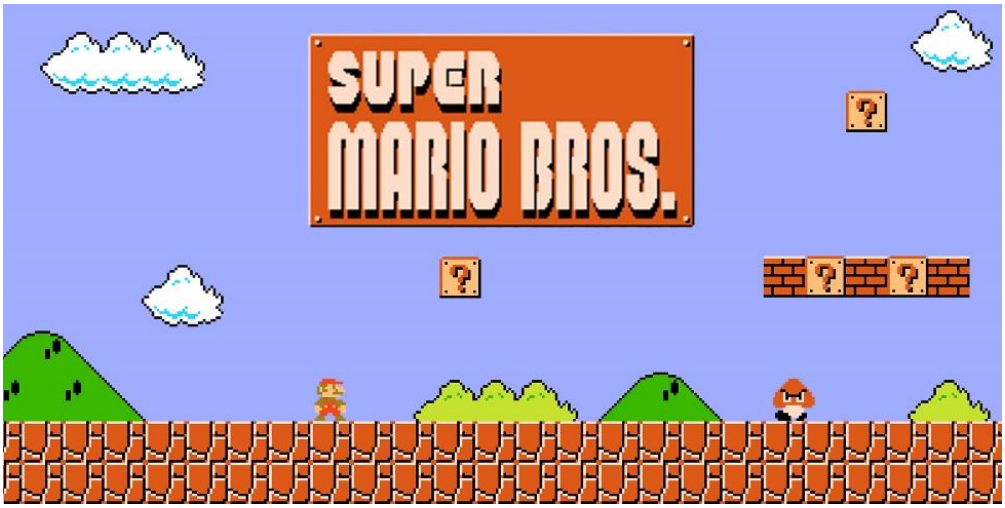
Od početka razvoja video igara, svaka je generacija definirana određenim igricama. Tako je sve trebalo krenuti i kod ovog žanra. Kroz povijest, postojale su razne igrice koje su obilježile neka razdoblja kao što su Halo, Uncharted, Doom i druge, no čak i uz toliko drugih igrica i žanrova žanr platformnih igara održava svoju popularnost sve do danas. Ove igrice su vjerovatno kroz desetljeća bile univerzalno najomiljenije igrice, kako mlađe, tako i starije publike. [8]

Može se reći da je sve krenulo od ere sa jednim zaslonom (eng. *Single screen era*) u kojoj su nastajale jednostavne platformne igrice gdje nije bilo potrebe da se kamera miče ili prati protivnika. Kao što samo ime kaže, bio je potreban jedan zaslon na kojem se odvijala cijela igrica. Ukoliko se igrač pokušao zabiti u rub, razina se ne bi pomicala desno ili lijevo. Igrica koja je popularizirala platformni žanr bila je „Space Panic“, arkadna igra iz 80-ih godina prošloga stoljeća u kojoj se igrač mogao penjati po ljestvama, bez pretjeranih dodatnih kretnji kao što je na primjer skok. Neposredno nakon ove igrice, Nintendo je 1981. godine osmislio igricu „Donkey Kong“ što se zapravo smatra prvom istinski platformnom igricom. Cilj ove igrice je bio je popeti se do majmuna i osloboditi ljubav glavnog protagonista. Ova franšiza završila je sa još dva nastavka nakon čega sa različitim unaprijeđenjima dolazimo do začetka igrica sa pomičnom kamerom (eng. *Side scrolling era*) čiji su predstavnici „Jump Bug“ i „Super Mario Bros“. U samim početcima, to su bile jednostavne igrice nakon čega su krenula određena unaprijeđenja. Jedno od njih je paralaksni efekt (eng. *Parallax effect*) koji se prvi puta pojavio

u igrici „Pac-Land“ 1984. Nadalje je krenio razvoj 2.5D i 3D igrica koje su gaming industriju prebacile na sljedeću razinu. [8]



Slika 1. Primjer prve istinske platformne igrice "Donkey Kong"



Slika 2. Primjer "Super Mario Bros" platformne igrice

4.2. Elementi žanra (Platformer)

Svaki žanr ima svoje elemente igre koju žanr razlikuju od drugih žanrova. Glavni od elemenata koji žanr čini platformnom je kao što samo ime kaže, kreiranje svijeta koji može biti 2D, 3D ili 2.5D u kojem se glavni protagonist kreće po različitim platformama. Da bi se igra učinila zanimljivijom, bitno je napraviti različite izazove da to bude teže. U ovom dijelu kratko su opisani elementi koji su korišteni u ovoj igri, ma da ih ima znatno više.

Jedan od načina dodavanja izazova je postavljanje raznih zamki ili protivnika koji mogu naštetiti igraču. Izgradnja protivnika se može napraviti implementacijom jednostavne umjetne inteligencije (eng. *Artificial intellingence*) gdje se protivnicima može omogućiti kretnja, napadi, pucanje i slično. Nadalje, postavljanje zamki, prepreka i misaonih puzli osiguravaju igraču bolje iskustvo i zabavniju igru.

Sljedeći element ovog žanra je omogućavanje igraču da prikuplja stvari (eng. *Collectibles*) koji će mu povećati finalni rezultat ili koje može iskorititi u neku drugu svrhu. Pravljenjem većeg broja različitih vrsta stvari omogućava igraču bolju zabavu i zanimljivije praćenje vlastitog rezultata. Prikupljene stvari također mogu poslužiti u vrstu unaprijeđenja glavnog lika što također može rezultirati poboljšanju igrajućeg isustva (eng. *Gaming experience*).

Još jedan, manje bitan element kojeg nema svaki platformer, ali koji igru može učiniti znatno napetijom i zabavnijom je izgradnja štoperice (eng. *Timer*) koja igraču odbrojava vrijeme potrebno za prelazak mape ili nivoa.

4.3. Ideja i cilj igre

Prilikom pokretanja bilo kakvog projekta, unaprijed bi trebala biti jasna vizija finalnog rješenja kao što je bitno i imati ideju kako do njega doći. Kao što je već ranije navedeno, lak način za zaranjanje u svijet izrade igrica jer baš „Platformer“ žanr kojeg sam odabrao iz istog razloga te sam sam lično igrao dosta takvih igrica. Nakon što je bio odabran žanr igrice, potrebno je bilo osmisliti tematiku koja će pratiti igricu.

U ovom primjeru odabran je ambijent džungle u kojem glavni protagonist kroz različite nivoe (eng. *Levels*) skuplja voće te se bori sa protivnicima kao što su pčele, orlovi, majmuni, divlje svinje, gljive i ježevi. Na samom početku igre igraču su dodjeljena tri života koja može izgubiti na razne načine. Igrač prilikom borbe sa protivnicima može skočiti na njihovu glavu te ih tako poraziti ukoliko protivnik nije jež. Jež sa svojim bodljama može ubiti glavnog protagonista ukoliko se on pronađe na njegovim leđima te mu se oduzima jedan od tri života. Na nekim mapama postoji sakriveno oružje koje igrač može pokupiti i tako si olakšati put do cilja.

Cilj igrača je sakupiti što je više moguće voća, poraziti što više protivnika i na vrijeme pronaći zlatnu breskvu koja završava level. Navedene stvari je potrebno izvesti u određenom vremenskom periodu jer ukoliko to nije moguće, igraču se oduzima život. Nakon gubitka svih života, igraču se poništava osvojeni rezultat, te se stvari vraćaju na početne.

5. Izrada programskog rješenja

U sljedećem poglavlju dolazimo do glavnog dijela završnog rada, a to je izrada programskog rješenja. Pod programsko rješenje smatra se primjer igrice izrađen u dosadašnje opisanim alatima. U nadolazećim poglavljima slijedi opis izrade igrice korak po korak, kako kroz Unity, tako kroz Microsoft Visual Studio. Svaki dio je opisan te je veći dio pažnje posvećen najbitnijim dijelovima programskog koda koja su omogućila igrivost igrice.

5.1. Pripema materijala

Kao što je za većinu projekata potreban rad u timu, tako je i za izradu igrice dosta bitan timski rad. Tim koji proizvodi igricu nema samo programere, već i testere, dizajnere, ljude koji osmišljavaju priče i druge ljude koji različitim sposobnostima doprinose izradi finalnog proizvoda. Kako je u ovom završnom radu fokus na izradi programskog rješenja, a ne dizajniranje, za glavnu grafiku i animacije preuzeti su gotovi proizvodi od treće strane (eng. *Third-party*).

Za ovu igricu korištene su tri vrste različitih paketa od kojih su dva kupljena i preuzeta sa stranice za izradu 2D imovine (eng. *2D Game Assets*) te je jedan preuzet kao besplatan paket (eng. *Freebie*) za izradu grafičkog sučelja (eng. *Grafical User Interface*, skraćeno GUI). Stranica korištena za nabavu animacija i ilustracija je Game Art 2D [9] čiji se materijali također mogu naći u ugrađenom „Asset Store-u“ unutar programskog alata Unity.

Prvi paket sadrži sličice (eng. *Sprites*) za glavnog lika što je preko programskog alata Unity pretvoreno u animacije te će biti pojašnjeno u sljedećim poglavljima. Osim glavnog lika, paket sadrži šest protivnika koji se lijepo uklapaju u ugođaj tematike džungle. Protivnici su divlja svinja, kornjača, orao, gljiva, pčela te majmun. Svi nabrojani protivnici također imaju sličice kretnje što je kroz programski alat Unity pretvoreno u animacije. [10]

Drugi paket sadrži set pločica (eng. *Tileset*) koji su u prijevodu različite platforme po kojima se igrač može kretati. U ovom paketu dostupne su platforme u dva izdanja, crvenom i sivom uz koje dolaze i razni objekti koji mogu pomoći u poboljšanju ambijenta igrice. [11]

Treći paket je paket koji sadrži elemente grafičkog korisničkog sučelja pomoću kojeg su realizirani menu-i. Isti paket besplatan je za korištenje te je preuzet sa iste stranice. Pomoću ovih elemenata izgrađen je glavni izbornik, izbornik za odabir nivoa te iskočni izbornici (eng. *Pop-up menus*) za pauzu, završetak nivoa i kraja igre ukoliko igrač izgubi sve živote. [12]

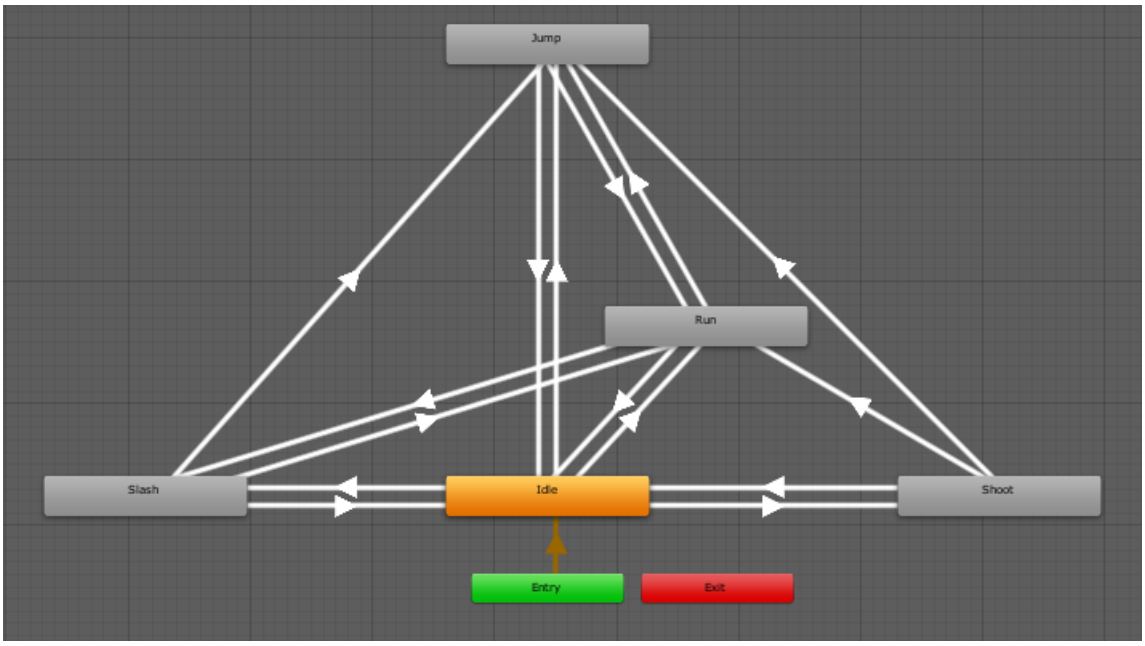
Nakon odabranog materijala za izradu igre, sve je spremno za početak izgradnje glavnog lika, protivnika kao i njihovih funkcionalnosti te upuštanje u dizajn novog svijeta koji je rezultat ljudske inspiracije.

5.2. Izrada glavnog lika

Početak izrade programskog rješenja započeo je sa izradom glavnog lika što nije tako jednostavan proces. Za izradu glavnog lika prvotno su potrebne animacije koje će se kroz igru mijenjati kako korisnik upravlja likom. Programski kod skripte glavnog lika se može podijeliti u dio vezan uz kretnju lika, napad te interakciju s drugim objektima. Za početak ovog poglavlja, objašnjeno je animiranje lika te kako je ono postignuto. Nadalje, dolazimo do opisa programskog koda koji omogućavaju te iste pokrete te promjenu animacija između različitih akcija. Na samom kraju opisan je dio koda vezan uz interakciju glavnog lika sa drugim objektima te način na koji su oni postignuti.

