Izrada platformske igre u programskom alatu Unity

Dumančić, Hrvoje

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Hrvoje Dumančić

IZRADA PLATFORMSKE IGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Dumančić

Matični broj: 43981/15–R

Studij: Informacijski sustavi

IZRADA PLATFORMSKE IGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Dr.sc. Mladen Konecki

Varaždin, kolovoz 2018.

Hrvoje Dumančić

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

Ovaj rad se zasniva na upoznavanju programskog alata i razvojnog okruženja Unity te izrade 2D platformske igre nazvane Dani Kedera. Igra se sastoji od uvodnog nivoa i dva nivoa s različitim elementima na kraju kojih se događa izračun rezultata i njegovo pamćenje. Nivoi su konceptualno različiti s raznim preprekama kao što su neprijatelji, šiljci ili kombinacije takvih kako bi prelazak bio što izazovniji. Igrač ima različite vrste napada poput skoka ili udaraca mačem za svladavanje neprijatelja te više vrsta skokova za svladavanje prepreka, koji se moraju kombinirati kako bi se nivo uspješno završio. Također, postoji sustav bodovanja u obliku novčića i brzine prelaska nivoa, što daje dodatnu razinu igri i mogućnost ponovne igrivosti.

Ključne riječi: 2D igra, platformer, Unity, nivo, animacije, parallax scrolling

Sadržaj

1. Uvod

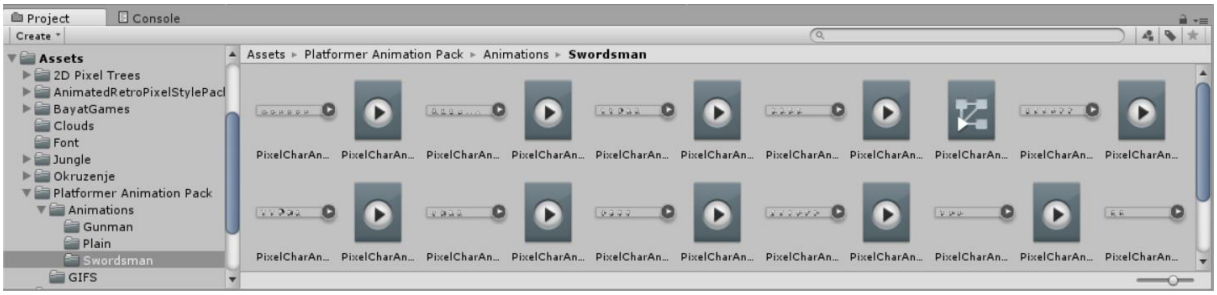
Jedne od najpoznatijih kategorija računalnih i konzolnih igara su platformeri, odnosno platformske igre. One postoje još od osamdesetih godina prošlog stoljeća, te iako su danas popularniji drugi žanrovi, njihov utjecaj se vidi u svakoj igri. Od skakanja na više razine unutar razine za dohvat posebnih pogodnosti, do prijelaza prepreka pomoću rješavanja zagonetki, platformer jer jedna od najbitnijih igara iz koje su nastali neki od najpoznatijih likova, kao što je Super Mario od Nintenda [6] ili Sonic the Hedgehog [5] od Sege.

Za ovaj projekt sam nastojao rekreirati same osnove platformera zbog kojih su postali popularni prije toliko godina. Igra je dosta jednostavna, s velikom mogućnosti igrivosti zbog postizanja što većeg rezultata i prelaska nivoa u što kraćem vremenu. Sastoji se od svega dva nivoa koji su konceptualno dosta različiti i zanimljivi za prolazak, svaki sa svojim jedinstvenim značajkama.

2. Programski alat Unity

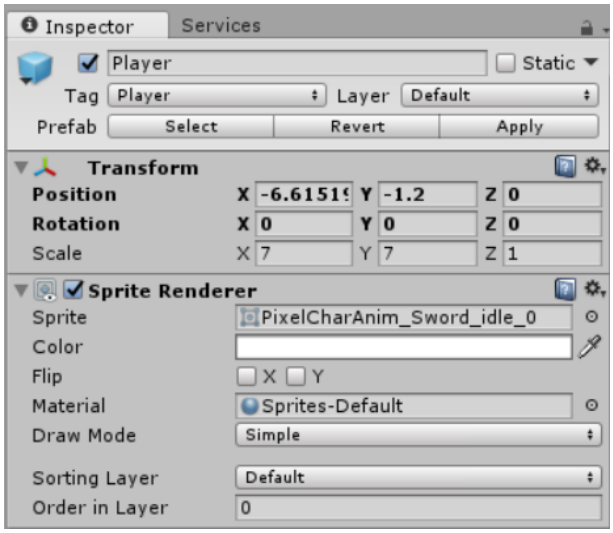
Baza za ovaj projekt izgradnje 2D igre je programski alat i razvojno okruženje Unity. Stvoren od strane Unity Technologies tvrtke 2005. godine, prvotno je bio isključivo namijenjen razvoju za Apple OS X operativne sustave, danas podržava 27 platformi, za koje se mogu stvarati sadržaji različitog tipa. Sadržaji koji se mogu pomoću Unityja stvarati su sučelja za pametne televizore i konzole, mobilne i desktop aplikacije, zadnje i daleko najpopularnije, dvodimenzionalne i trodimenzionalne igre za mobilne uređaje, računala ili konzole. [2]

Glavni prozor u kojem se događa najveći dio razvoja se sastoji od pet elemenata: prikaz preglednika projekta, prikaz inspektora, prikaza igre, prikaza scene te hijerarhije. Preglednik projekta se nalazi u donjem dijelu ekrana te on služi za navigaciju kroz elemente (engl. assets) kojom se raspolaže tijekom razvoja projekta i drugih stvari, poput scena ili skripti prilikom razvoja igara. On je na identičnoj poziciji i skoro pa sličnog izgleda na svim operativnim sustavima na kojima se može raditi u Unityju, radi bržeg i jednostavnijeg snalaženja



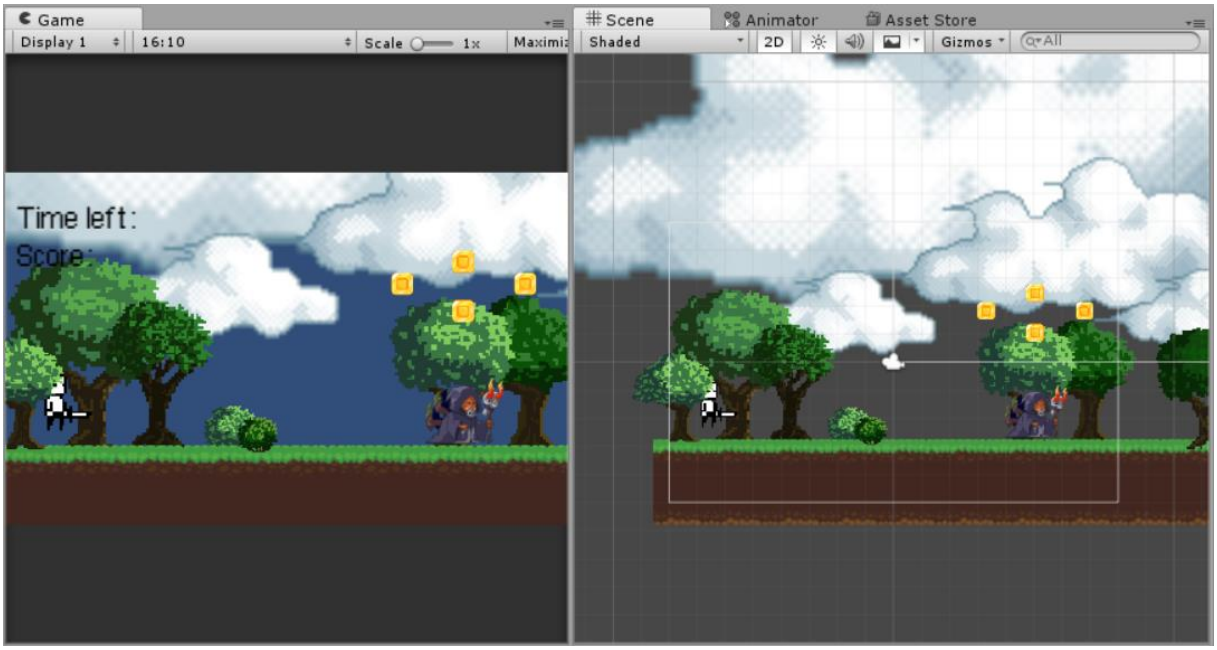
Slika 2. Prikaz preglednika u programskom alatu Unity

developera.[3] Inspektor je prikaz sastava pojedinog objekta koji nam se nalazi na desnoj strani ekrana gdje se mogu vidjeti sve skripte vezane uz objekt, svi njegovi elementi i pridružene vrijednosti.



Slika 1. Prikaz inspektora u programskom alatu Unity

Sljedeći dio je prikaz igre, koji služi za pregledavanje kako izgleda konačna igra koju izrađujemo sa svim elementima povezanim u cjelinu. Taj prikaz se razlikuje od prikaza scene jer u sceni možemo mijenjati i podešavati pojedine elemente, dok u prikazu igre ne možemo ništa mijenjati. Također, kameru u prikazu scene možemo sami kontrolirati i neovisna je o izvođenju igre, dok kamera u prikazu igre predočava finalnu kameru kakva će biti u samoj igrici. Osim prikaza scene i igre, mogu se uključiti unutar tih prozora i drugi prikazi poput animatora, same stranice za elementu na internetu te drugih stvari, od koje smo već neke i spomenuli. [2]



Slika 3. Prikaz igre i prikaz scene u programskom alatu Unity

Hijerarhija je posljednji prikaz koji nam je ostao, te koji govori o svim objektima koji su uključeni u trenutnoj sceni.