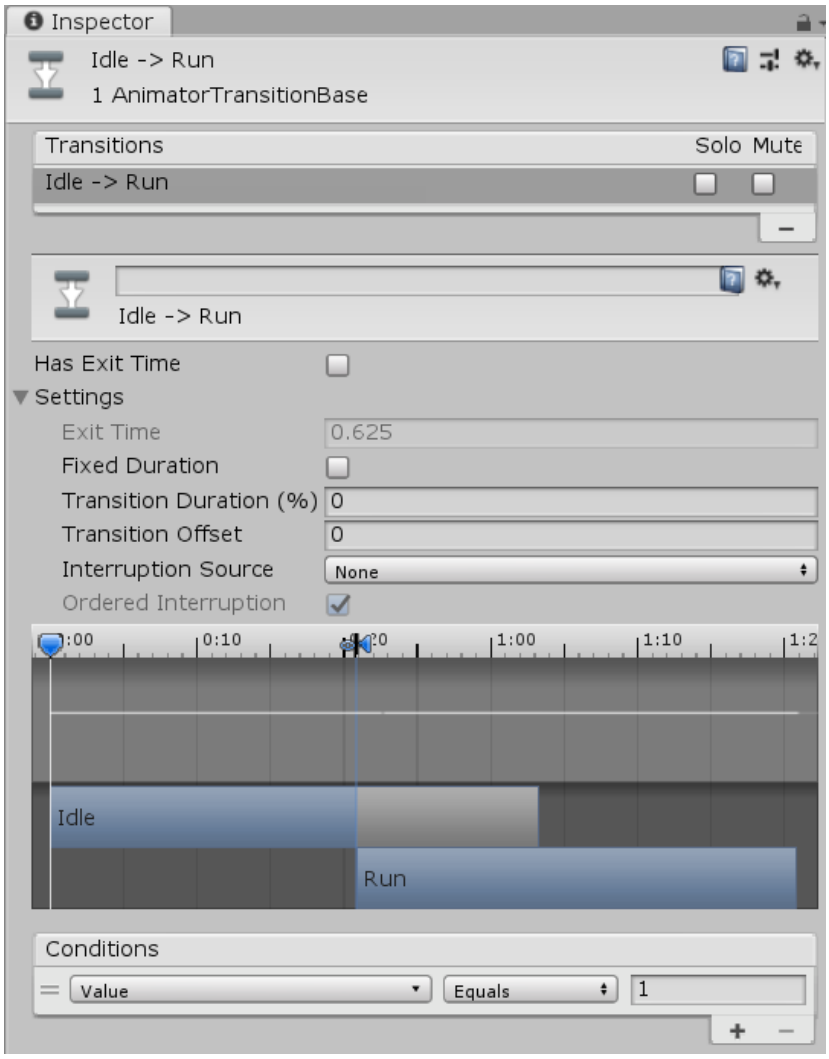
5.2.1. Animiranje glavnog lika

Kod kreiranja igrica koje će biti igrane na mobilnom uređaju, bitno je ne napraviti prekomplicirane kontrole jer bi igrivost mogla postati jako teška. Prilikom kreiranja animacija kreirane su one najosnovnije koje će korisniku omogućiti kretnju i borbu. Zamišljeno je da se koriste animacije trčanja, skakanja, pucanja i lupanja koje će se izvoditi kroz svega pet gumbova (eng. *Button*) prikazanih kroz mobilno korisničko sučelje (eng. *Mobile user interface*). Na sljedećoj slici može se vidjeti „Animator“ u programskom alatu Unity u kojemu se jasno vide sve animacije lika sa njihovim prijelazima (eng. *Transitions*). Na slici strelica označava iz kojeg stanja je moguće preći u koje te se može uočiti kako se iz skoka ne može prijeći u stanje pucanja ili lupanja što je napravljeno tako da se oteža igrivost. [21]



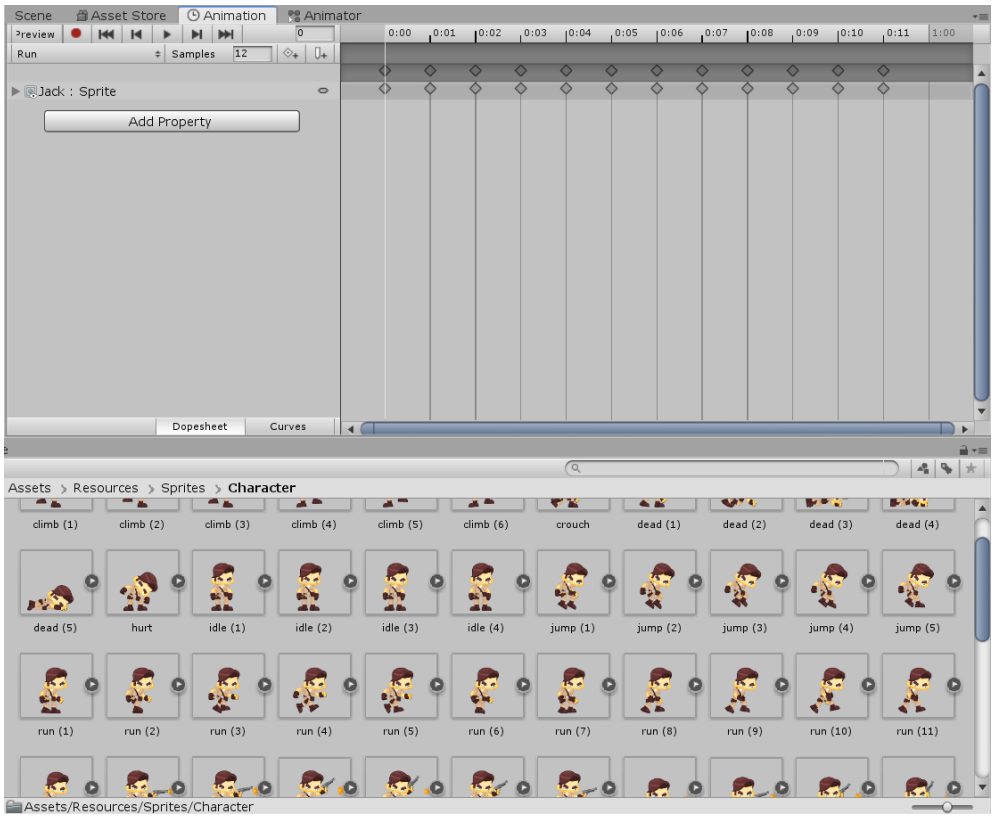
Slika 3. Animator glavnog lika

Prijelazi, odnosno tranzicije napreljene su na način da je isključen „Has Exit Time“ te smanjen „Transition Duration“ na 0 kako ne bi došlo do kašnjenja animacije koja slijedi nakon prvotne. Kroz „Animator“ je postavljen brojevni parametar koji ima vrijednost za svaku animaciju što će biti potrebno kod skriptiranja glavnog lika. [21]



Slika 4. Prikaz prijelaza sa jedne animacije na drugu

U dosadašnjem tekstu bilo je govora samo o „Animatoru“, no pitanje je kako napraviti animaciju koja se u „Animatoru“ koristi. Kao što je već spomenuto, za kreiranje animacije potrebni su određene sličice, odnosno „Sprite-ovi“ koje su u ovom slučaju preuzete od treće strane. Animacije su osnova svake igrice te Unity ima ugrađen alat za kreiranje istih. Kreiranje animacija je poprilično jednostavno jer sve što se traži je odabir sličica i broja uzoraka (eng. *Samples*). Animaciji je potrebno dati ime i spremna je za korištenje. Osim ovakvog načina izrade animacija postoji mnogo drugih načina te se u ovom radu koristio još jedan što će biti prikazano u daljnjem tekstu, a što se tiče samog glavnog lika ovo su sve potrebne animacije za realizaciju zamišljenog.



Slika 5. Primjer izrade animacije

5.2.2. Kretnje glavnog lika

Upravljanje osobom ili objektom u igrici je jedan od esencijalnih elemanata svake igre. Prilikom igranja igre to može izgledati jednostavno, no iza svake kretnje, okretaja, skoka ili čučnja stoji dio programskog koda koji baš tu krenju omogućavaju. U ovom dijelu opisane su kretnje glavnog lika koji se može kretati lijevo i desno, skakutati po platformama te dok je u zraku aktivirati dupli skok. Svaki od ovih elemenata kretnje opisan je jedan po jedan uz prilog odgovarajućeg dijela programskog koda.

5.2.2.1. Horizontalne kretnje

Pod horizontalnu kretnju se podrazumijeva hodanje glavnog lika u lijevu ili desno stranu. U nekim igricama moguće je kretanje samo napred, dok je ovdje realizirana kretnja u oba smjera. Osim horizontalne kretnje, postoji vertikalna kretnja prema gornjem smjeru, a to je skok te je objašnjen u sljedećem poglavlju. Za realizaciju kretnje lijevo i desno zadužen je nitko drugi nego programski kod. Prvi put dolazimo do tog dijela gdje se uz pomoć koda postiže jedna od funkcionalnosti bez kojih igra ne bi bila igriva. To se postiže kroz pisanje skripte izrađene u programskom alatu Microsoft Visual Studio koristeći jezik C#. U nastavku možemo vidjeti jedan takav primjer što je zapravo kod zadužen za pokretanje glavnog lika te se nakon koda nalazi i obrazloženje istog.

```
//PlayerController
public float speedBoost;
bool leftPressed;
bool rightPressed;
bool isRunning;
Rigidbody2D rigidBody2D;
SpriteRenderer spriteRenderer;
Animator animator;

void Start()
{
   rigidBody2D = GetComponent<Rigidbody2D>();
   spriteRenderer = GetComponent<SpriteRenderer>();
   animator = GetComponent<Animator>();
}

void Update(){
   float initialSpeed = Input.GetAxisRaw("Horizontal");
   float playerSpeed = initialSpeed * speedBoost;
   if(playerSpeed != 0){
      Move(playerSpeed);
      isRunning = true;
   }else{
      Stop();
      isRunning = false;
   }
   if (leftPressed)
      Move(-speedBoost);
   if (rightPressed)
      Move(speedBoost);
}

void Move(float playerSpeed){
   rigidBody2D.velocity = new Vector2(playerSpeed, rigidBody2D.velocity.y);
   Flip(playerSpeed);
   if(!isJumping)
      animator.SetInteger("Value", 1);
}

void Stop()
{
   rigidBody2D.velocity = new Vector2(0, rigidBody2D.velocity.y);
   if (!isJumping)
      animator.SetInteger("Value", 0);
}

void Flip(float playerSpeed){
   if(playerSpeed < 0){
      spriteRenderer.flipX = true;
   } else {
      spriteRenderer.flipX = false;
   }
}
```

Ovo je djelomičan dio skripte glavnog lika koji prikazuje implementaciju kretnje. Na početku se nalazi deklaracija varijabli potrebnih za omogućavanje horizontalne kretnje. U Start() metodi instanciran je objekt tipa RigidBody2D koji je osnovni element potreban za osposobljavanje kretnje te se to omogućuje sa GetComponent<> [25]. Start() metoda se

automatski poziva na početku pa je perfektno mjesto za takve stvari. Za početak bitno je znati da je Update() metoda ona u kojoj se poziva svaka sličica (eng. *Frame*) te je u nju najbolje smjestiti stvari kao što je pomicanje lika.

Flip() metoda omogućava promjenu smjera glavnog lika koja je omogućena kroz „Sprite Renderer“. Na početku Update() metode deklarira se inicijalna brzina protivnika koja se definira odabirom tipke na tipkovnici ili gumba na mobilnom korisničkom sučelju. Inicijalna brzina nam u pravilu daje predznak kretnje odnosno negativan ukoliko je kretnja u lijevom smjeru ili pozitivan, ukoliko je kretnja u desnom. Nakon što je određen smjer kretanja, ista brzina se multiplicira sa brzinom kretnje lika, što je svima vidljiva varijabla (eng. *Public*) te joj se s toga može pristupiti kroz Unity sučelje. Nakon što je postavljena brzina glavnog lika, možemo izračunati vrijednost varijable „playerSpeed“ što je rezultat inicijalne brzine i brzine lika. Sada dolazimo do Flip() metode koja se poziva u Update() metodi te joj se pruža varijabla „playerSpeed“. Flip() metoda provjerava predznak date varijable te ukoliko je preznak negativan okreće lika u jednom smjeru, a ako je pozitivan u suprotnom. Ova stvar omogućena je pristupanjem elementu „FlipX“ koja okreće objekt po x-osi. Svrha ove metode je ta da se omogući kretnja u oba smjera, mada je to moguće i bez nje, ali bi imali osjećaj da glavni lik izvodi popularni ples „Moonwalk“. [21]

Move() i Stop() metode su dosta jednostavne, te kao što samo ime kaže omogućavaju kretnju i zaustavljanje. U obe metode koristit se varijabla „rigidBody2D“ kojoj se pridružuje vektor smjera pomoću metode Vector2() [22] koja prima dva parametra. U slučaju kretnje pružamo brzinu kretnje glavnog lika, dok u slučaju zaustavljanja istu postavljamo na nula kako bi glavni lik stao. Dalje, u obje metode možemo vidjeti provjeru da li lik skače, jer ukoliko skače, ne bi se smjele prikazivati animacije trčanja i mirovanja. U prvom slučaju se provjerava da li lik skače, te ukoliko ne skače znači da se kreće po tlu što bi značilo da se koristi animacija trčanja (eng. *Running*) koja za vrijednost ima cijeli broj 1. Vrijednost se postavlja na 1 sa metodom SetInteger() [33] koja funkcionira jednostavno te za parametar prima integer vrijednost. U drugom slučaju se također provjerava da li lik skače, te ukoliko ne skače prikazuje se animacija mirovanja (eng. *Idle*). [21]

Nakon što su objašnjene metode, može se preći na glavnu stvar, a to je sadržaj koda u metodi Update(). Na početku metode se definira brzina lika kroz inicijalnu brzinu i brzinu kretanja što je već ranije pojašnjeno. Inicijalna brzina se određuje pritiskom tipke što je realizirano kroz metodu GetAxisRaw() [32] koja kao parametar prima ime predefinirane radnje. Nakon izračuna brzine, provjerava se njena vrijednost. Vrijednost različita od nule označava krenju lika, dok vrijednost 0 označava mirovanje. Ovisno o tom uvjetu pokreću se metoda Move() ili metoda Stop() uz promjenu varijable „isRunning“ na odgovarajuću vrijednost koje će se koristiti za ostale dijelove koda. U Update() metodi se također odvija dio promjene predznaka brzine ovisno o tome da li je pritisnuta tipka kretnje u lijevo ili u desno. [21]

Nakon izrade ovog dijela koda lik se može kretati, no to nije sve što jednu igricu može učiniti zanimljivom. U sljedećem poglavlju objašnjena je implementacija vertikalne kretnje odnosno skoka u vis te kako je postignuta mogućnost duplog skoka.