Slika 4. Prikaz hijerarhije u programskom alatu Unity

Osim samih prikaza objekata kojom se raspolaže za izradu igre, potrebno je spomenuti i mjesto s kojeg se ona može i preuzeti, odnosno kupiti. Unity Asset Store je online trgovina koja služi za nabavu objekata i elemenata (engl. asset packova) koji se mogu koristiti odmah u igrici. Ovisno o pojedinom preuzetom elementu, neke se mogu odmah drag-and-drop metodom uključiti u samu igru i kao takve su spremne za korištenje, dok se kod drugih moraju napraviti neke jednostavnije preinake poput mijenjanja veličina, uključivanja skripti itd. kako bi se mogle koristiti u igrici. Ovisno o samoj dovršenosti elementa koja se preuzima, cijene se razlikuju od potpuno besplatnih do pojedinih dosta skupih elemenata. [4]

3. Povijest platformerskih igara

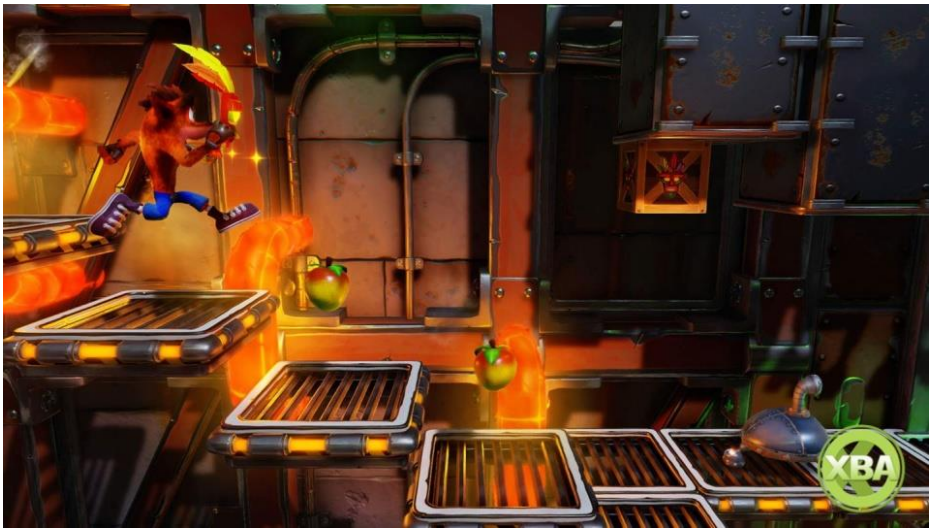
Prve igre takvog tipa počele su se pojavljivati 80-ih godina prošlog stoljeća, kao što su Space Panic [12] te Donkey Kong [7]. Takve igre sastojale su se od samo jednoga ekrana, odnosno nije postojalo mogućnosti za kretanjem u stranu ili visinu više od samog okvira kamere, zbog čega su takve igre zasnivane na postizanju što većeg rezultata dok neprijatelji neprestano dolaze. Iako su predstavljale tek uvod u mnogo kompleksnije i veće igre, određeni ljudi još uvijek igraju prvotne platformere, te se čak i natječu na globalnoj razini tko će postići više bodova kroz jedno igranje.

Sredinom 80-ih godinama, napretkom tehnologije i pojavom novih konzola poput NESa (Nintendo Entertainment System) te Sega Master Systema, koje su imale veću procesorsku snagu od svojih prethodnika, postalo je moguće stvaranje većih nivoa s mogućnosti kretanja u stranu, pojavljivanje glavnih bitki s „Bossovima“, te pojačanjima s kojima se dobivaju super moći na određeno vrijeme. Dvije kompanije koje su se bavile proizvodnjom konzola, Nintendo i Sega, upravo su kompanije koje su u to doba vodile bitku za proizvodnju što boljih igara za svoje korisnike. Nintendo je tako došao na ideju Super Maria [6], poznatog vodoinstalatera koji spašava svoju princezu od raznih neprijatelja na razne načine. S druge strane, Sega je proizvela igricu u kojoj jež Sonic [5] spašava svoje sugrađane od neprijatelja. Priče su ionako zapravo bile praktički nebitne u samim igrama, jer su se htjeli prikazati napreci u industriji, a te dvije igre bile su idealan primjer takvih postignuća. Zbog toga, igre su postale iznimno popularne te su proizvedene u još nastavaka, te su glavni likovi postali maskote kompanije i jedni od najvećih ikona modernih igara.

Slika 5. Prikaz iz igre Super Mario Odyssey [21]

Sljedeća razdoblja bitna za svijet platformera počela su sa sljedećim sustavima, koji su omogućivali razvoj 16-bitnih igara, te kasnije i 3D igara. 16-bitne igre kao što su Super Mario World 2: Yoshi's Island [8] ili Mega Man [9] donijeli su nove napretke, u obliku Parallax Scrollinga, pomoću kojega su određene igre dobile prividnu dubinu preko različite brzine micanja oblaka u pozadini. Također, povećanje s 8-bitnih tehnologija na 16-bitne, omogućilo je puno detaljnije prikaze likova i okoliša.

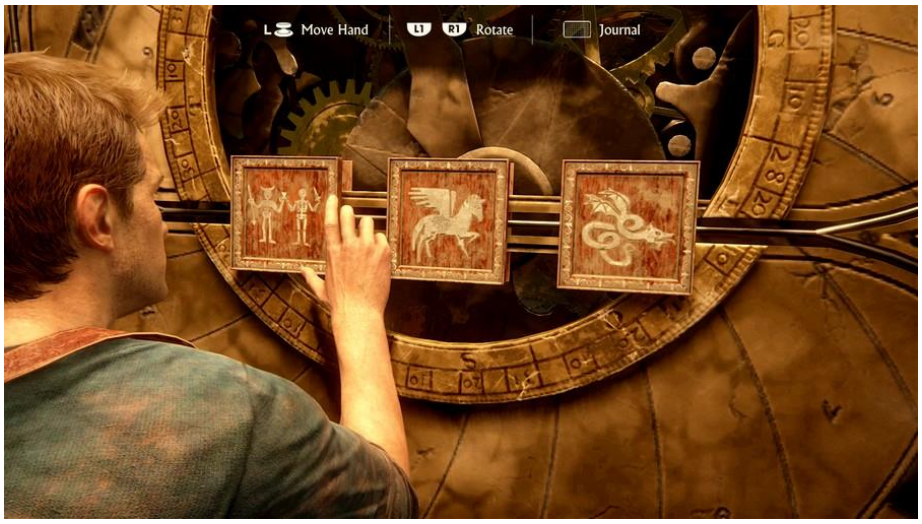
Trodimenzionalni platformeri su predstavljali najveću revoluciju platformerskog žanra te igara općenito. Iako su prvotno proizvođači igara pokušavali implementirati 2D grafiku u 3D svijetu kao što je Konamijeva Antarctic Adventure igra [11] ili Segina Congo Bongo [10]. Iako su te igre u svoje vrijeme bile popularne, moglo je se otpočetka vidjeti da postoji mjesta za napredak. Sljedeći pokušaji kao što su Crash Bandicoot od Sonyja su postali iznimno popularne igre, ali ponovno, to nisu bili pravi 3D platformeri sa slobodnom pokreta u svim smjerovima. Prva igra koja je uspjela u pokušajima za potpunom slobodom pokreta unutar svijeta je Super Mario 64. Nintendo je nakon razvijanja nove konzole razvio Nintendo 64, 64-bitnu i kao takvu, jedinu konzolu na svijetu. Najveća procesorska moć u to vrijeme povezana s velikim otvorenim svijetom i jednostavnim kontrolama dovela je do velike popularnosti igre, s time i konzole. Nakon što je Nintendo uvidio popularnost takvog tipa igara, počeo je s razvojem novih, poput Banjoo-Kazooie [13] te Donkey Kong 64 [7] , dok su druge kompanije kao Sony pokušale sa svojim igrama, poput Lara Croft Tomb Raidera [15] te mnogih drugih.



Slika 6. Prikaz iz igre Crash Bandicoot [22]

Novo stoljeće dolazi s novih igrama, boljim grafikama, ali sličnim kontrolama kao i u prošlim generacijama. Ove generacije ne donose revolucionarne napretke kod igara, nego

donose integraciju platfromerskih segmenata u velikom broju drugih žanrova. Tako novi naslovi poput Uncharteda [16] ili God Of Wara [17] implementiraju u određenim svojim dijelovima rješavanje zagonetki, koje potječe upravo iz platformera. Druge igre poput Cupheada [18] ili Hollow Knighta [19] žele replicirati osjećaj platformera iz prošlog stoljeća, ali s modernim segmentima i današnjim normama. Također se pojavljuju remasteri ili remakeovi starih igara koje više ne izlaze poput Crash Bandicoota [14] ili Spyro The Dragon [20], koje su zapravo identične replike originalnim naslovima, ali napravljene na moderan način s detaljnijim grafikama. [1]



Slika 7. Prikaz iz igre Ucharted 4: A Thieves End [23]

4. Razvoj vlastite igre

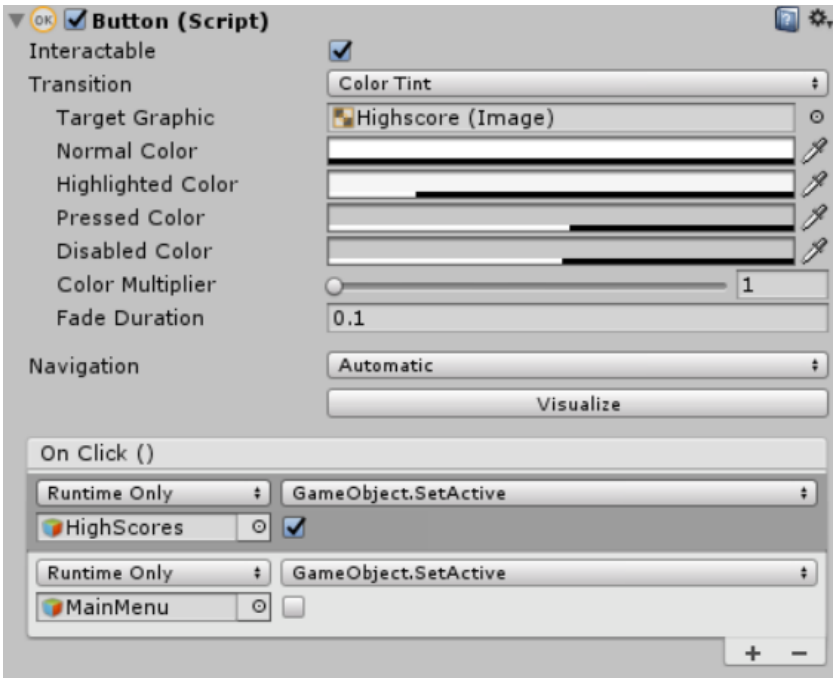
Poglavlje opisuje proces razvoja vlastite igre pod nazivom Dani Kedera u kojem će biti opisane scene od kojih se igra sastoji, skripte te objekti koji su se koristili u izradi igre.

4.1. Scene

Napravljena igra se sastoji od četiri scene, Meni, Uvodni_nivo, Nivo_1 i Nivo_2.

4.1.1. Meni

Meni je scena koja se pokreće prva prilikom ulaska u igricu. Scena Meni prvi vrhu ima prikaz naslova igre, Dani Kedera, te ispod mogućnosti odabira različitih razina unutar igre, uvoda, prvog i drugog nivoa. Pri dnu se nalazi opcija za pregled najvećih rezultata iz dva nivoa te opcija za izlazak iz aplikacije pomoću koje se zatvara igra. Sve opcije osim naslova imaju mogućnost klika, pomoću kojih se otvaraju druge scene ili određene implementirane opcije. Opcije su element tipa dugme (engl. button), na koji je stavljen element tekst. Dugme je transparentno kako bi se uklopilo s pozadinom, no prilikom prelaska mišem preko njega, malo manje postane transparentno kako bi korisnik vidio da se nalazi iznad dugmeta. Klikom na njega još manje postaje transparentno radi ostavljanja ljepšeg ugođaja.

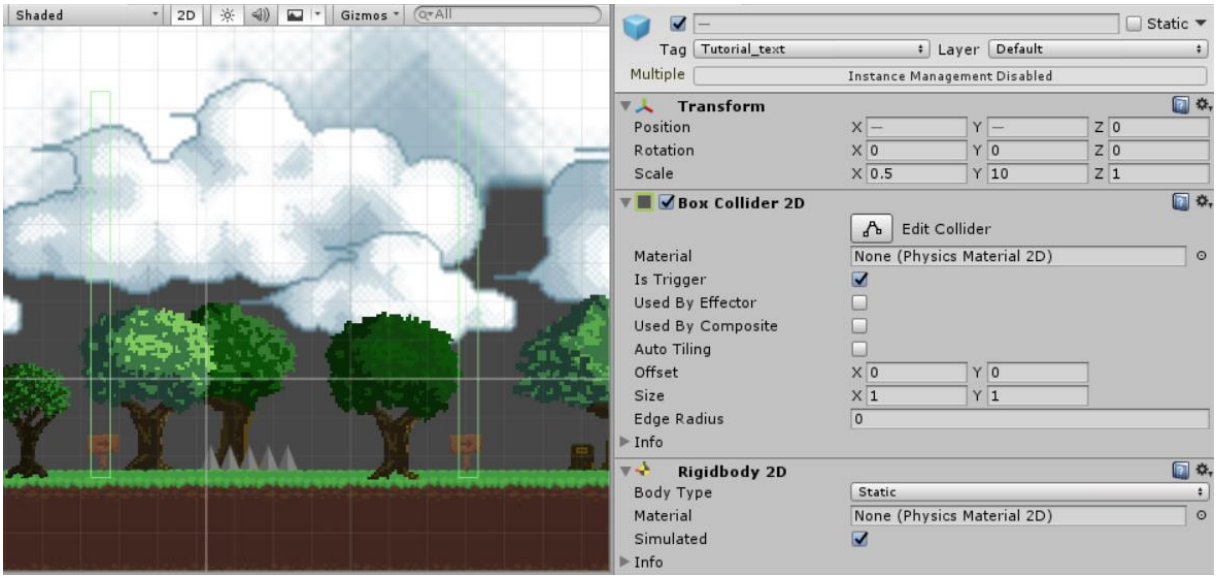


Slika 8. Deaktivacija MainMenu zaslona i aktivacija HighScoresa preko dugmeta

Osim navedenih opcija koje se sve nalaze unutar GameObjecta MainMenu, postoji i GameObject Highscore koji sadržava prikaz najvećeg broja bodova iz prve dvije razine. On je prvotno isključen, no klikom na High Score dugme na Glavni Meni zaslonu on se aktivira, dok se Highscore Gameobject aktivira, odnosno postane vidljiv. On sadrži osim prikaza rezultata i naslova igre i dugme nazad koji imaju obrnuti učinak, deaktivira Highscore, a aktivira MainMenu.

4.1.2. Uvodni Nivo

Iako je moguće odabrati bilo koji nivo prilikom pokretanja igre, ovo je prvi nivo, odnosno scena, koju bi osoba koja je tek ušla u igricu trebala otvoriti. Scena predstavlja uvodni nivo koji je napravljen da predstavlja uvod u mehaniku igre, zbog čega je dosta jednostavan i sadrži osnovne mehanike koje će se koristiti. Tako se prilikom kretanja po nivou dolazi do tabli koje predstavljaju pojedini tekst koji govori korisniku što i kako raditi u igri. Table zapravo ništa ne označavaju, nego je u pozadini postavljen veliki nevidljivi element koji je ujedno Box Collider koji djeluje kao trigger. Kada se aktivira trigger, na temelju imena tog elementa se u switch elementu odabere tekst koji će se ispisati na ekran, te tako dobivamo ispis uputa. Također postoji i za svaki nevidljivi Box Collider i prikladna boolean varijabla preko koje se u switchu provjerava dali se uputa već ispisala ili ne. Ako nije, ući će u blok koda gdje će se ispisati i postaviti se na istinito, kako bi se daljnji ulaz ograničio, odnosno koliko god puta aktiviramo trigger za taj natpis, on će se ispisati samo jednom.



Slika 9. Box Collidera kao trigger za ispis uputa na ekran

Nevidljivi elementi također postaju i kako bi ograničili kretanje igrača, odnosno neprijatelja u igrici. Za igrača postoje tri nevidiljiva elementa, koji su Box Collideri kao triggeri. Po jedan na početku i na kraju nivoa kako igrač ne bi mogao skočiti s kraja razine u smrt, gdje ga kamera ne bi pratila, a treći element će se opisati kod prvog nivoa gdje dolazi do potpunog izražaja. Drugi nevidljivi elementi koji se nalaze postoje za kretanje neprijatelja, gdje su svakome stavljeni lijevi i desni zid kako bi se osiguralo da oni budu točno u određenom rasponu, kako bismo ih mogli postaviti više na određenoj površini.

Uvodni nivo je dosta jednostavan, te sadrži elemente koji se u sljedećim nivoima isprepliću i složeniji su. Tako u uvodu jedna od prvih stvari na koju se susreće su šiljci. Potrebno je preko njih preskočiti ili će igrač umrijeti, zbog čega će se morati nivo ispočetka igrati. Šiljci su napravljeni kao Polygon Collidersi, zbog toga što želimo da točan oblik šiljka da postane collider za igrača, odnosno želimo što točniji prikaz šiljka za slučaj da igrač sleti na njega i od njega umre.

```
private void OnTriggerEnter2D(Collider2D collision)
    {
        switch(collision.gameObject.name)
        {
            case "Tutorial_move":
                if(tutorial_move == false)
                {
                    PrintOutText("Use the arrow keys to move left and
right!");
                    tutorial_move = true;
                }
                break;
            case "Tutorial_text_jump":
                if(tutorial_jump == false)
                {
                    PrintOutText("Press space to jump!");
                    tutorial_jump = true;
                }
                      break;
```

Sljedeći elementi su Floating Tilesi, također Box Collideri koji lete u prostoru te kao takvi predstavljaju objekte na koje igrač može sletiti i koji daju osjećaj platformera napravljenoj igrici. Služe za nastavak razine ako je previše šiljaka da bi se preskočilo ili kao drugi način ubijanja neprijatelja. Kasnije ti sami floating tilesi su dobili još jednu dodatnu funkciju, kojom se igraču daje još veći izazov, a to je da propadaju kada igrač stoji na njima, koji će biti bolje opisani kasnije. Na jedne floating tilese pri početku nivoa postavljeni su šiljci kako bi korisnik

morao čučnuti da prođe ispod njih. Prije tog dijela nalazi se ploča na kojoj se aktivira natpis kako korisnik treba čučnuti, stvar koja će mu kasnije trebati za prolazak drugih nivoa.