5.2.2.2. Skakanje

Platformna igrica bez skakanja nije platformna igrica. U sljedećem dijelu dan je programski kod koji omogućava baš taj dio, a također je dio skripte glavnog igrača koji se nalazi na istom mjestu kao i prethodni kod horizontalne kretnje.

```
//PlayerController
public LayerMask whatIsGround;
public float boxWidth;
public float boxHeight;
public bool isGrounded;
public bool canDoubleJump = false;
bool isJumping;
Rigidbody2D;
Animator;

void Start()
{
   rigidBody2D = GetComponent<Rigidbody2D>();
   animator = GetComponent<Animator>();
}

void Update(){
   isGrounded = Physics2D.OverlapBox(new Vector2(feet.position.x,
feet.position.y), new Vector2(boxWidth, boxHeight), 360.0f, whatIsGround);
   if (Input.GetButtonDown("Jump")){
      Jump();
   }
}

void Jump(){
   if (isGrounded){
      isJumping = true;
      rigidBody2D.AddForce(new Vector2(0, jumpBoost));
      animator.SetInteger("Value", 2);
AudioController.instance.PlayerJump(gameObject.transform.position);
      Invoke("EnableDoubleJump", delayForDoubleJump);
      }
   if(canDoubleJump && !isGrounded){
      rigidBody2D.velocity = Vector2.zero;
      rigidBody2D.AddForce(new Vector2(0, jumpBoost));
      animator.SetInteger("Value", 2);
AudioController.instance.PlayerJump(gameObject.transform.position);
      canDoubleJump = false;
      }
}

void EnableDoubleJump(){
   canDoubleJump = true;
}
```

Iznad se nalazi programski kod zadužen za omogućavanje skoka i duplog skoka. Ovaj kod se nalazi u istoj skripti “PlayerController” kao i kod za horizontalnu kretnju stoga nema potrebe ponovnog opisivanja deklaracije varijabli i Start() metode.

Za početak zavirit ćemo u Update() metodu gdje se svaku sličicu provjerava da li se lik nalazi na tlu. Na početku se dodjeljuje istinita ili neistinita vrijednost “isGrounded” varijabli pomoću Physics2D.OverlapBox() metode [34] gdje se provjerava da li se međusobno diraju dva vektora. U ovom slučaju provjerava se da li noge lika (eng. *Player feet*) dodirivaju tlo koje ima ima predefiniran tag “Ground”. Ako noge dodirivaju objekt sa tagom “Ground” lik se nalazi na tlu, inače ne. Ovo stanje se provjerava svaku sličicu te je jako bitno za sljedeću metodu, a to je metoda Jump() koja se pokreće svaki puta kada korisnik pritisne tipku ili gumb za skok što je omogućeno preko GetButtonDown() metode [35] koja u ovom slučaju za parametar prima ime predefinirane radnje. [21]

Metoda Jump() provjerava da li je lik na tlu te ukoliko je postavlja vrijednost varijable “isJumping” na istinitu kako bi se mogle kontrolirati animacije u prijašnjim dijelovima koda. Glavna stvar koja omogućuje skok je dodavanje sile vrijednosti preko AddForce() metode [36] koja je pružena kroz Unity sučelje u varijablu “jumpBoost” koja je vidljiva svima (eng. *Public*) i iz tog razloga joj možemo pristupiti kroz Unity sučelje. Neposredno nakon skoka, potrebno je promjeniti animaciju lika na skok koja ima vrijednost cijelog broja 2 što je realizirano na isti način objašnjen u prošlom poglavlju. Isto tako, pokrenut je zvuk skoka koji je omogućen kroz instance klase “AudioController” te se zvuk stvara na mjestu pozicije skoka. [21]

Dok je lik u zraku omogućen mu je dupli skok što se ostvaraju sa Invoke opcijom u kojoj definiramo ime metode koja će se pokreniti te joj dajemo određeni vremenski period. U ovom slučaju za dani vremenski period pokreće se metoda EnableDoubleJump() te se varijabla “canDoubleJump” postavlja na istinitu dok je lik u zraku što je omogućeno preko Invoke() metode [24]. Kada lik padne, ako je već skočio ili ako je duže vrijeme u zraku dupli skok se onemogućava. Dio koda koji se nalazi unutar druge selekcije je već ranije objašnjeni programski kod skoka te se, kao što je vidiljivo, može pokrenuti samo u slučaju kada lik nije pozicioniran na tlu te ako je varijabla “canDoubleJump” istinita. [21]

5.2.3. Napad glavnog lika

Igra bez akcije ne bi bila zabavna. Za platformerske igrice opće je poznato da lik s kojim igrač upravlja ima neki način napada. To može biti skok na glavu, kao što je to primjer u popularnoj igrici „Super Mario“. Napad ručnim oružjem (eng. *Melee weapon*) ili streljno oružje (eng. *Ranged weapon*). U ovom primjeru napravljena su oba načina napada koja korisniku omogućavaju dovoljno mogućnosti da preživi put do cilja. Da bi stvar bila zanimljivija, streljno oružje, odnosno revolver nije uvijek dostupan za korištenje. Revolver se može naći u

određenim mapama gdje je nužan za prijelaz ili kao sakriveno oružje koje nije nužno za prijelaz mape, ali ga olakšava.

5.2.3.1. Napad skokom na glavu

Napad skokom na glavu je jednostavan potez kojim igrač, ukoliko dobro pozicionira lika te uspije skočiti na samu glavu hodajućeg protivnika eliminira istog. Ova mogućnost realizirana je tako što je glavnom objektu igrača dodan objekt dijete (eng. *Child object*) koji na sebi ima „Edge Collider 2D“ s kojim se omogućava provjera dodirivanja istog objekta sa drugim. Svaki protivnik ima svoju kutiju koja na dodir šteti igraču (eng. *Hitbox*). Na isti način, svakom protivniku je dan „hitbox“ za glavu koji je pokretač (eng. *Trigger*) te prilikom dodira sa igračem, omogućuje različite akcije. Ukoliko se dijete objekta igrača, noge igrača, dodirnu sa „hitbox-om“ glave protivnika, događaju se određene radnje. Sljedeći dio koda prikazuje baš taj dio te je u nastavku dodatnu objašnjen. [21]

```
//EnemyHeadController
public GameObject enemy;
private void OnTriggerEnter2D(Collider2D collision) {
    if(collision.gameObject.CompareTag("Player Feet")){
        GameController.instance.PlayerStompsEnemy(enemy);
    ParticlesController.instance.ShowDustParticle(enemy.transform.position);
    }
}

//GameController
public int enemyValue;
public void PlayerStompsEnemy(GameObject enemy){
    enemy.tag = "Untagged";
    Destroy(enemy);
    UpdateScore(enemyValue);
}
```

Ovaj kod se nalazi u dvije različite skripte te svaki dio radi određene poslove kako bi se omogućila funkcionalnost skoka na glavu.

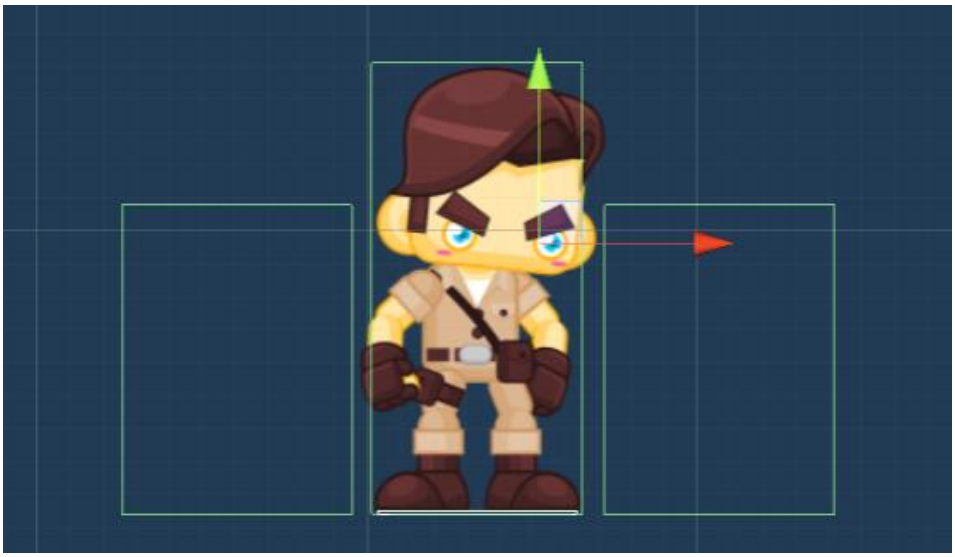
Prvi kod dio je skripte “EnemyHeadController” koja se pričvršćuje (eng. *Attach*) na glavu protivnika. Taj dio izvodi određeni dio koda u trenutku kada se dogodi kolizija objekta na koji je skripta pričvršćena sa objektom koji ima tag “Player Feet”, a to su noge glavnog lika. Ukoliko dođe do takve situacije pokreće se metoda PlayerStompsEnemy() iz skripte “GameController” te joj se pruža isti protivnik kako bi se mogle dogoditi sljedeće akcije. Za ovu implementaciju koristi se OnTriggerEnter2D [26] što uvelike olakšava stvari. Nakon skoka na glavu, poziva se metoda djelovanja čestica (eng. *Particle effect*) koja nakon uništenja protivnika, stvara prikaz prašine. [21]

Drugi dio koda dio je skripte “GameController” u kojoj se događa magija. Nakon što je pozvana ista metoda, odnosno nakon kolizije protivnika i igrača, protivniku se mijenja tag sa “Enemy” na “Untagged” pomoću čega se omogućava da taj protivnik više ne može naštetiti

glavnome liku. Neposredno nakon toga, protivnik se uništava pomoću metode Destroy() [30] te se na dosadašnji rezultat igraču dodaju dodatni bodovi koji ovise o protivniku kojeg je uništio. [21]

5.2.3.2. Udaranje nožem

Druga vrsta napada je napad nožem koji igraču omogućа ručni napad na kratku udaljenost. Na isti način kao i sa kreiranjem djeteta objekta, kreirana su još dva objekta koja će služiti kao pokretač akcije nakon što je napad napravljen. Ukratko rečeno, dva kreirana objekta imaju svoj 2D kockasti okvir (eng. *Box Collider 2D*) koji je inicijalno ugašen te se pali pritiskom tipke napada. Nakon što je pritisnuta tipka, izvodi se animacija napadanja te se provjerava da li se protivnik nalazi u „slash collider-u“ te ukoliko se nalazi događaju se određene akcije. Na idućoj slici se vidi način na koji su okviri napravljeni te se u nastavku nalazi odgovarajući kod koji omogućava ovu funkcionalnost. [13]



Slika 6. Kreiranje okvira za napad nožem

```
//PlayerController
public Collider2D leftSlashTrigger;
public Collider2D rightSlashTrigger;
bool slashing = false;
float slashTimer = 0;
float slashCooldown = 0.3f;
Animator animator;

private void Awake(){
   rightSlashTrigger.enabled = false;
   leftSlashTrigger.enabled = false;
}

void Start(){
   animator = GetComponent<Animator>();
}
```

```
void Update(){
    if (Input.GetButtonDown("Fire2") && !slashing){
        Slash();
    }

    if (slashing){
        ResetSlash();
    }
}

private void Slash(){
    slashing = true;
    slashTimer = slashCooldown;
    if (spriteRenderer.flipX == false)
        rightSlashTrigger.enabled = true;
    if (spriteRenderer.flipX == true)
        leftSlashTrigger.enabled = true;
    animator.SetInteger("Value", 4);
}

private void ResetSlash(){
    if (slashTimer > 0){
        slashTimer -= Time.deltaTime;
    }
    else{
        slashing = false;
        rightSlashTrigger.enabled = false;
        leftSlashTrigger.enabled = false;
    }
}
```