4.1.3. Prvi nivo

Ovo je razina u kojem se prvi put pojavljuju novčići, odnosno novčići koje igrač može skupljati tijekom prolaska nivoa gdje za svaki skupljeni dobije plus 10 bodova koji mu se računaju u ukupan rezultat za konačan high score, te se svaki skupljeni novčić uništava, odnosno nestaje.

Slika 10. Slike za animaciju novčića

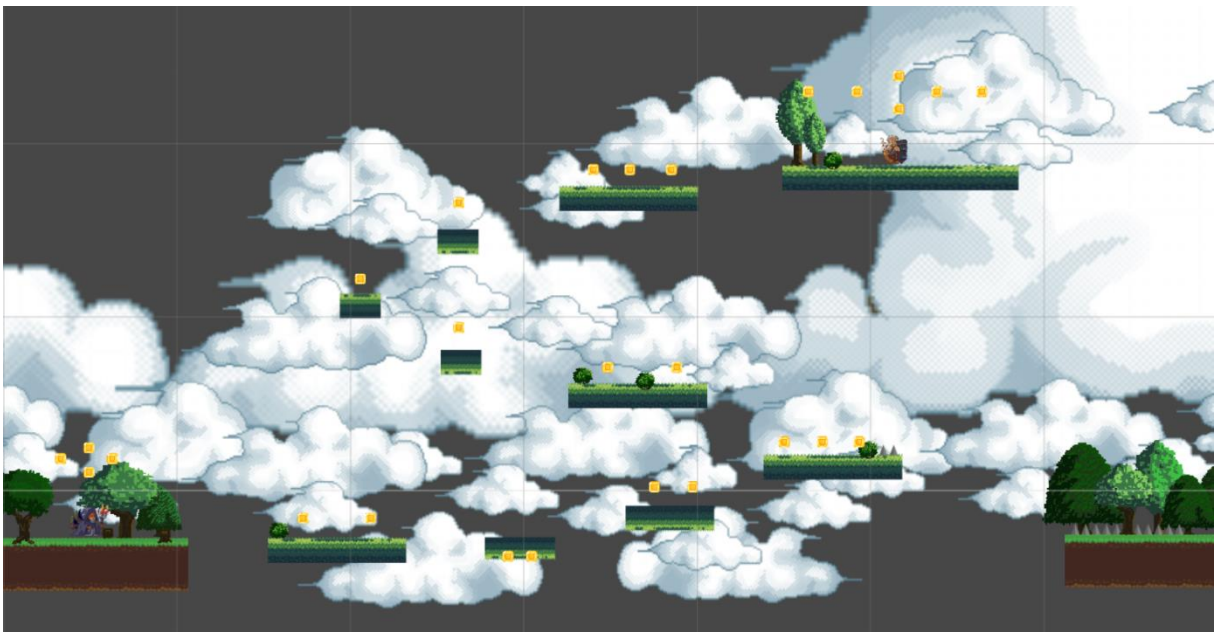
Osvojeni bodovi mogu se provjeriti u gornjem lijevom kutu ekrana, gdje se također može vidjeti i vrijeme koje je ostalo da se prijeđe razina. Za oba nivoa vrijeme u kojemu se trebaju preći je 120 sekundi, odnosno dvije minute. Ako to vrijeme istekne, igrač umre i resetira se na početnu poziciju nivoa gdje mora ponovno sve proći kako bi ga završio. Također se poništavaju svi bodovi. U nivou je postavljeno dosta novčića, ali igrač ako bude skupljao sve, onda će mu isteći previše vremena te neće uspjeti ostvariti dovoljno visok rezultat, zbog čega treba uzeti omjer između brzine i skupljanja novčića kako bi se postigao što veći rezultat. Treći nevidljivi Box collider element kojeg smo već spomenuli je nevidljivi element je EndLevel i ima tag Finish. On služi kako bismo znali da je korisnik došao do kraja nivoa, gdje mu se bodovi za preostalo vrijeme zbrajaju na broj skupljenih novčića, kako bi se izračunao high score. Izračunavanje se jedino ne događa kod uvodnog nivoa, gdje nema vremena, ali nema ni novčića koji se trebaju skupljati.

Slika 11. Prikaz preostalog vremena i prikupljenih bodova

Ovdje se po prvi put pojavljuju propadajući floating tilesi koji su stavljeni, iako se to igraču ne vide, kao pozadinski element, odnosno stavljeni su na poseban layer kako bi se održali u pozadini. To je zbog toga da se prilikom propadanja, oni propadaju u pozadinu kako bi se šiljci ostavili u prvom planu i kako bi igrač znao da nema još puno vremena ih izbjeći. Ti elementi propadaju samo dok je korisnik na njima, stoga ako se žele skupiti svi novčići moguće je skakati s jedne na druge u brzini kako bi se stiglo skupiti sve njih i kako bi se moglo dobiti što više bodova. Nivo završava s dva neprijatelja koje je potrebno poraziti kako bi se došlo do kraja.

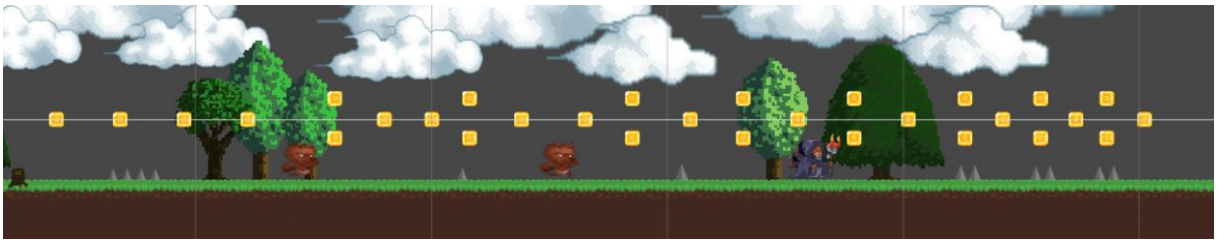
4.1.4. Drugi nivo

Ovaj nivo počinje s par neprijatelja koje odmah treba zaobići ili ubiti kako bi se nastavio. Dalje, nakon neprijatelja dolazimo do novog dizajna kojeg još prije nismo nigdje susreli, a to je skup floating tilesa od kojih neke propadaju, koji se treba proći kako bi se nastavio nivo. Ako se na neke ne doskoči, igrač propada dolje u smrt ili na nižu razinu tilesa koje je već prošao. Usto, određene od njih propadaju s vremenom kad se stoji na njima kako bi nivo bio još teži. Novčići su na nekima postavljeni ispod tilesa koji propadaju, tako da bi se stajanjem na njima određeno vrijeme oni mogli pokupiti, ali s obzirom na to da takvi tipovi tilesa propadaju, tada je teže skočiti na sljedeći tile, odnosno nastaviti nivo. Tu se događa i promjena kamere po vertikalnoj osi koja se očituje sa skakanjem igrača, što je specifično za ovaj nivo. Na zadnjem floating tileu se nalazi novi neprijatelj sa štitom. Tog neprijatelja moguće je riješiti kao i sve prošle neprijatelje, na isti način te skupiti novčiće pored njega.



Slika 12. Koncept drugog nivoa s floating tilesima

Nakon toga dolazi poveći skok koji završava na ground tilesu, na istoj razini na kojoj smo i počeli nivo. Tu je namjerno napravljen jedan mali trik za igrača, a to je da prilikom skoka na ground tile igrač mora napraviti veliki skok i to sa samoga ruba, inače će se dočekati na šiljke i umrijeti, te će morati cijeli nivo ponovno igrati. Također, šiljci su prisutni i u nastavku gdje igrač mora izbjegavati ih dok ubija neprijatelje, čime je još teže preći nivo. Ti šiljci su prikladno pozicionirani na nevidljivim elementima kojima se ograničava kretanje neprijatelja čime osiguravamo kako se neprijatelji neće s njima sudarati te tako usporavati ili zaustavljati svoje



Slika 13. Završetak drugog nivoa

kretanje. Razina završava s još šiljaka koji su postavljeni na malom razmaku te igrač ne može odmah skakati s jednih na druge jer bi umro, nego mora nakon svakog skoka dočekati se te tek nakon toga skočiti opet.

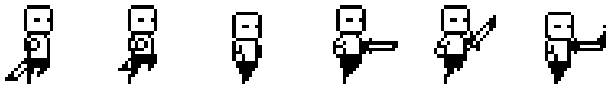
4.2. Elementi

Postoji veliki broj objekta koje sam koristio u projektu, te će u nastavku nekolicina njih biti podrobnije objašnjena, kao što su objekti za glavnog lika, neprijatelje i slično.

4.2.1. Glavni igrač

Elementi za glavnog lika sastoje se od tri kategorije, od kojih je samo jedna korištena s praktički svim funkcionalnostima. Uz elemente gdje glavni lik ima mač, postoje elementi gdje je on bez oružja te također gdje on ima pištolj u rukama. Prvotna ideja je bila kombinacija pištolja i mača, ali nakon pregleda animacija i uvida da ne postoji opcija pucanja s igračem, preostala je samo opcija s mačem. Igra tako ima tri vrste napada s mačem, quick, medium i heavy attack, uz mogućnost skoka na neprijatelja kako bi ga se usmrtilo. Za svaki napad je drugačija animacija te različita veličina Box collidera koja se mijenja, koja će biti objašnjena kasnije u mehanici igre. Osim te tri opcije za napad, objekt također sadrži i animacije za skok, mirovanje, trčanje te čučanje za prolazak ispod određenih prepreka. Prva animacija koja se pokreće prilikom pokretanja bilo kojeg nivoa je mirovanje, te je također animacija preko koje se pristupa svim ostalima. Sve one su povezane preko boolean varijabli koje se postavljaju na true ili false ovisno o tipkama ili situacijama u kojima se igrač nalazi, što će biti kasnije opisano.

Slika 15. Prikaz slika animacije za teški napad



Slika 14. Prikaz slika animacije za trčanje

4.2.2. Neprijatelji

U ovom folderu se nalaze elementi koji se koriste za neprijatelje, kojih imamo tri vrste. Imamo vještицu, vuka i neprijatelja sa štitom. Svo troje ima istu funkcionalnost i isti način borbe, kreću se u jednom smjeru dok ne naiđu na zid gdje se onda okrenu te nastave kretanje u drugom smjeru. To se događa cijelo vrijeme, neovisno o igraču. Također neprijatelj ima opciju udarca koja se aktivira nakon određenog trenutka koji je predefiniran te se prilikom udarca povećava Box collider kako bi sam udarac bio iste veličine kao i animacija koja se mijenja, što će biti

kasnije opisano u mehanici. U samom elementu postoji još tipova neprijatelja, ali za ovu fazu razvoja igre odabrana su ova tri zbog njihove zanimljivosti.

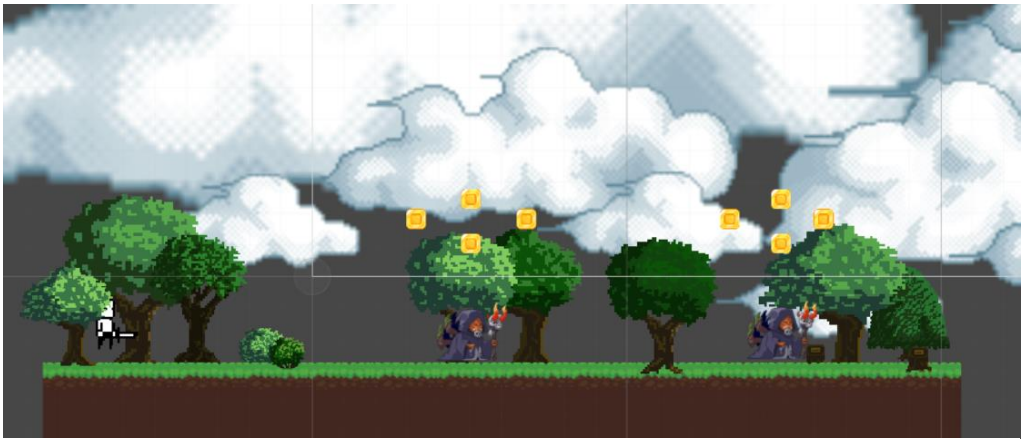
Slika 16. Prikaz slika animacije za vještice

Slika 17. Prikaz slika animacije za vuka

4.2.3. Okruženje

Postoje više objekata koji su se koristili za okruženje u igrici. Prvi, osnovni je Pixel Platformer Art Pack koji se koristio kako bi imali ground tilese po kojima se odvija većina nivoa, jer po njima igraču omogućavamo da hoda. Osim tih ground tilesa, ti objekti također sadrži i table koje smo postavili u uvodnom nivou za ispis uputa odnosno kontrola u igrici.

Sljedeći objekt je Jungle Asset Pack kojeg koristimo za floating tilese u dva oblika, kao stabilne ili kao nestabilne koji propadaju ako igrač stoji na njima.



Slika 18. Prikaz drveća i oblaka

Drveća i oblaci u pozadini predstavljaju obične slike koje su postavljene bez ikakvih elemenata povezanih na sebe, sve u smislu stvaranja ljepšeg ugođaja i okoline u kojoj se igrač nalazi. Drveća su sva poredana u tri layera, foreground, background ili more background, gdje je njihova dubina, odnosno Z os nepromijenjena, dok je kod oblaka to napravljeno pomoću

metode zvane Parallax Scrolling kojom uspjevamo dobiti određenu dubinu u igrici iako se nalazimo u 2D prostoru, što će kasnije biti detaljnije opisano.

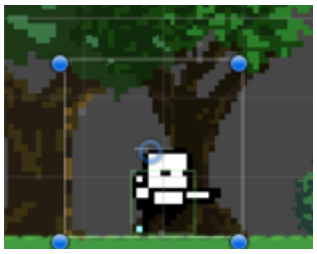
Preostali objekti su oni koji sadržavaju novčiće zajedno s njihovim animacijama te šiljke, koji su na određenim mjestima u razinama postavljeni gore ili dolje. Ti se objekti nalaze u folderu BayatGames, koji sadržava još dodatne stvari, od kojih su samo ove korištene.

4.3. Mehanika

4.3.1. Igrač

Tri su skripte koje su specifične za igrača, a to su skripte za kontrolu, rezultat i zdravlje. Skripta za kontrolu prilikom svojeg pokretanja upisuje u dvije varijable veličinu i offset Box collidera, što je potrebno kontrolirati kad igrač čučne u igrici. Kod čučnja se događa nova animacija, koja ostaje sve dok igrač ne stisne opet tipku kako bi se vratio u normalno stanje. Sama animacija nije dovoljna, zbog toga što Box collider ostane iste veličine te nije moguće proći ispod niskih predmeta u nivoima. Zbog toga smo uzeli prije početka izvođenja skripte startnu veličinu i offset Box collidera kako bismo mogli vratiti početnu veličinu kada igrač prestane s čučanjem, a kad se čučne veličinu podijelimo s 2, a offset pomnožimo s 2.1. Veličinu dijelimo s 2 radi smanjenja veličine samog Box collidera, a offset dijelimo zato što bi dijeljenjem s 2.1 igrač ostao u zraku, tj. pomaknuo bi se malo u zrak, ali trebamo suprotan učinak zbog čega ga dijelimo s 2.1 da što točnije i što bliže zemlji se nalazi. Potrebno je napomenuti još da prilikom bilo kojeg drugog pritiska tipke čučanj se poništi.

Slika 19. Igrač u normalnom položaju



Slika 20. Igrač u crouch položaju

```
public void Crouch()
    {
        BoxCollider2D collider = gameObject.GetComponent<BoxCollider2D>();
        Animator animator = gameObject.GetComponent<Animator>();

        if (crouching == false)
        {
            GetComponent<Animator>().SetBool("IsRunning", false);
            collider.size = new Vector2(collider.size.x, collider.size.y /
2);
            collider.offset = new Vector2(collider.offset.x,
collider.offset.y * 2.1f);
            animator.SetBool("IsCrouching", true);
            crouching = true;
        }
        else
        {
            collider.size = playerBoxColliderSize;
            collider.offset = playerBoxColliderOffset;
            animator.SetBool("IsCrouching", false);
```

```
            crouching = false;
        }
          }
```

Sljedeća funkcija je skok, kojeg postoje dvije vrste. Običan skok pritiskom na tipku skače samo do određene razine, ali postoji još jedan veći skok, kad se prvo sagne a tek onda skoči, radi privida odraza, zbog čega je taj skok veći. Skok je riješen tako da postoji boolean varijabla koja pokazuje da li je igrač prizemljen ili nije. Prilikom svakog dodira collidera s predmetom boolean varijabla IsGrounded se postavi na true, kao i animacija za mirovanje. Nakon toga se provjerava vrsta collidera na kojeg je sletio igrač, te ako je on vrste spike, odnosno šiljak, tada se pokrene animacija za smrt iz druge skripte, za zdravlje. Prilikom ulaza u funkciju za skok ta ista boolean varijabla se postavi za false te se upravo tako onemogućava da igrač skače beskonačno puno puta, nego samo jednom. Nadalje, provjerava se dali igrač čuči ili ne, ako čuči tada je skok višlji, odnosno ako nije onda je malo slabiji.