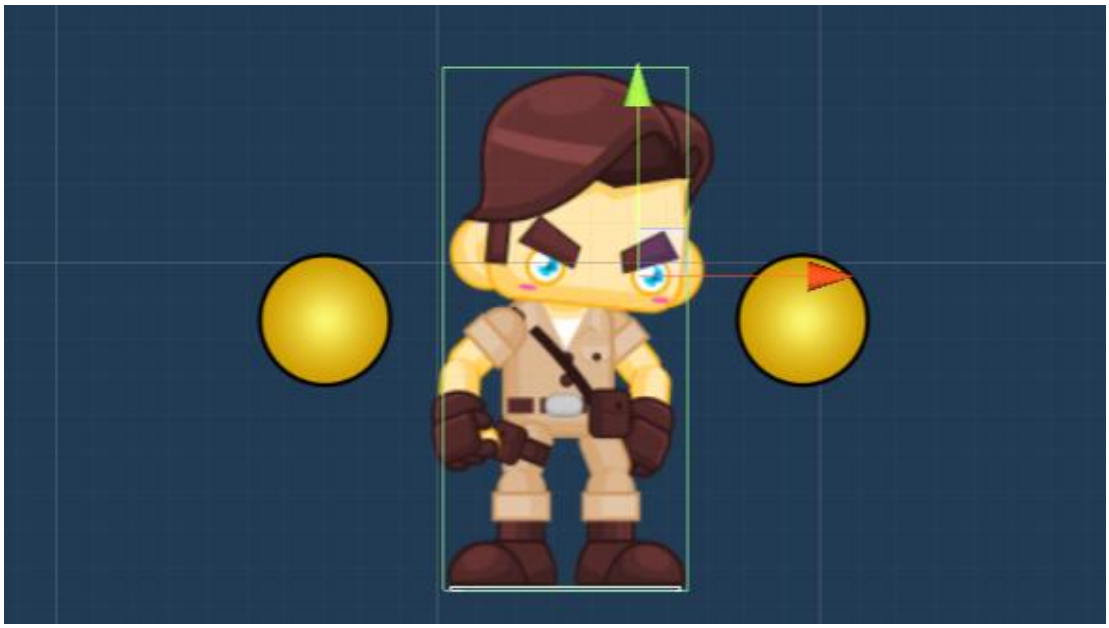
Deklariranje varijabli, instanciranje i inicijalno definiranje vrijednosti je već u ranijim primjerima bilo objašnjeno tako da će se odma krenuti na objašnjavanje glavnog dijela, a to su metode Slash() i ResetSlash() koje se pokreću unutar Update() metode ukoliko dođe do pritiske tipke ili gumba za napad.

U Slash() metodi vrijednost varijable „slashing“ postavlja se na istinitu, kako bi se ista varijabla kasnije mogla upotrijebiti za resetiranje napada. Druga linija koda nam dodjeljuje vremenski period za napad nakon prvotnog napada kako isti potez ne bi bio prekomjerno korišten. U ovom slučaju taj period je postavljen na 0.3 sekunde, no s povećanjem bi se igrica otežala. Dolazimo do dvije selekcije u kojima se provjerava strana na koju je glavni lik okrenut. Ukoliko je glavni lik okrenut na desnu stranu, aktivirat će se desni okvir napada, u suprotnom će se aktivirati lijevi okvir napada. Na kraju dolazimo do prikaza animacije napada bez koje ne bi bilo indikatora da je napad napravljen. [13]

ResetSlash() kao što samo ime kaže, nakon određenog vremenskog perioda isključuje „slash collider“ te na taj način vraća stvari na inicijalno zadane postavke. Na taj se način omogućuje beskonačan broj udaraca nožem koje korisnik može koristiti da si olakša put do cilja. [13]

5.2.3.3. Pucanje iz revolvera

Treći i zadnji napad je napad iz revolvera. Ovaj napad je poseban jer nije uvijek dostupan. Svaku mapu igrač kreće bez revolvera te ga može pokupiti ukoliko ga na mapi ima. Nakon što se revolver pokupi igraču je moguće korištenje streljnog oružja do kraja nivo-a ili do smrti. Ovo je najsloženiji od napada jer zahtijeva dvije vrste animacija, nove objekte i slično. Prva napravljena animacija je animacija pucanja glavnoga lika, za pucanje je također potreban metak koji će biti ispaljen. U ovom slučaju metak također ima animaciju te se ispaljuje na predefiniranom mjestu koje je dijete objekta glavnog lika. Na sljedećoj slici se mogu vidjeti te pozicije te se metak ispaljuje iz one pozicije ovisno na koju je glavni lik okrenut. Kada se metak ispali postoji mogućnost da se dogodi kolizija sa drugim objektom te se taj objekt uništava ukoliko je to protivnik, a ukoliko nije uništava se metak. Treća opcija je da je metak opaljen u prazno. Ukoliko je to slučaj, metak se automatski uništava nakon nekoliko sekundi zahvaljujući „DestroyWithDelay“ skripti. [21]



Slika 7. Kreiranje početnih pozicija za ispaljivanje metaka

U sljedećem dijelu nalazi se prikaz programskog koda iz tri skripte koji je omogućio funkcionalnost pucanja.

```
//DestroyWithDelay
public float delay;
void Start(){
    Destroy(gameObject, delay);
}

//BulletController
public Vector2 velocity;
Rigidbody2D rigidBody2D;

void Start(){
```

```
    rigidBody2D = GetComponent<Rigidbody2D>();
}

void Update(){
    rigidBody2D.velocity = velocity;
}

private void OnCollisionEnter2D(Collision2D collision){
    if (collision.gameObject.CompareTag("Enemy")){
     GameController.instance.BulletHitEnemy(collision.gameObject.transform);
        Destroy(gameObject);
    } else if (!collision.gameObject.CompareTag("Player")){
        Destroy(gameObject);
    }
}
//PlayerController
public float timeBetweenShots;
public bool hasRevolver;
public Transform leftBulletSpawn;
public Transform rightBulletSpawn;
public Image imageRevolver;
public GameObject leftBullet;
public GameObject rightBullet;
bool canShot = true;

void Update(){
    if (Input.GetButtonDown("Fire1")){
        FireBullets();
    }
}

private void FireBullets(){
    if (hasRevolver){
        if (!isRunning && canShot){
            if (spriteRenderer.flipX){
                Instantiate(leftBullet, leftBulletSpawn.position,
Quaternion.identity);
            }
            if (!spriteRenderer.flipX){
                Instantiate(rightBullet, rightBulletSpawn.position,
Quaternion.identity);
            }
AudioController.instance.FireBullets(gameObject.transform.position);
            canShot = false;
            animator.SetInteger("Value", 3);
            StartCoroutine(ShootDelay());
        }
    }
}

IEnumerator ShootDelay(){
    yield return new WaitForSeconds(timeBetweenShots);
    canShot = true;
}

//GameController
public void BulletHitEnemy(Transform enemy){
    Vector3 pos = enemy.position;
    pos.z = 20f;
    ParticlesController.instance.ShowEnemyParticle(pos);
    Destroy(enemy.gameObject);
```

```
    AudioController.instance.EnemyExplosion(pos);
}
```

U gornjem dijelu vidimo veći dio koda koji će sada biti objašnjen. Prva skripta je poprilično jednostavna te se radi o skripti koja uništava objekt nakon određenog vremenskog perioda što je realizirano kroz metodu Destroy() [30]. Napravljena je u svrhu uništavanja metaka nakon određenog broja sekundi, no kasnije je korištena i u druge svrhe.

Drugi dio koda prikazuje "BulletController" skriptu te je dosta jednostavna. Nakon deklariranja i instanciranja objekta u Update() metodi mu je konstatno omogućeno kretanje u smjeru koji je zadan kroz Unity sučelje preko varijable "velocity" koja je vidljiva svima (eng. *Public*). Vrijednost je postavljena na x:6 i y:0 što metku konstantno omogućuje kretanje u istom pravcu. Daljnji dio koda provjerava da li se dogodila kolizija metka sa drugim objektima omogućeno kroz OnCollisionEnter2D [28] te se za realizaciju usporedbe koristi gotova metoda CompareTag() [37]. U slučaju da se dogodila događaju se određene akcije. Ukoliko je metak dodirnuo objekt sa tagom "Enemy" poziva se metoda BulletHitEnemy() i metak se uništava. Zahvljujući metodi koja je pozvana te se nalazi u "GameController" skripti, protivnik se također uništava te se odvijaju dodane radnje. [21]

Treći dio koda je taj koji omogućuje pokretanje metka iz početne pozicije te vizualni prikaz i animaciju. Radi se o dijelu koda "PlayerController" skripte gdje je najbitnija metoda FireBullets() u kojoj se prvo provjerava istinitost "hasRevolver" varijable što je istinito jedino u slučaju kad je igrač pokupio revolver na određenoj mapi. Kada igrač ima revolver, omogućeno mu je pucanje u slučaju kada ne trči. Nadalje, imamo dva slučaja u kojima lik može biti okrenut na lijevu ili desnu stranu, ovisno o tome stvara se metak sa metodom Instantiate() [29] gdje stvaramo predefinirani metak na određenoj lokaciji. Nakon toga, pokreće se zvuk pucanja te se sa ShootDelay() metodom pokrenutom preko StartCoroutine() [27] onemogućava pucanje za određeni vremenski period. [21]

Zadnji dio koda napisan je u „GameController" skripti i omogućava uništenje protivnika nakon kolizije sa metkom te stvaranje djelovanja čestica (eng. *Particle effect*). Ovaj dio također omogućava proizvodnju zvuka kroz „AudioController" skriptu koja će biti objašnjena u lekciji vezanoj uz zvučne efekte. [21]

5.3. Izrada protivnika

Izrada protivnika, kao i izrada glavnog lika, može se podijeliti u dvije kategorije. Prva kategorija je izrada animacija za protivnika te druga, skriptiranje. U prvom poglavlju ukratko je opisana izrada animacije, no nije joj posvećena prevelika pažnja jer je animiranje već detaljno objašnjeno u prijašnjim dijelovima te su animacije izgrađene na identičan način. Nadalje slijedi glavni dio izrade protivnika, odnosno izrada umjetne inteligencije (eng. *Artificial Intelligenc*e,

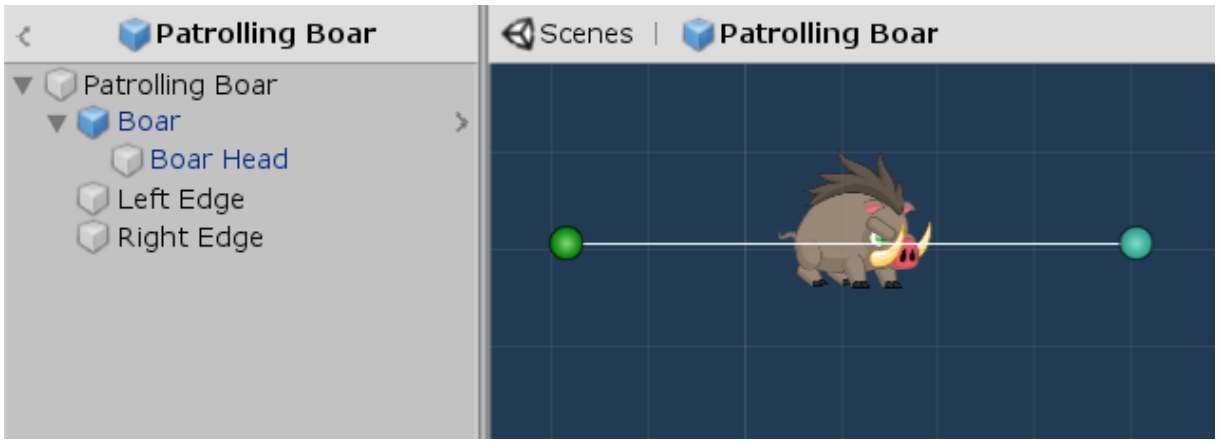
skraćeno AI) koja pokreće protivnike, omogućava im patroliranje te ono glavno, nanošenje štete i oduzimanje života glavnom liku.

5.3.1. Animiranje protivnika

Animiranje protivnika postignuto je kroz korištenje gotovih „Sprite-ova" kroz Unity Animator. Protivnici zahtjevaju dosta manje animacija od glavnog lika jer se njima ne upravlja. U ovom slučaju protivnici imaju jednu animaciju i to animaciju hodanja jer su napravljeni tako da izvode patroliranje od jedne do druge točke. U ovoj igrici postoji šest vrsta jednostavnih protivnika koje su u pravilu životinje, stoga je bilo potrebno napraviti animacije hodanja za šest vrsta životinja.

5.3.2. Skriptiranje protivnika

Prije početka samog skriptiranja, bilo je potrebno napraviti određene objekte koji su djeca glavnog objekta. U ovom slučaju imamo glavni objekt koji sadrži izgrađen „prefab“ protivnika. Na taj način dobivamo protivnika koji se kreće od jednog objekta do drugoga i protivnika koji može patrolirati od određene do određene točke.