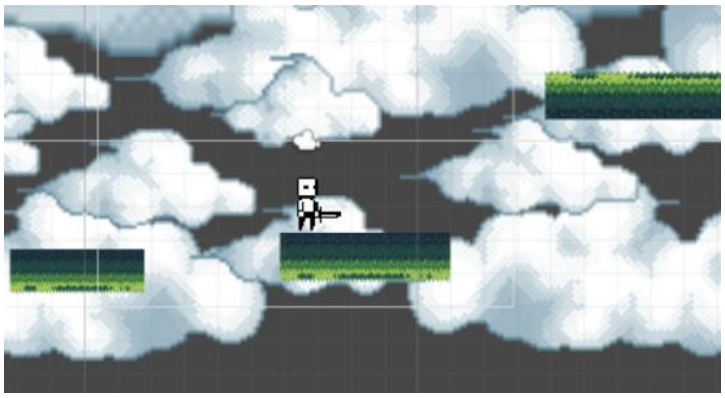
```
public void PlayerMove()
    {
        // CONTROLS
        moveX = Input.GetAxis("Horizontal");
        if (Input.GetKeyDown(KeyCode.Space) && isGrounded == true){
            Jump();
              }
```

Osim provjere dali je igrač sletio na šiljke u funkciji onColliderEnter2D, također se slična stvar provjerava u funkciji PlayerRaycast, koja se poziva unutar svakog Update-a. Unutar te funkcije se odvijaju dva Raycasta, jedan prema dolje i jedan prema gore. Raycast prema dolje nam je bitan zbog toga što pomoću njega provjeravamo dali je igrač sletio na neprijatelje ili možda na floating tiles, po njihovom tagu. Ako je skočio na neprijatelja, onda deaktiviramo određene skripte od neprijatelja od kojih je najbitnija skripta da je neprijatelj RigidBody2D, zbog koje neprijatelj više nije u doticaju s okruženjem, odnosno može propasti kroz površinu na dno. Ako je igrač skočio na floating files, tada oni propadaju po određenoj predefiniranoj brzini, koja iznosi 0.02 po Y osi, čime dobivamo konzistentan i dovoljno brz pad. Drugi Raycast koji se ispaljuje je prema gore, te on također provjerava dvije stvari. Jedna od tih je jesmo li skočili na šiljke koji se mogu nalaziti ispod floating tilesa, zbog čega ne smijemo kad se nalazimo u tom dijelu skakati na njih, a druga stvar koja se provjerava su elementi tipa box,

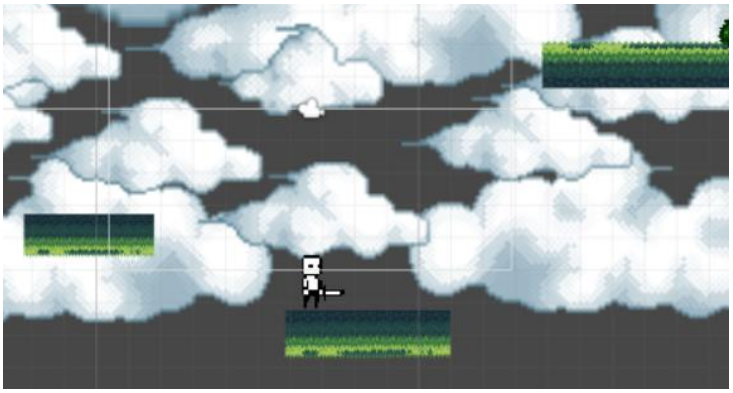
koji su inicijalno trebali biti uključeni, ali zbog ne pronalaska dovoljno dobrog prikaza tih kutija se od te ideje u ovoj fazi odustalo.

Slika 21. Prikaz slika animacije za skok

```
public void Jump()
{
 if(crouching == true)
 {
  Crouch();
  GetComponent<Rigidbody2D>().AddForce(Vector2.up * playerJumpPowerCrouch);
 }
 else
 {
  GetComponent<Rigidbody2D>().AddForce(Vector2.up * playerJumpPower);
 }
 isGrounded = false;
 GetComponent<Animator>().SetBool("IsGrounded", false);
}
```



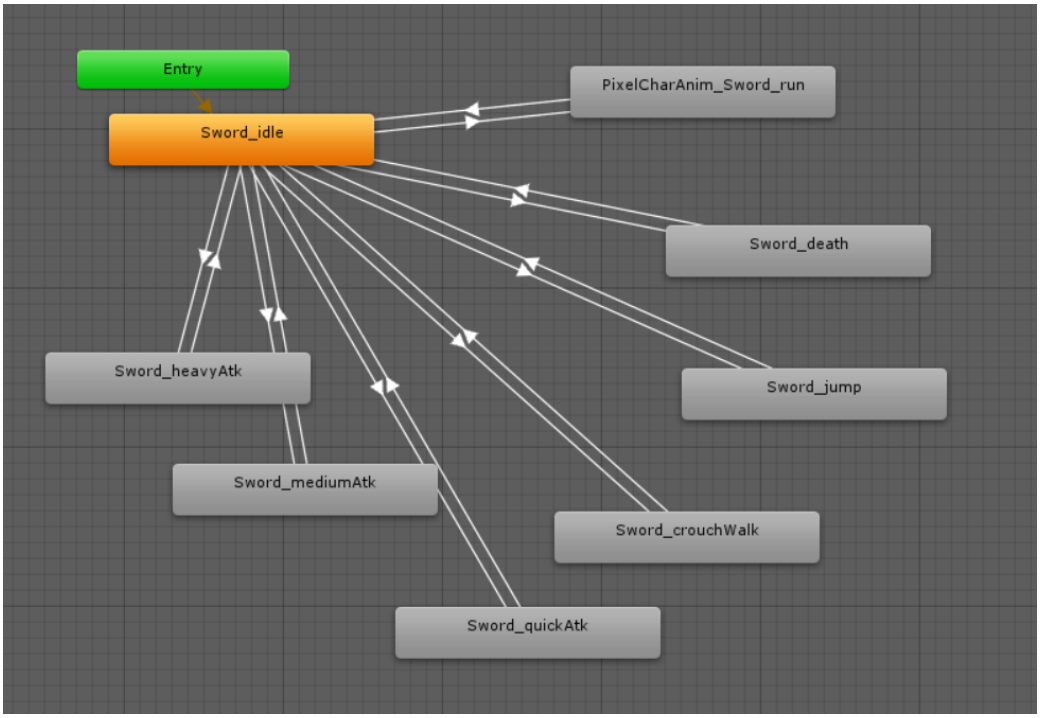
Slika 22. Propadanje floating tilesa



Slika 23. Propadanje floating tilesa

Još jedna funkcija koja se poziva prilikom Update-a je funkcija PlayerMove, možda i najbitnija funkcija, koja omogućuje kretanje i napad igrača i koja očitava koja tipka je pritisnuta na tipkovnicu u tom trenutku. Prvotno u funkciji se dobiva pomak po osi X, koji se zapisuje u varijablu moveX. Tada, se provjerava dali je pomak ako postoji pozitivan ili negativan, odnosno u kojem smjeru igrač ide kako bismo znali okrenuti igrača ako se kreće u suprotnom smjeru.

```
// PLAYER DIRECTION
        if (moveX < 0.0f)
        {
            GetComponent<SpriteRenderer>().flipX = true;
            if(GetComponent<BoxCollider2D>().offset.x > 0.0f)
            {
                GetComponent<BoxCollider2D>().offset = new
Vector2(GetComponent<BoxCollider2D>().offset.x * -1,
GetComponent<BoxCollider2D>().offset.y);
            }
            if (crouching == false)
            {
                GetComponent<Animator>().SetBool("IsRunning", true);
            }
        }
        else if (moveX > 0.0f)
        {
            GetComponent<SpriteRenderer>().flipX = false;
            if (GetComponent<BoxCollider2D>().offset.x < 0.0f)
            {
                GetComponent<BoxCollider2D>().offset = new
Vector2(GetComponent<BoxCollider2D>().offset.x * -1,
GetComponent<BoxCollider2D>().offset.y);
            }
            if(crouching == false)
            {
                GetComponent<Animator>().SetBool("IsRunning", true);
            }
        }
        else
        {
            GetComponent<Animator>().SetBool("IsRunning", false);
        }
        // PHYSICS
gameObject.GetComponent<Rigidbody2D>().velocity = new Vector2(moveX *
playerSpeed, gameObject.GetComponent<Rigidbody2D>().velocity.y);
```



Slika 24. Prikaz animacija igrača

```
public void Attack()
    {
        if (crouching == true)
        {
            Crouch();
        }

        Vector2 newCollider =
gameObject.GetComponent<BoxCollider2D>().size;
        if (Input.GetKeyDown(KeyCode.Alpha1) && !heavyAttack)
        {
            GetComponent<Animator>().Play("Sword_heavyAtk");
            heavyAttack = true;
            newCollider = new Vector2(newCollider.x * 2.35f,
newCollider.y);
        }
        else if (Input.GetKeyDown(KeyCode.Alpha2) && !mediumAttack)
        {
            GetComponent<Animator>().Play("Sword_mediumAtk");
            mediumAttack = true;
            newCollider = new Vector2(newCollider.x * 2.0f, newCollider.y);
        }
        else if (Input.GetKeyDown(KeyCode.Alpha3) && !quickAttack)
        {
            GetComponent<Animator>().Play("Sword_quickAtk");
            quickAttack = true;
            newCollider = new Vector2(newCollider.x * 1.7f, newCollider.y);
        }
        gameObject.GetComponent<BoxCollider2D>().size = newCollider;
}
```

Također, ako se igrač kreće postavljamo u Animatoru boolean IsRunning na true, kako bi se aktivirala animacija za trčanje, koja ostaje aktivna sve dok igrač se kreće po osi X. Sljedeća

stvar koja se provjerava su jesu li pritisnute tipke za skok, čučanj ili napad. Ako je pritisnuta bilo koja od tipki za napad, tada se poziva funkcija Attack koja prvotno ako igrač čuči poništava to, a drugo provjerava koji napad je pritisnut. Zbog toga što napad traje samo određeno vrijeme i nije ga moguće zadržavati kao čučanj, napravljene su nove boolean varijable kojima se kontrolira izvršenje napada. Prilikom pritiska tipke za napad, varijabla se postavlja na true, povećava se Box collidera neprijatelja jer se napadom povećava njegov fizički oblik te se pokreće animacija za taj napad pomoću funkcije Play u klasi Animator. U Update-u se provjerava dali se izvršava ijedna animacija, te ako se izvršava poziva se funkcija CheckAnimationPlaying. Unutar te funkcije se u switch grananju provjerava se animacija te se nakon njenog završetka pravilni boolean postavlja na false i vraća se prvotni Box collider. Ako se odvija animacija za smrt igrača, tada se poziva funkcija iz druge skripte.

```
void CheckAnimationPlaying()
    {
        var m_AnimatorClipInfo =
gameObject.GetComponent<Animator>().GetCurrentAnimatorClipInfo(0);

        switch (m_AnimatorClipInfo[0].clip.name)
        {
            case "PixelCharAnim_Sword_death":
                gameObject.GetComponent<Player_health>().DeathAnimation();
                break;
            case "PixelCharAnim_Sword_idle":
                if (heavyAttack)
                {
                    heavyAttack = false;
                    gameObject.GetComponent<BoxCollider2D>().size =
playerBoxColliderSize;
                }
                else if (mediumAttack)
                {
                    mediumAttack = false;
                    gameObject.GetComponent<BoxCollider2D>().size =
playerBoxColliderSize;
                }
                else if (quickAttack)
                {
                    quickAttack = false;
                    gameObject.GetComponent<BoxCollider2D>().size =
playerBoxColliderSize;
                }
                break;
        }
    }
```

Sljedeća skripta bitna za funkcioniranje igre jest Player_score zajedno s klasom DataManagement. U toj skripti odvija se zbrajanje bodova koji se steknu tijekom nivoa putem skupljanja novčića te na kraju ukupnog zbrajanja svih bodova, te ako je to highscore, zapisivanja u datoteku.

```
private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Camera_change")
        {
            var camera = GameObject.FindGameObjectWithTag("MainCamera");

            if (changedCamera == false)
            {

camera.GetComponent<Camera_system>().ChangeVerticalCameraAxis(30, 0);
                changedCamera = true;
            }
            else
            {

camera.GetComponent<Camera_system>().ChangeVerticalCameraAxis(0, 0);
                changedCamera = false;
            }
        }
    }
```

U skripti se prilikom njenog pokretanja učitavaju podaci iz statične instance klase DataManagement u objekt dana, iz kojeg izvlačimo highscore za prvi i drugi nivo. Nadalje, prilikom automatskom poziva funkcije Update unutar skripte, ako trenutni nivo nije uvodni nivo, upisujemo u objekte TimeLeftUI i PlayerScoreUI preostalo vrijeme za prelazak nivo i trenutni rezultat. Vrijeme za prelazak je predefinirano kao varijabla TimeLeft te se unutar svakog poziva Update-a smanjuje za Time.deltaTime, što je vrijeme između dva frame-a, jer se Update poziva i do 30 puta u sekundi. Ako se TimeLeft spusti do nule, igrač automatski umire i resetira se na početku nivo.

```
void Update(){
        if(SceneManager.GetActiveScene().name != "Tutorial_level"){
            TimeLeft -= Time.deltaTime;
            TimeLeftUI.gameObject.GetComponent<Text>().text = "Time Left: "
+ (int)TimeLeft;
            PlayerScoreUI.gameObject.GetComponent<Text>().text = "Score: "
+ playerScore;

            if (TimeLeft < 0.1f){
                ResetPlayerPosition();
            }
        }
}
```

Također u ovoj skripti imamo funkciju OnTriggerEnter2D u kojoj kada naiđemo na objekt provjeravamo njegov tag. Ako tag govori da je objekt novčić, tada povećavamo rezultat, a ako je objekt tipa Finish, tada svo preostalo vrijeme množimo s deset i pridodajemo trenutnom rezultatu kako bi dobili konačan rezultat. U funkciji u kojoj dobivamo finalni broj također se događa provjera trenutnog high scorea, da vidimo treba li upisivati novi rekord ili ne. Ako treba, u već postojeći dohvaćeni objekt s high scoreovima se upisuje novi high score te se poziva

funkcija za upis u postojeću datoteku. Ako ne postoji datoteka stvara se nova prilikom prvog dohvata high scorea. Također, prilikom svakog upisa rezultata nakon završetka nivoa upisuju se i trenutni postignuti rezultat i high score za taj nivo kako bi se moglo u glavnom izborniku ispisati nakon završetka nivoa postignuti rezultat. To se upisuje u novu klasu LevelScores koja ima tri statične varijable, highScore, levelScore te levelName. Nju pozivamo iz glavnog izbornika kako bismo mogli ispisati na ekran rezultate.

```
 void CountScore()
    {
        playerScore += (int)(TimeLeft * 10);

        LevelScores.levelName = SceneManager.GetActiveScene().name;
        LevelScores.levelScore = playerScore;
        switch (SceneManager.GetActiveScene().name)
        {
            case "Scena_1":
                if (playerScore > highScore_level1)
                {
                    DataManagement.dataManagement.HighScore_level1 =
playerScore;
                    DataManagement.dataManagement.SaveData();
                }
                LevelScores.highScore =
DataManagement.dataManagement.HighScore_level1;
                break;
            case "Scena_2":
                if (playerScore > highScore_level2)
                {
                    DataManagement.dataManagement.HighScore_level2 =
playerScore;
                    DataManagement.dataManagement.SaveData();
                }
                LevelScores.highScore =
DataManagement.dataManagement.HighScore_level2;
                break;
        }
    }
}

public class LevelScores
{
    public static int highScore;
    public static int levelScore;
    public static string levelName;
}
```

Zadnja i najjednostavnija skripta je Player_health. U njoj se provjerava prilikom Update-a pozicija igrača po Y osi, te ako se nalazi na minus 10, tada se igrač resetira na početnu lokaciju nivoa. Druga stvar koja se tu odvija je kontroliranje izvršavanja animacije za smrt igrača. Prvo se pozove iz drugih skripti metoda DeathAnimation gdje se postavlja boolean startDeathAnimation na true. Prilikom sljedećeg Update-a omogućen je ulazak u drugi blok

koda gdje se umanjuje predefinirano vrijeme waitDeathAnimation koje govori koliko će trajati animacija za smrt igrača. Kada to vrijeme istekne, tada se poziva nova funkcija ResetPlayerPosition koja u sebi poziva funkciju iz druge skripte, Player_score, koja igrača resetira na početak nivoa.

```
void Update()
    {
        if (gameObject.transform.position.y < -10)
        {
            ResetPlayerPosition();
        }

        if(startDeathAnimation == true)
        {
            waitDeathAnimation -= Time.deltaTime;
            if(waitDeathAnimation < 0)
            {
                ResetPlayerPosition();
            }
        }
    }

    public void ResetPlayerPosition()
    {
        gameObject.GetComponent<Player_score>().ResetPlayerPosition();
    }

    public void DeathAnimation()
    {
        var m_AnimatorClipInfo =
gameObject.GetComponent<Animator>().GetCurrentAnimatorClipInfo(0);
        gameObject.GetComponent<Player_controls>().enabled = false;
        startDeathAnimation = true;
    }
```

4.3.2. Neprijatelji

Neprijatelji se kontroliraju kroz dvije skripte, Enemy_health i Enemy_controls. Prva skripta je dosta jednostavna, te sadrži samo kontrolu neprijatelja po osi Y kako se ne bi dogodilo da neprijatelji padaju neograničeno puno po osi Y i time troše resurse računala, nego se samo preko metode Destroy uništi taj gameObject. Druga skripta je dosta kompliciranija te ona služi za kretanje neprijatelja i njegove napade. U Update metodi hit i backHit kontroliraju neprijateljev raycast u smjeru njegovog kretanja i suprotno. Ako se vidi da je neprijatelj ispod određenog razmaka od igrača, tada se ulazi u blok koda gdje se provjerava dali je igrač u napadu ili nije. Ako je igrač u napadu tada neprijatelj umire, a u suprotnom umire igrač.

```
void Update()
    {
        RaycastHit2D hit = Physics2D.Raycast(transform.position, new
Vector2(XMoveDirection, 0));
```

```
        RaycastHit2D backHit = Physics2D.Raycast(transform.position, new
Vector2(XMoveDirection * -1, 0));
        gameObject.GetComponent<Rigidbody2D>().velocity = new
Vector2(XMoveDirection, 0) * EnemySpeed;
        if (hit.distance < 1.2f || backHit.distance < 0.6f)
        {
            if (hit.collider.tag == "Player")
            {
                var player = GameObject.FindGameObjectWithTag("Player");

if(player.GetComponent<SpriteRenderer>().sprite.name.Contains("Atk"))
                {
                    EnemyDie();
                }
                else
                {
                    player.GetComponent<Animator>().Play("Sword_death");
                }
            }
        }
        if(hit.distance < 1.2f)
        {
            Flip();
        }

              CheckAnimationPlaying();

}
```