Slika 8. Prefab patrolirajućeg protivnika

Ovo poglavlje sadržava opis izrade protivnika na jednom primjeru jer je izrada za ostale protivnika veoma slična. U sljedećem poglavlju date su male razlike između protivnika na kojih treba obratiti pažnju. Kao što se na slici može vidjeti, objekti patrolirajuće divlje svinje su sama divlja svinja i njena glava te lijevi i desni rub. Kao što je već ranije objašnjeno, napravljena je „EnemyHeadController“ skripta koja je pričvršćena (eng. *Attach*) za protivnikovu glavu te omogućava skakanje na protivnika. Druge dvije skripte su „EnemyMovementController“ te „EnemyPatrolController“ koje sadrže zajedničke elemente, no razlikuju se u dijelovima preokretanja protivnika po x-osi. U jednoj skripti, protivnik se kreće dok ne naleti na drugi objekt koji ima „collider“ te se okreće u drugu stranu. U drugoj skripti se unaprijed definiraju lijevi i

desni rub od kojeg do kojeg će se protivnik kretati te je na taj način omogućeno patroliranje. [21]

```
//EnemyMovementController
public float speed;
Rigidbody2D rigidBody2D;
SpriteRenderer spriteRenderer;

void Start(){
   rigidBody2D = GetComponent<Rigidbody2D>();
   spriteRenderer = GetComponent<SpriteRenderer>();
   IndicateInitialDirection();
}

void Update(){
   MoveHorizontally();
}

void IndicateInitialDirection(){
   if (speed > 0)
      spriteRenderer.flipX = false;
   else if (speed < 0)
      spriteRenderer.flipX = true;
}

void FlipOnCollision(){
   speed = -speed;
   IndicateInitialDirection();
}

void MoveHorizontally(){
   Vector2 temp = rigidBody2D.velocity;
   temp.x = speed;
   rigidBody2D.velocity = temp;
}

private void OnCollisionEnter2D(Collision2D collision){
   if (!collision.gameObject.CompareTag("Player"))
      FlipOnCollision();
}
```

U Start() metodi poziva se metoda IndicateInitialDirection() u kojoj se određuje početni smjer. Početni smjer, kao što smo već ranije imali, određuje se kroz predznak varijable za brzinu. Ukoliko je predznak pozitivan, smjer kretnje je prema desnoj strani te ukoliko je negativan, smjer je obrnut. Ovisno o tome u kojoj strani se protivnik kreće, njegova slika se također okreće prema toj strani. U ovoj metodi to je postignuto pomoću „FlipX“ komponente koja omogućava okretanje po x-osi. Nadalje, svaku sličicu (eng. *Frame*) poziva se MoveHorizontally() metoda koja kroz pomoćnu varijablu kreira novi vektor te mu dodjeljuje brzinu kretnje danu kroz Unity sučelje. Varijabli brzine moguće je pristupiti kroz Unity sučelje jer je ona vidljiva svima (eng. *Public*) što je bio slučaj i u ranijim primjerima. U ovoj skripti fora je u tome da se protivnik okreće kada dodirne drugi objekt koji ima „collider“. To je omogućeno

kroz metodu FlipOnCollision() koja se poziva svaki puta kada protivnik dodirne objekt koji nije tagiran sa imenom „Player“. Okret je realiziran jednostavnom linijom koda koja mjenja predznak brzine protivnika te na taj način, mjenja njegov smjer kretanja. [21]

```
//EnemyPatrolController
public Transform leftEdge;
public Transform rightEdge;

void FlipOnEdges(){
   if (!spriteRenderer.flipX && transform.position.x >=
rightEdge.position.x){
      if (canTurn){
         originalSpeed = speed;
         speed = 0;
         spriteRenderer.flipX = true;
         speed = -originalSpeed;
      }
   } else if (spriteRenderer.flipX && transform.position.x <=
leftEdge.position.x){
      if (canTurn){
         originalSpeed = speed;
         speed = 0;
         spriteRenderer.flipX = false;
         speed = -originalSpeed;
      }
   }
}

private void OnDrawGizmos(){
   Gizmos.DrawLine(leftEdge.position, rightEdge.position);
}
```

Gornji dio programskog koda dio je skripte za patroliranje protivnika te sadrži jednake metode kao i skripta za normalne kretnje protivnika. Razlika između ove dvije skripte je u FlipOnEdges() metodi. Naime, ova klasa sadrži dvije varijable „leftEdge“ i „rightEdge“ koje su vidljive svima te su tipa „Transform“. Cilj je da se na te dvije varijable “prenese” objekt koji označava lijevi rub te objekt koji označava desni rub kako bi se mogao nastavit daljnji rad. Uvjeti u ovom dijelu koda su veoma jednostavni te su istiniti ako se protivnik kreće u desnu stranu te je njegova pozicija na x-osi veća od pozicije desnog ruba te obrnuto za lijevu stranu. Unutarnji dio selekcija smo već imali ranije te realizira zaustavljanje protivnika, okret i promjenu smjera. Posljednji dio koda služi za vizualni prikaz horizontalne crte omogućen preko DrawLine() metode [38] koja omogućava bolji prikaz u načinu rada razvijača (eng. *Developer mode*). [21]

5.3.3. Razlike između protivnika

Dosad je objašnjeno kreiranje jednog protivnika, no igrica sadrži šest protivnika koji se razlikuju po nekim komponentama. Naime, protivnici su orao, pčela, divlja svinja, majmun, jež i gljiva te svaki od njih ima jednu stvar koja ju obilježava. Kao što se može zaključiti, orao i

pčela su leteći protivnici te je za njihovo ostvaranje bilo potrebno isključiti gravitacijsku silu kako ne bi padali do sljedećeg “collider-a”. Ostali protivnici se kreću po tlu te svaki protivnik ima različitu brzinu što igrača nekad može zeznuti i otežati mu neke situacije. Svaki protivnik se razlikuje po svojoj veličini gdje je najbolji primjer gljiva koja je dovoljno niska i mala da ju igrač svojim pucanjem ne može uništiti te majmun koji je dovoljno visok da protivniku otežа skok na glavu. Nadalje, jež svojim bodljama onemogućuje igraču skok na glavu jer to rezultira smrti glavnoga lika. Što se tiče letećih protivnika, postoje orao i pčela te se razlikuju jedino po brzini. Pčela je jako brza i ima mali “hitbox” što nekad igrača dovodi u situaciju da ju jednostavno zaobiđe i izbjegne rizik gubljenja života dok je orao spor i lagan za svladavanje. Pravljenje većeg broja protivnika te izgradnja sitnih razlika između pojedinog napravljena je kako igra ne bi bila monotona uz jednu te istu prijetnju. Uvođenje većeg broja protivnika definitivno poboljšava i uljepšava igru te poboljšava korisničko iskustvo.

Slika 9. Različite vrste protivnika

5.4. Izrada voća

Izrada voća jedan je od glavnih aspekata realizacije korisničkog rezultata. Na temelju ubijenih protivnika, ali najvećim dijelom prikupljenih „collectibles-a“ kreira se rezultat korisnika koji se poništava gubljenjem tri života. U ovoj igrici postoje četiri vrste voća, ali i revolver koji se broji pod „collectible“ i kupljenjem omogućava pucanje. Što se tiće voća, postoje jabuke, višnje i jagode koje prikupljanjem donose različit rezultat igraču. Četvrti „collectible“ je zlatna breska koja, prilikom kupljenja, završava nivo. Stoga se može reći da je cilj igrice prikupiti sve zlatne breskve i imati što veći rezultat prilikom završetka. Kao i svaka dosadašnja stvar, tako i ova podjeljena je u izradu animacija i skriptiranje. Animacija ovoga puta nije ista kao i prije te je realizirana promjenom veličine jedne sličice te je tako postignuta rotacija po x-osi.

5.4.1. Animiranje voća

Animiranje voća postignuto je na način da je sa jednim jedinim „Sprite-om“ kreirana animacija promjenom „scale“ komponente. Periodično je mjenjan „scale“ i korištenjem petlje

kreirano je beskonačno okretanja objekta. Promjena „scale“ komponente se mjenja na skali od 2 do 0.2 zatim ponovno od 0.2 do 2 što daje isti efekt. Na sljedećoj slici vizualno je prikazano kako to izgleda u Unity-ju. Ovo je primjer animacije za jabuku te je isti način korišten za ostale „collectibles“ objekte kao što su višnja, jagoda, zlatna breska te revolver. [21]

Slika 10. Animiranje voća

5.4.2. Skriptiranje voća

Skriptiranje voća realizirano je kroz par metodi u različitim skriptama koje korisniku omogućavaju kupljenje objekta, uništenje istog te povećanje rezultata nakon skupljanja.

```
//FruitController
private void OnTriggerEnter2D(Collider2D collision){
    if (collision.gameObject.CompareTag("Player")){
        Destroy(gameObject);
    }
}

//PlayerController
private void OnTriggerEnter2D(Collider2D collision){
    switch (collision.gameObject.tag){
        case "Strawberry":
ParticlesController.instance.ShowRedParticle(collision.gameObject.transform
.position);
            GameController.instance.UpdateStrawberries();

AudioController.instance.FruitPickUp(gameObject.transform.position);
            break;
        case "Apple":
ParticlesController.instance.ShowRedParticle(collision.gameObject.transform
.position);
            GameController.instance.UpdateApples();

AudioController.instance.FruitPickUp(gameObject.transform.position);
            break;
        case "Cherry":
ParticlesController.instance.ShowRedParticle(collision.gameObject.transform
.position);
            GameController.instance.UpdateCherries();

AudioController.instance.FruitPickUp(gameObject.transform.position);
            break;
        case "Peach":
```

```
ParticlesController.instance.ShowGoldenParticle(collision.gameObject.transf
orm.position);
         GameController.instance.UpdatePeaches();

AudioController.instance.FruitPickUp(gameObject.transform.position);
         Invoke("ShowLevelCompletedMenu", 1f);
         break;
       default:
          break;
   }
}

//GameController
public void UpdateCherries(){
   data.cherryCount += 1;
   txtCherryCount.text = " x " + data.cherryCount;
   UpdateScore(cherryValue);
}

public void UpdateApples(){
   data.appleCount += 1;
   txtAppleCount.text = " x " + data.appleCount;
   UpdateScore(appleValue);
}

public void UpdateStrawberries(){
   data.strawberryCount += 1;
   txtStrawberryCount.text = " x " + data.strawberryCount;
   UpdateScore(strawberryValue);
}

public void UpdatePeaches(){
   data.peachCount += 1;
   txtPeachCount.text = " x " + data.peachCount;

   UpdateScore(peachValue);
}

public void UpdateScore(int value){
   data.score += value;
   txtScore.text = "SCORE: " + data.score;
}
```

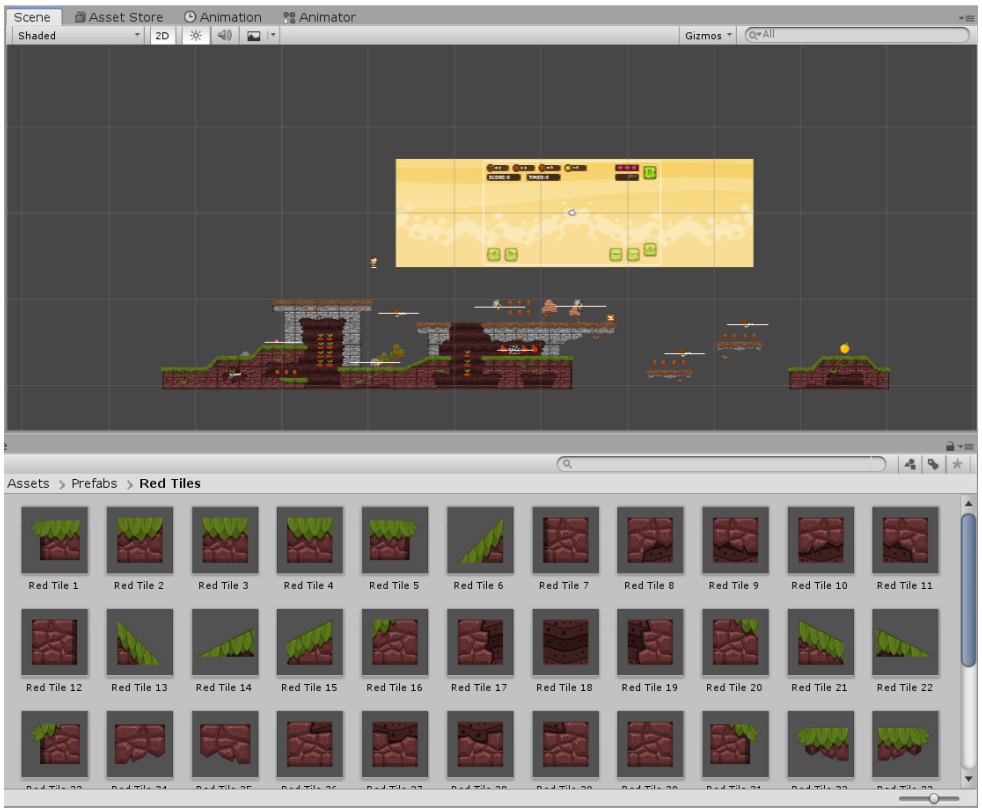
Ovaj dio podjeljen je u tri skripte od kojih je svaka zadužena za jednu funkcionalnost. Prva skripta je najjednostavnija te se pričvršćuje na objekte voća i pri koliziji sa glavnim likom realizira uništenjenje istog objekta. Dio koji se nalazi u „PlayerController“ skripti na prvi pogled izgleda velik i kompliciran, a zapravo je dosta jednostavan. Kroz “switch-case” napravljeni su slučajevi kolizije sa različitim oblicima voća. Pri koliziji sa jednom od vrsta voća pokreću se određene metode koje realiziraju djelovanje čestica, pokretanje zvuka te ažuriranje trenutnog rezultata. Razlika je zlatna breskva koja također poziva i ShowLevelCompletedMenu() što završava nivo i otvara dio grafičkog korisničkog sučelja za odabir daljnjih akcija. Metode iz GameController() skripte se razlikuju jedino po tome što se svaka poziva za različitu vrstu voća, a zapravo imaju istu ulogu. Njihova uloga je povećanje broja voća u bazi podataka te

ažuriranje prikaza na ekranu igre (eng. *Heads Up Display*, skraćeno HUD) kao i povećanje ukupnog rezultata (eng. *Score*) što je realizirano kroz UpdateScore() metodu. [21]

Slika 11. Različite vrste voća

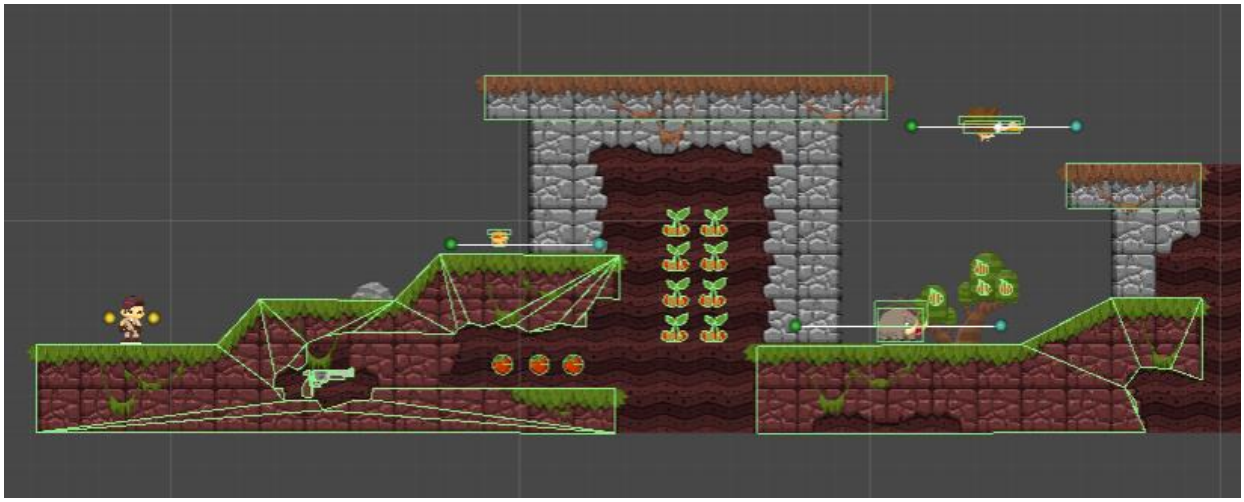
5.5. Izrada mapa

Za izradu mapa korišten je gotov „tileset“ crvenih i sivih blokova. Za ovu igricu izrađeni su deset kratkih mapa koje služe za demonstraciju u kojima korisnik može isprobati mogućnosti igre. Naravno, igra je nadogradiva i bilo kad se mogu dodati nove mape ili proširenja trenutnih. Izrada ovih mapa nije bila jednostavan proces te se sastojala od kreiranja „prefab-a“ i slaganja istih kako u hijerarhiji, tako i na sceni. Na sljedećoj slici može se vidjeti cjelokupan prikaz jedne mape i određenih blokova od kojih je mapa izgrađenja.



Slika 12. Prikaz izrade mape

Sama izrada mapa nije bila dovoljna jer bi glavni lik kroz nju propadao. Jako je bitno gotovim mapama pridružiti određeni „collider“ koji obilježava tlo i po kojemu se može hodati. Za potrebe kreiranja svijetova koriste su razni „collider-i“ od kojih su najviše korišteni „Box Collider 2D“ i „Polygon Collider 2D“. „Polygon Collider 2D“ je korišten iz razloga neravnog terena te je namjenjen za dodjeljivanje „collider-a“ n-terokutima koji mogu biti i nepravilnih oblika što mu daje veliku moć. „Box Collider 2D“ korišten je za jednostavne kockaste platforme ili ravno tlo. Na sljedećoj slici moguće je vidjeti korištenje istih „collider-a“ i njihova primjena.



Slika 13. Korištenje "Collider-a" pri izradi mape

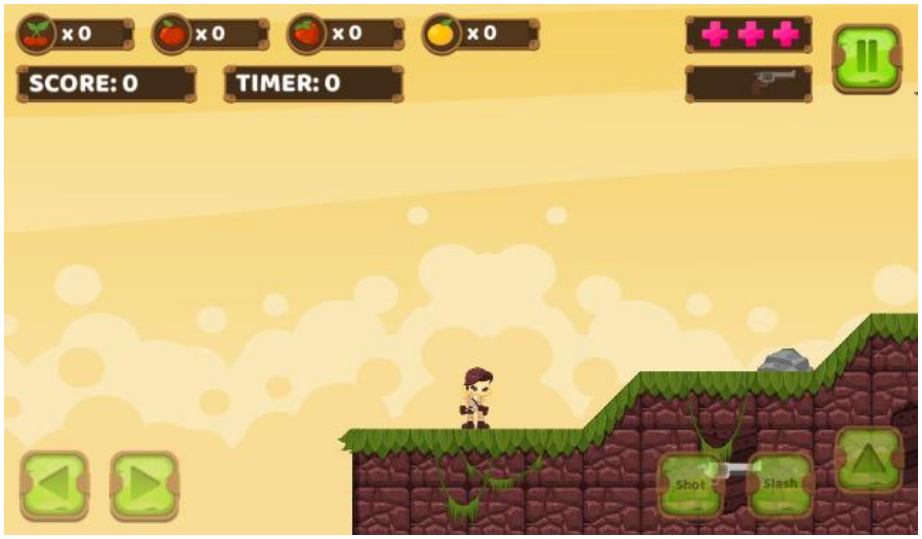
5.6. Izrada grafičkog korisničkog sučelja

Izrada grafičkog korisničkog sučelja bitan je element svake igrice. Grafičko korisničko sučelje (eng. *Graphical user interface*, skraćeno GUI) sustav je interaktivnih vizualnih komponenti računalnog softvera. Služi za prikazivanje objekata koji prenose informacije s kojima korisnik može komunicirati, odnosno predstavljaju radnje koje korisnik može poduzeti. Objekti mogu mjenjati boju, veličinu ili vidljivost kada korisnik poduzme jednu od radnji. To su najčešće ikone, pokazivači ili gumbi koji ponekad znaju biti poboljšani zvukom ili vizualnim efektima. [14]

U ovoj igrici izgrađeno je više interaktivnih menu-a s kojima korisnik može poduzimati radnje. Prvi i najbitniji je glavni izbornik (eng. *Main menu*) koji je jednostavan te prikazuje pokretanje ili izlazak iz igrice. Pokretanjem igre dolazimo do izbornika mapa (eng. *Level selection menu*) s kojima korisnik može odabrati koja će se mapa sljedeća pokrenuti. Nakon odabira mape dolazimo do pokretanje igre gdje možemo vidjeti razne stvari. Dok je igra pokrenuta možemo vidjeti mobilno korisničko sučelje (eng. *Mobile user interface*, skraćeno Mobile UI) koje omogućava pokretanje glavnog lika na mobilnom uređaju. Iznad igrača može se vidjeti prikaz rezultata, života i prikupljenih stvari (eng. *Heads up display*, skraćeno HUD) te gumb koji zaustavlja igru i tako pokreće izbornik pauze (eng. *Pause menu*). Kroz isti izbornik možemo doći na izbornik odabira levela, ponovno pokreniti mapu ili uključiti/isključiti zvuk. Sljedeći izbornik je izbornik završetka levela (eng. *Level completed menu*) koji se pokreće prilikom završetka mape, odnosno nakon što je pokupljen „collectible“ koji završava nivo. Kroz njega možemo pokrenuti sljedeću mapu ili se vratiti na izbornik odabira nivo-a. Zadnji, manje bitan, izbornik završene igre (eng. *Game over menu*) koji se prikazuje kada korisnik izgubi sve živote. Na njemu igrač ima mogućnost ponovnog pokretanja mape ili povratka na izbornik odabira mapa. [21]

Slika 14. Glavni izbornik

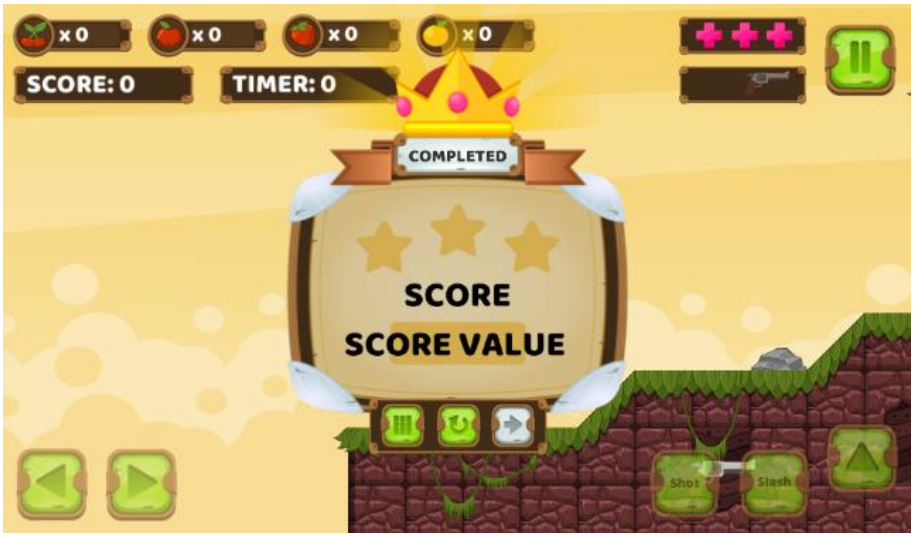
Slika 15. Izbornik odabira mapa



Slika 16. Mobile UI i HUD

Slika 17. Izbornik pauze

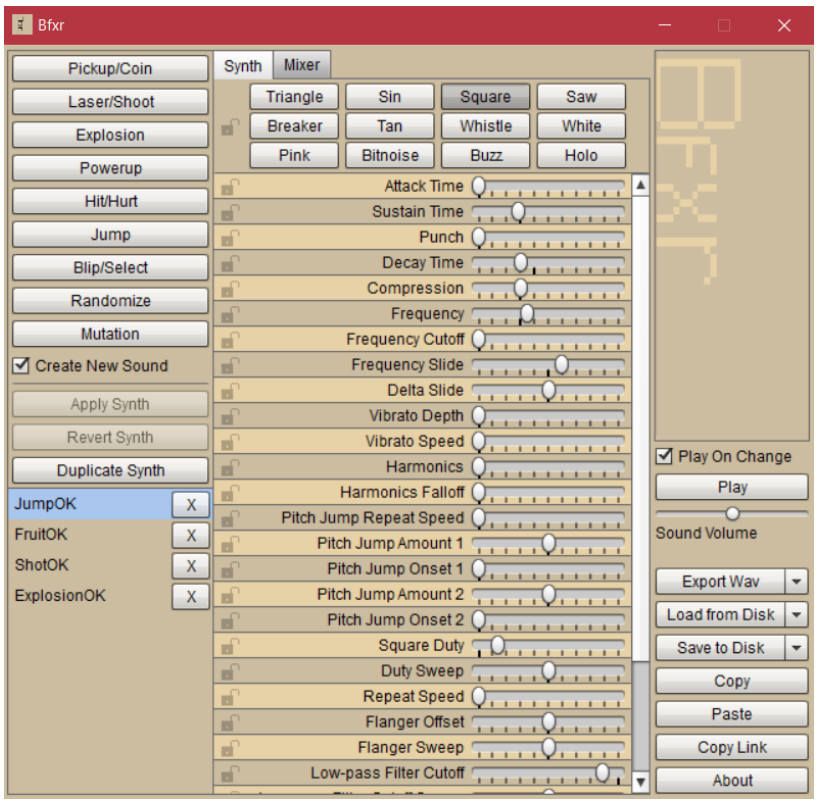
Slika 18. Izbornik završene igre



Slika 19. Izbornik završene mape

5.7. Izrada zvučnik efekata

Zvučni efekti (eng. *Sound effects*) upotpunjavaju gotov proizvod te korisnicima daju bolju zabavu prilikom igranja igrice. Zvučni efekti se danas mogu pronaći na raznim stranicima te su besplatni za komercijalnu upotrebu. U ovoj igrici to nije slučaj jer su zvučni efekti izgrađeni preko alata za izradu retro zvučnih efekata. Korišten alat naziva se BFXR te je jednostavan alat za korištenje. [15]



Slika 20. Izrada zvučnih efekata u alatu "BFXR"

Za zvučne efekte koristi se jednostavna skripta „AudioController“ u kojoj se nalaze glavne metode koje se pozivaju na određenu radnju. Glavne metode za izradu ove funkcionalnosti nalaze se u sljedećem programskog kodu.

```
//AudioController
public void PlayerJump(Vector3 playerPos){
    if (soundOn){
        AudioSource.PlayClipAtPoint(audioData.playerJump, playerPos);
    }
}

public void FruitPickUp(Vector3 playerPos){
    if (soundOn){
        AudioSource.PlayClipAtPoint(audioData.fruitPickUp, playerPos);
    }
```

```
}

public void FireBullets(Vector3 playerPos){
    if (soundOn){
        AudioSource.PlayClipAtPoint(audioData.fireBullets, playerPos);
    }
}

public void EnemyExplosion(Vector3 playerPos){
    if (soundOn){
        AudioSource.PlayClipAtPoint(audioData.enemyExplosion, playerPos);
    }
}
```

Programski kod je jednostavan te je bitna stvar pozicija glavnog lika što znači da se zvučni efekt kreira na poziciji na kojoj se glavni lik nalazi kako bi dobili efekt zvuka kao u stvarnome svijetu. Druga stvar je selekcija u kojoj se provjerava da li je zvuk upaljen što se može kontrolirati kroz izbornik pauze. Zadnja stvar je pokretanje zvuka što se ostvaruje kroz PlayClipAtPoint() [31] gotovom metodom. [21]

5.8. Izrada djelovanja čestica (eng. Particle effect)

Efekti čestica jedinstven su alat koji našim interakcijama može dodati interaktivnost i reaktivnost. U ovom primjeru također je obrađno djelovanje čestica na nekim akcijama kao što je efekt pokupljenog voća, uništenja protivnika, skoka na tlo i slično. Ovi efekti izgrađeni su u alatu Unity koji nudi jako veliku mogućnost izgradnje klikom na „Effects“ zatim „Particle System“. Nakon što je kreiran objekt, klikom na isti otvara se velika mogućnost izgradnje djelovanja čestica u kojima se može postaviti vlastiti „sprite“, promjeniti veličina, životni vijek, boja, oblik i mnoge druge komponente. [16]

```
//ParticlesController
public void ShowRedParticle(Vector3 position){
    Instantiate(redFruitParticle, position, Quaternion.identity);
}

public void ShowGoldenParticle(Vector3 position){
    Instantiate(goldenFruitParticle, position, Quaternion.identity);
}
```

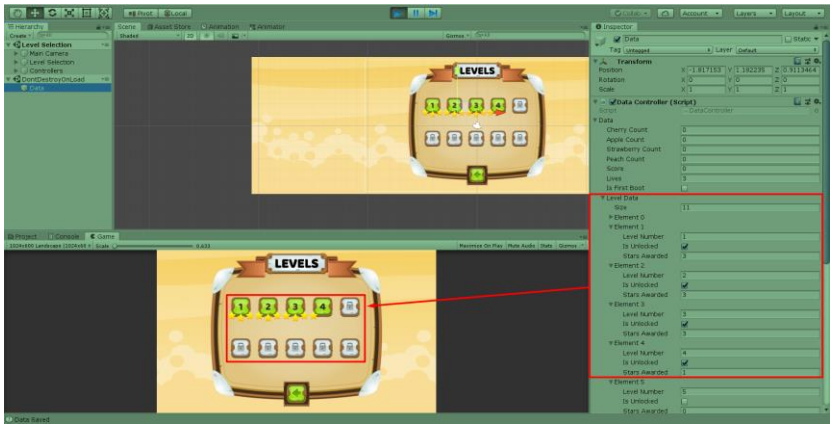
Ovaj dio programskog koda pokazuje instanciranje djelovanja čestica koji su ranije napravljeni kao “prefab”. Kod prikazuje primjer kreiranja djelovanja čestica za sakupljanje crvenog i zlatnog voća što je samo jedan od primjera korištenih u izradi igre. Ove metode pozivaju se kod kolizije igrača sa voćem. Prikaz efekta čestica realiziran je kroz Instantiate() metodu [29] u kojoj koristimo parametre za sam efekt, njegovu poziciju te „Quaternion.identity“

[23] što znači da nema rotacija objekta. Na isti način realizirani su ostali efekti te ih je u bilo kojem trenutku moguće nadograditi ili nadodati nove. [21]

5.9. Korištenje binarne datoteke (baza podataka)

Igrice kao što je ova ne zahtijevaju puno podataka stoga nije praktično izgrađivati cijelu bazu ako to nije potrebno. Kao baza podataka u ovom slučaju korištena je binarna datoteka u koju su se spremili podaci kao što su broj pokupljenog voća, igračev rezultat, životi, polje svih mapa te postavke muzike (upaljeno/ugašeno). Svaki element polja također sadrži podatke kao što su broj mape, podaci o tome da li je otključana ili zaključana te broj zarađenih zvjezdica. Na temelju ovih podataka prilikom pokretanja scene glavnog izbornika kreira se „singleton“ objekt „Data“ koji sadrži iste podatke. Ovaj objekt je poseban te se razlikuje od drugih jer je proteže kroz sve scene. Nakon kreiranja korištenjem metode DontDestroyOnLoad() [39] omogućeno je da se ne uništava te na taj način uvijek možemo koristiti iste podatke neovisno o sceni u kojoj se nalazimo. [21]

Slika 21. Spremanje podataka - 1. dio



Slika 22. Spremanje podataka - 2. dio

U sljedećem dijelu koda možemo vidjeti strukturu datoteke u koju spremamo podatke te što ona sadrži. Ovaj dio koda sadrži podatke navedene u prijašnjem odlomku preko kojih se omogućava pohrana te pristup istima. [21]

```
//GameData
public class GameData{
   public int cherryCount;
   public int appleCount;
   public int strawberryCount;
   public int peachCount;
   public int score;
   public int lives;
   public bool isFirstBoot;
   public LevelData[] levelData;
   public bool playSound;
   public bool playMusic;
}

//LevelData
public class LevelData{
   public int levelNumber;
   public bool isUnlocked;
   public int starsAwarded;
}
```

Sljedeći dio prikazuje esencijalni dio programskog koda koji omogućuje rad sa datotekom. Omogućava način čitanja i spremanja podataka u istu na način pozivanja metoda u određenom trenutku. Kao primjer na slikama dana je mogućnost mjenjanja podataka u razvijačkom modu (eng. *Developer mode*) te spremanja istih podataka pritiskom na gumb spremi koji se po potrebi može otključati. Za potrebe rada je također dodan ispis u konzolu realiziran sa Log() metodom [40]. Gumb spremi je ništa drugo nego poziv SaveData() metode bez parametara te se na taj način izmjenjeni podaci pohranjuju u datoteku. Drugi primjer je izmjena podataka prilikom nekog događaja što je prikazano dijelom programskog koda iz „GameController“ skripte. Gubljenjem sva tri života korisničke rezultate potrebno je postaviti na nula. U sljedećem dijelu koda vidimo tu radnju te se prilikom gubljenja svih života neke varijable od „GameData“ postavljaju na 0 te se poziva SaveData() metoda te se na taj način omogućava ažuriranje postojećih elemenata. Što se tiče RefreshData() metode pozivamo ju u Start() metodi „GameController“ skripte te na taj način omogućavamo čitanje podataka i postavljanje istih na scenu. [21]

```
//DataController
private void Awake(){
   if(instance == null){
      instance = this;
      DontDestroyOnLoad(gameObject);
   }else{
      Destroy(gameObject);
```

```
    }
    binaryFormatter = new BinaryFormatter();
    dataFilePath = Application.persistentDataPath + "/gameData.dat";
    Debug.Log(dataFilePath);
}

public void RefreshData(){
    if (File.Exists(dataFilePath)){
        FileStream fileStream = new FileStream(dataFilePath, FileMode.Open);
        data = (GameData)binaryFormatter.Deserialize(fileStream);
        fileStream.Close();
        Debug.Log("Data Refreshed");
    }
}

public void SaveData(GameData data){
    FileStream fileStream = new FileStream(dataFilePath, FileMode.Create);
    binaryFormatter.Serialize(fileStream, data);
    fileStream.Close();
    Debug.Log("Data Saved");
}

//GameController
if(data.lives == 0){
    data.lives = 3;
    data.score = 0;
    data.cherryCount = 0;
    data.appleCount = 0;
    data.strawberryCount = 0;
    data.peachCount = 0;
    DataController.instance.SaveData(data);
    Invoke("GameOver", restartDelay);
}
```

6. Android postavke

Kako bi igra bila igriva na Android uređaju potrebno ju je urediti te nadodati neke stvari koje će igru razlikovati od računalne. Napravljena igrica moguća je za pokretanje na računalu, no kako bi bila igriva na Android uređaju potrebno je dodati mobilno korisničko sučelje te binarnu datoteku napraviti tako da mobitel automatski prepozna o čemu se radi.

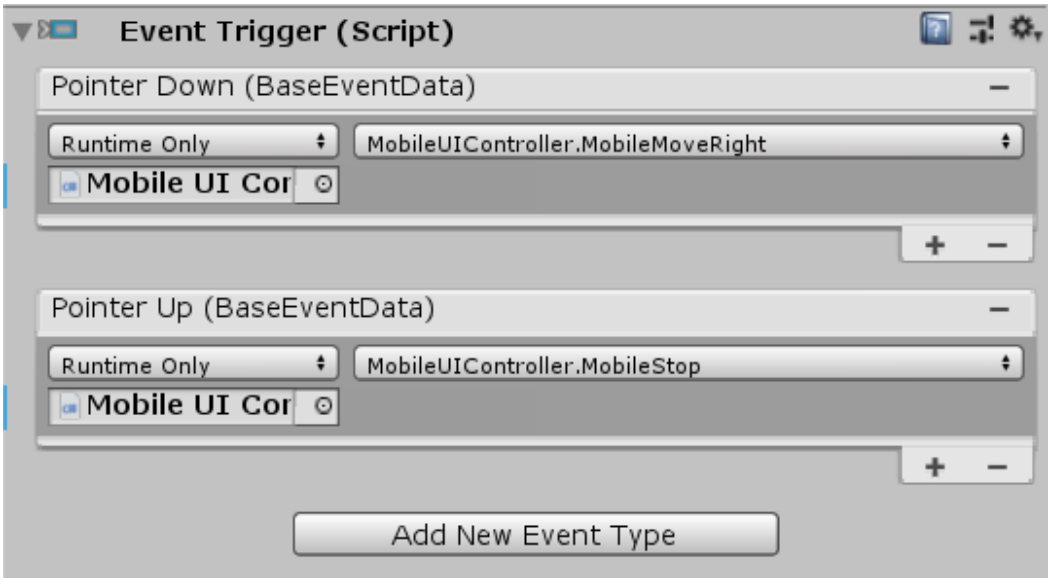
Prva stvar je mobilno korisničko sučelje koje je već ranije spominjano. Mobilno korisničko sučelje igraču omogućava kretnje i napade preko gumbova koji mogu biti aktivirani na dodir ekrana. Mobilno korisničko sučelje bitan je element svake mobilne igre te ga ima svaka igra u kojoj je potrebno koristiti više gumbova za kontrolu glavnog lika. Ova mogućnost realizirana je pomoću „Event Trigger-a“ koji je ugrađen u Unity. Na sljedećoj slici možemo vidjeti kako je realiziran pokret igrača u desnu stranu. Na Android uređaju potrebno je napraviti da se igrač kreće dok držimo pritisnut gumb za krenju u desno što je realizirano sa „Pointer Down“ i „Pointed Up“ pokretačima. Prilikom pritiska tipke pokreće se MobileMoveRight() metoda te MobileStop() prilikom otpuštanja iste. Ove metode su ništa drugo nego poziv napravljenih metoda za kretanje koje su objašnjene ranije. [21]

```
//PlayerController
public void MobileMoveRight(){
    rightPressed = true;
}

void Update(){
    if (leftPressed)
        Move(-speedBoost);
    if (rightPressed)
        Move(speedBoost);
}
```

Prikazani programski kod smo ranije već imali, no ovo je bolje mjesto za njegovo objašnjenje. Primjer prikazuje implementaciju za jednu od mobilnih tipki, a to je hod u desno. Prva, veoma jednostavna metoda je ta koja prilikom dodirivanja gumba na ekranu, mijenja vrijednost varijable u istinitu, što mijenja predznak brzine glavnoga lika te na taj način omogućuje kretanje lijevo ili desno. Prilikom otpuštanja tipke događa se sljedeće te igrač tada staje. [21]

```
//PlayerController
public void MobileStop(){
    leftPressed = false;
    rightPressed = false;
    Stop();
}
```

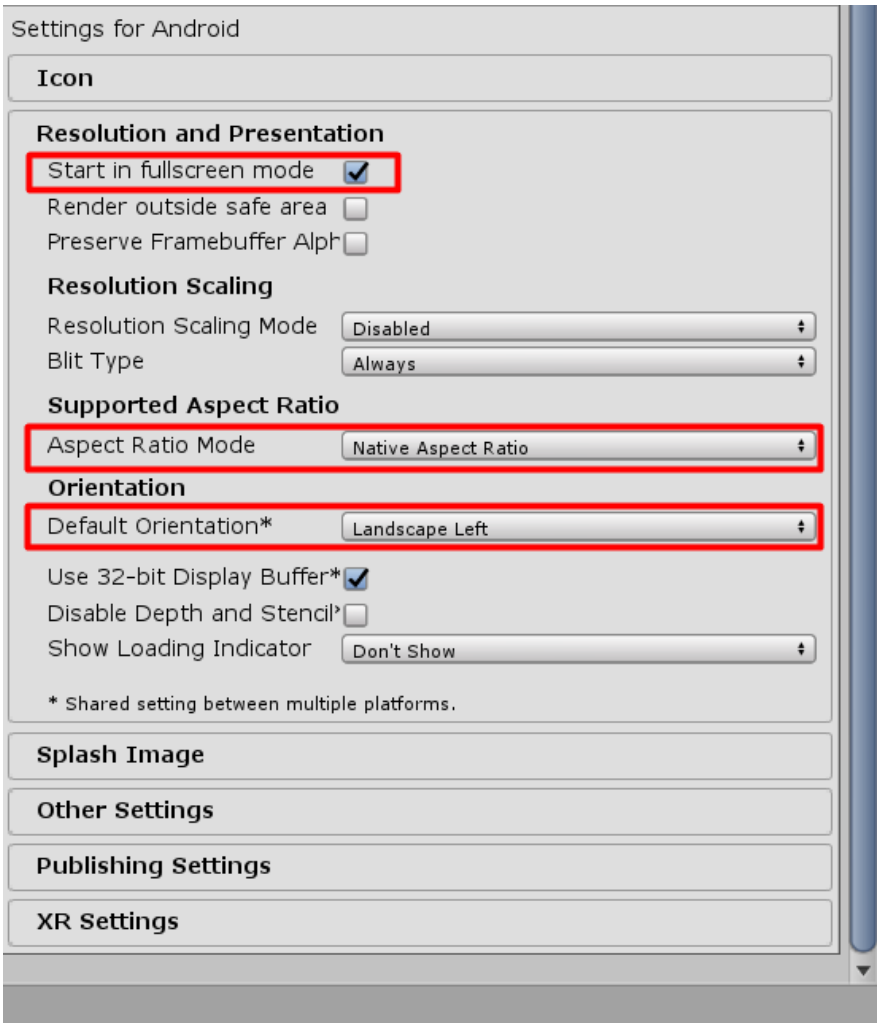


Slika 23. Izrada mobilnog korisničkog sučelja

Sljedeća bitna stavka prilikom kreiranja bila je prepoznavanje da li se igra pokreće na računalu ili mobilnom uređaju. Ova stvar rješena je na način tako da se binarna datoteka koja ima svrhu baze podataka pohranila u „Streaming Assets“ mapu. [17]

Zadnja stvar vrijedna spomena je izgradnja „builda“ za mobitel što je potrebno odraditi kako bi se odradila instalacija za pokretanje na mobilnom uređaju. Prije samog „builda“ potrebno je odretiti inicijalne postavke određene preko *File -> Build Settings -> Player Settings* gdje se može pronaći podnaslov „*Android Settings*“. Postavki ima puno, te su na sljedećim slikama prikazane samo neke od najosnovnijih obilježene crvenim okvirom. [21]

Slika 24. Android postavke - 1. dio



Slika 25. Android postavke - 2. dio

7. Publiciranje igre

Cilj izrade igrica može biti iz zabave, no kod većine razvojih timova ili individualnih osoba cilj je zaraditi novac. Zarada novca kroz Android igrice većinom se vrši kroz prikazivanje reklama ili dodjeljivanje nagrada za pogledanu reklamu. Kako bi imali što veći broj igrača igricu je također potrebno i objaviti javno. U ovom poglavlju govori se o monetizaciji i publiciranju igre što nije realizirano u programskom rješenju iz razloga što je ovaj primjer namjenjen za demonstraciju te bi se prije javne objave igru trebalo nadograditi.

Kako bi se napravila monetizacija, potrebno je napraviti skriptirani dio u kojim trenutcima će se pojavljivati reklame. Unity sadrži sustav za izradu reklami koje programeri mogu koristiti kao oblik zarade. Za ovakve stvari Unity nam pruža metodu Analytics.Transaction() koja služi za praćenje događaja transakcija unutar igrice. [18]

Nadalje, kada je igrica izgrađena potrebno ju je i publicirati. Igrica se može publicirati na razne načine jer danas imamo puno platoformi za igrice. Za računalne igrice primjer bi bio „Steam“ dok se za mobilne igrice najčešće koristi „Google App Store“. Unity sadrži integriran alat za publiciranje te se kroz određeni broj koraka mogu postaviti konfiguracije koje služe za pripremu. [19]

Nakon publiciranja, ukoliko programer ima velike ambicije, potrebno je oglasiti igricu. Danas postoje razni načini oglašavanja te se na taj način može postići popularnost i veći broj korisnika.

8. Zaključak

Od uvijek sam želio napraviti svoju igricu, ali najteže je napraviti prvi korak i krenuti od nule. Danas se na internetu može naći materijal za učenje bilo kojeg programskog jezika, ali opet su neke stvari teško shvatljive ukoliko se tek započinje. Korištenjem programskog jezika C# na kolegijima na fakultetu dobio sam ideju da isti jezik upotrijebim u ovu svrhu te da napravim svoju prvu igricu koristeći naučeno znanje. Uz znanje koje sam imao prije početka ovog rada bilo je potrebno mnogo više što sam naučio kroz određenu literaturu na internetu. Dok sam se sa Microsoft Visual Studijom koristio i ranije, Unity je bio sasvim novi pojam te je za svladavanje istog programa bilo potrebno određeno vrijeme. Iako je program jednostavan za korištenje, sadrži jako puno mogućnosti koje se jednostavno sve moraju isprobati da bi ih čovjek razumio. Odabran je baš ovaj žanr jer pokriva dosta elemenata 2D igrica te se izrada igrice za Android mobilni operacijski sustav može gledati samo kao nadogradnja na znanje vezano uz izgradnju Unity igrica za računalnu platformu. Nakon izrade finalnog programskog rješenja mogu reći da je dosta toga jasnije te bi svaki idući projekt bio bolji i napredniji. Isto tako, prije početka izrade prve igrice nisam znao šta me čeka niti sam mogao zamisliti kako će projekt napredovati, dok sad to mogu jer su kroz rad naučeni osnovni elementi koju bi većina igrica trebala imati. Nadam se da ću se i dalje nastaviti baviti Unity-jem i izradom igara jer svaka priča ima svoj početak i kraj, a ovo je tek početak. Nakon završetka studija htio bih se baviti radom u Unity-ju te se naći u razvojnom timu koji proizvodi profesionalne proizvode jer to je ono što volim i raditi taj posao mi ne bi bio problem, nego uživancija.

9. Popis literature

9.1. Literatura korištena za izvor informacija

[1] *„What is Unity? Everything you need to know“* [Na internetu] Dostupno: https://www.androidauthority.com/what-is-unity-1131558/ Pristupano: 14.08.2020.

[2] *„Unity“* [Na internetu] Dostupno: https://unity.com/ Pristupano: 15.08.2020.

[3] *„How Unity3D became a Game-Development beast“* [Na internetu] Dostupno: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ Pristupano: 15. 08.2020.

[4] *„Unity User Manual“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/index.html Pristupano: 15. 08.2020.

[5] *„Microsoft Visual Studio Library Support“* [Na internetu] Dostupno: https://devblogs.microsoft.com/cppblog/c99-library-support-in-visual-studio-2013/ Pristupano: 15. 08.2020.

[6] *„Visual Studio Tools for Unity“* [Na internetu] Dostupno: https://docs.microsoft.com/en-us/visualstudio/cross-platform/visual-studio-tools-for-unity?view=vs-2019 Pristupano: 15. 08.2020.

[7] *„10 Types of Platforms in Platform Video Games“* [Na internetu] Dostupno: https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games Pristupano: 17. 08.2020.

[8] *„The Evolution of Platform games in 9 steps“* [Na internetu] Dostupno: https://www.redbull.com/in-en/evolution-of-platformers Pristupano: 18. 08.2020.

[9] *„Royalty free 2D Game Assets“* [Na internetu] Dostupno: https://www.gameart2d.com/ Pristupano: 20.07.2020.

[10] *„Jungle Adventure: Game Sprites“* [Na internetu] Dostupno: https://www.gameart2d.com/character-spritesheet-7.html Pristupano: 20.7.2020.

[11] *„Jungle Platformer Game Tileset“* [Na internetu] Dostupno: https://www.gameart2d.com/platformer-game-tileset-6.html Pristupano: 20.7.2020.

[12] *„Free Fantasy Game GUI“* [Na internetu] Dostupno: https://www.gameart2d.com/free-fantasy-game-gui.html Pristupano: 20.7.2020.

[13] *„Unity 5 2D Platformer Tutorial – Part 20 – Melee Attacking“* [Na internetu] Dostupno: https://www.youtube.com/watch?v=xwPahXLpNh8 Pristupano: 18.08.2020

[14] *„Computer Hope: GUI“* [Na internetu] Dostupno: https://www.computerhope.com/jargon/g/gui.htm Pristupano: 24.08.2020.

[15] *„BFXR“* [Na internetu] Dostupno: https://www.bfxr.net/ Pristupano: 24.08.2020.

[16] *„Creating 2D Particle Effects in Unity 3D“* [Na internetu] Dostupno: https://blog.kongregate.com/creating-2d-particle-effects-in-unity3d/ Pristupano: 24.08.2020.

[17] *„Unity Manual: Streaming Assets“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/StreamingAssets.html Pristupano: 24.08.2020.

[18] *„Unity Manual: Monetisation“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/UnityAnalyticsMonetization.html Pristupano: 24.08.2020.

[19] *„Unity Manual: Configuring for Google Play Store“* [Na internetu] Dostupno: https://docs.unity3d.com/Manual/UnityIAPGoogleConfiguration.html Pristupano: 24.08.2020.

[20] *„Udemy: Complete C# Masterclass“* [Na internetu] Dostupno: https://www.udemy.com/course/complete-csharp-masterclass/ Pristupano: 03.07.2020.

[21] *„Udemy: Build a Complete 2D Mobile Platformer Game“* [Na internetu] Dostupno: https://www.udemy.com/course/unity-2d-course-build-a-mobile-platformer-game-from-scratch/ Pristupano: 15.07.2020.

9.2. Literatura korištena u programskom kodu

[22] Unity Technologies, *Vector2* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Vector2.html Pristupano: 18.07.2020.

[23] Unity Technologies, *Quaternion.identity* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Quaternion-identity.html Pristupano 19.07.2020.

[24] Unity Technologies, *MonoBehaviour.Invoke* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Invoke.html Pristupano 19.07.2020.

[25] Unity Technologies, *GameObject.GetComponent* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html Pristupano 17.07.2020.

[26] Unity Technologies, *Collider.OnTriggerEnter(Collider)* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html Pristupano: 19.07.2020.

[27] Unity Technologies, *MonoBehaviour.StartCoroutine* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html Pristupano: 19.07.2020.
[28] Unity Technologies, *MonoBehaviour.OnCollisionEnter2D* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnCollisionEnter2D.html Pristupano: 19.07.2020.
[29] Unity Technologies, *Object.Instantiate* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html Pristupano: 19.07.2020.
[30] Unity Technologies, *Object.Destroy* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Object.Destroy.html Pristupano: 18.07.2020.
[31] Unity Technologies, *AudioSource.PlayClipAtPoint* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/AudioSource.PlayClipAtPoint.html Pristupano: 20.07.2020.
[32] Unity Technologies, *Input.GetAxisRaw* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Input.GetAxisRaw.html Pristupano: 18.07.2020.
[33] Unity Technologies, *Animator.SetInteger* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Animator.SetInteger.html Pristupano: 19.07.2020.
[34] Unity Technologies, *Physics2D.OverlapBox* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Physics2D.OverlapBox.html Pristupano: 19.07.2020.
[35] Unity Technologies, *Input.GetButtonDown* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Input.GetButtonDown.html Pristupano: 20.07.2020.
[36] Unity Technologies, *Rigidbody2D.AddForce* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Rigidbody2D.AddForce.html Pristupano: 20.07.2020.
[37] Unity Technologies, *GameObject.CompareTag* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/GameObject.CompareTag.html Pristupano: 19.07.2020.
[38] Unity Technologies, *Gizmos.DrawLine* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Gizmos.DrawLine.html Pristupano: 19.07.2020.
[39] Unity Technologies, *Object.DontDestroyOnLoad* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Object.DontDestroyOnLoad.html Pristupano: 20.07.2020.
[40] Unity Technologies, *Debug.Log* [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Debug.Log.html Pristupano: 19.07.2020.

10. Popis slika

Slika 1. Primjer Platformer igre Donkey Kong: https://www.classicgaming.cc/classics/donkey-kong/images/donkey-kong-arcade-screen-bentgirders.jpg Pristupano 18.8.

Slika 2. Primjer Platformer igre Super Mario Bros: https://www.novilist.hr/wp-content/uploads/2020/07/super-mario.jpg Pristupano 18.8.

Slika 3. – 25. Vlastite slike (Isječci ekrana)

11. Popis korištenih resursa

11.1. Korišteni programski alati

1. Programski alat za izradu igrica (eng. *Game Engine*): Unity, https://unity3d.com/get-unity/download Pristupano 14.7.2020.
2. Razvojno okruženje (eng. *Integrated Development Environment*, skraćeno IDE): Microsoft Visual Studio uz Unity dodatak https://visualstudio.microsoft.com/downloads/ Pristupano 14.7.2020.
3. Program za izradu ilustracija: Inkscape, https://inkscape.org/hr/ Pristupano 18.7.2020.
4. Program za izradu zvučnih efekata: BFXR, https://www.bfxr.net/ Pristupano 24.7.2020.

11.2. Korišteni „Asset-si“

1. Pozadinska muzika igre: Marimba Boy https://www.dl-sounds.com/royalty-free/marimba-boy/ Pristupano 24.7.2020.
2. Korišteni font unutar igre: Baloo Regular https://www.fontsquirrel.com/fonts/baloo Pristupano 17.7.2020.
3. Set Ilustracija i animacija za glavnoga lika i protivnike: https://www.gameart2d.com/character-spritesheet-7.html Pristupano 15.7.2020.
4. Set blokova (eng. *Tileset*) za izgradnju svijeta: https://www.gameart2d.com/platformer-game-tileset-6.html Pristupano 15.7.2020.
5. Set ilustracija za izradu grafičkog korisničkog sučelja: https://www.gameart2d.com/free-fantasy-game-gui.html Pristupano 15.7.2020.