Isto tako, ako je neprijatelj manje od određene dužine udaljen od nevidljivog zida koji je definiran, tada se neprijatelj okreće i kreće u drugom smjeru do drugoga zida. Sljedeće se u funkciji provjerava u kojem dijelu animacije je neprijatelj. Ako je neprijatelj u fazi udarca, tada se preko switch naredbe aktivira za njega metoda ChangeEnemyHitSize koja povećava Box collider neprijatelja kako bi njegov imao veći doseg prilikom udarca. Kada ta animacija završi, opet preko iste switch naredbe poziva se suprotna funkcija, ReturnEnemyHitSize koja vraća Box collider u prvotnu veličinu.

```
switch(gameObject.GetComponent<SpriteRenderer>().sprite.name)
        {
            case "Player6-8":
                ChangeEnemyHitSize(2);
                break;
            case "Player6-10":
                ReturnDefaultEnemyHitSize(2);
                break;
            case "Boss1-2":
                ChangeEnemyHitSize(1.5f);
                break;
            case "Boss1-4":
                ReturnDefaultEnemyHitSize(1.5f);
                break;
            case "Boss2-3":
                ChangeEnemyHitSize(1.5f);
                break;
            case "Boss2-4":
                ReturnDefaultEnemyHitSize(1.5f);
```

```
                break;
            }
```

4.3.3. Skripta za uvodni nivo

Ova skripta nam služi za prikaz teksta u uvodnom nivou. Skripta je dosta jednostavna, te se sastoji od tri metode. Prva je trigger tipa, OnTriggerEnter2D gdje se prilikom kontakta sa svakim Box colliderom označenim kao trigger ulazi u funkciju. U njoj se na temelju imena tog objekta u switch naredbi odabire pravilan ispis poruke na ekran, odnosno instrukcija za igrača. Tada ovisno o poruci poziva se funkcija PrintOutText s proslijeđenim stringom koji se ispisuje na ekran, a metoda RemoveText ne prima parametre nego samo stavlja prazan string za ispisivanje na zaslon ekrana. Svaka poruka unutar switcha može se ispisati samo jednom zbog toga što također postoji i boolean varijabla prikadnog naziva koja je isprva postavljena na false, a prilikom ulaza switcha u taj blok koda postavlja se na true te u svakom sljedeće odabiru tog bloka koda on se neće izvršiti.

```
private void OnTriggerEnter2D(Collider2D collision)
    {
        switch(collision.gameObject.name)
        {
            case "Tutorial_move":
                if(tutorial_move == false)
                {
                    PrintOutText("Use the arrow keys to move left and 
right!");
                    tutorial_move = true;
                }
                break;
            case "Tutorial_text_jump":
                if(tutorial_jump == false)
                {
                    PrintOutText("Press space to jump!");
                    tutorial_jump = true;
                }
                      break;

public void PrintOutText(string text)
    {
        TutorialInstructions.gameObject.GetComponent<Text>().text = text;
    }

    public void RemoveText()
    {
        TutorialInstructions.gameObject.GetComponent<Text>().text = "";
}
```

4.3.4. Parallax Scrolling

Jedan od najzanimljivijih koncepata koji se koriste u igrici je koncept sadržan u ovoj skripti. Iako se ovdje radi o 2D igrici koja nema nikakvu dubinu, prividna dubina može se

postići postavljanjem pozadinskim objekata na različite dubine, odnosno na različite pozicije po osi Z. Ovisno o toj poziciji na osi Z na kojoj se nalaze naši objekti, u skripti ćemo postaviti njihovo kretanje u stranu u kojoj se kreće igrač, kako bismo dobili osjećaj približenosti, odnosno udaljenosti od pojedinih objekata, kao što su u ovoj igrici oblaci. Oblaci u igrici postavljeni su na tri razine, te je za svaku razinu stavljen poseban layer kako bi uvijek najbliži oblaci bili u glavnome planu, a najveći u pozadini. U automatskom pozivu funkcije update događa se for petlja u kojoj se za svaki oblak koji se nalazi u polju backgrounds događa izračun njegovo pomaka na temelju njegove pozicije na osi Z. Tako dobivamo prividnu dubinu jer će se najbliži elementi najviše pomicati, dok će se oni u pozadini minimalno pomicati.

```
 void Start () {
        previousCameraPosition = camera.position;

        parallaxScales = new float[backgrounds.Length];

        for(int i = 0; i < backgrounds.Length; i++)
        {
            parallaxScales[i] = backgrounds[i].position.z*-1;
        }
}

void Update () {
            for(int i = 0; i < backgrounds.Length; i++)
        {
            float parallax = (previousCameraPosition.x - camera.position.x)
* parallaxScales[i];

            float backgroundTargetPositionX = backgrounds[i].position.x +
parallax;
            Vector3         targetBackgroundPosition         =         new
Vector3(backgroundTargetPositionX,               backgrounds[i].position.y,
backgrounds[i].position.z);
            backgrounds[i].position = Vector3.Lerp(backgrounds[i].position,
targetBackgroundPosition, smoothing * Time.deltaTime);
        }
        previousCameraPosition = camera.position;
}
```

4.3.5. Glavni izbornik

Skripta namijenjena početnoj sceni preko koje se ulazi u sve nivoe te izlazi iz aplikacije. Na Canvas objekt na sceni za glavni izbornik smo stavili određenu dugmad koju smo ovdje povezali s određenim značenjima. Kada se klikne na dugme za bilo koji nivo tada se pokreće određena scena, te ako se klikne na izlaz iz aplikacije zatvara se cijeli program. Također, u ovoj se skripti provjerava prilikom njenog početka dali je nivo tek završio ili se u skriptu ulazi tek po prvi put. Ako se otvara scena nakon završetka nivoa, tada će se ispisati rezultat nivoa i

poruka o novom high scoreu ako ga ima na ekran na temelju static varijabli iz klase LevelScores, koje upisujemo u skripti Player_score. Na kraju ispisa rezultata postavljaju se varijable na null kako bi se izbjeglo njihovo nepotrebno ponovno ispisivanje.

```
private void Start()
    {

        if (LevelScores.levelName != null)
        {
            var gameScoresText =
GameObject.FindGameObjectsWithTag("GameScores");

            switch (LevelScores.levelName)
            {
                case "Scena_1":
                    gameScoresText[0].GetComponent<TextMeshProUGUI>().text
= "Score: " + LevelScores.levelScore;
                    if (LevelScores.levelScore == LevelScores.highScore)
                    {

gameScoresText[0].GetComponent<TextMeshProUGUI>().text += "\nNew
Highscore!!!";
                    }
                    break;
                case "Scena_2":
                    gameScoresText[1].GetComponent<TextMeshProUGUI>().text
= "Score: " + LevelScores.levelScore;
                    if (LevelScores.levelScore == LevelScores.highScore)
                    {

gameScoresText[1].GetComponent<TextMeshProUGUI>().text += "\nNew
Highscore!!!";
                    }
                    break;
            }
            LevelScores.levelName = null;
        }
    }

public void PlaySecondLevel()
    {
        LoadScene("Scena_2");
    }

    public void LoadScene(string scene)
    {
        SceneManager.LoadScene(scene);
    }

    public void ExitButton()
    {
        Application.Quit();
    }
```

4.3.6. Skripta za kontrolu kamere

Kamera je jedan od bitnih segmenata igre jer je dosta važno da prati igrača u koraku kroz njegov prolazak kroz nivoe. U ovoj skripti postoje četiri varijable, xMin i xMax, yMin i yMax. Varijable su public kako bi im se moglo pristupiti preko samog editora i tako ih mijenjati u pokretu. Prilikom starta skripte dobije se objekt s tagom Player, što označava našeg glavnog igrača te se taj objekt koristi prilikom svakog update u funkciji kako bi mu se pridonijele definirane varijable. Ako nemamo tako velikog micanja po vertikalnoj osi kao što je to slučaj u uvodu ili u prvom nivou, tada yMin i yMax možemo ostaviti na nuli, dok kod drugog nivoa gdje imamo prave oznake platformera moramo micati yMax kako bi kamera pratila igrača se on vertikalno pomiče, zbog čega imamo funkciju ChangeVerticalCameraAxis.

```
void Update()
    {
        float x = Mathf.Clamp(player.transform.position.x, xMin, xMax);
        float y = Mathf.Clamp(player.transform.position.y, yMin, yMax);
        gameObject.transform.position = new Vector3(x, y,
gameObject.transform.position.z);

    }

    public void ChangeVerticalCameraAxis(int yMax, int yMin)
    {
        this.yMax = yMax;
        this.yMin = yMin;
}
```

Ta funkcija se poziva iz skripte Player_controls kada se pomoću triggera očita da smo prošli nevidljivi Box collider objekt s posebnom oznakom. S time ide i boolean varijabla kojom znamo trebamo li vratiti prvotne postavke kamere ili promijeniti ih. Ako je boolean changedCamera postavljen na false, tada ćemo dodati na yMax određeni broj kako bi se kamera po vertikali kretala s nama, odnosno ako je postavljen na true, tada ćemo vratiti kameru u prvotno stanje, što zapravo označava da ulazimo, odnosno izlazimo iz područja u kojem moramo kontrolirati kameru po Y osi.

5. Zaključak

Iako u današnje vrijeme više nisu toliko zastupljeni kao samostalne igre, platformeri su jedan od najbitnijih žanrova u povijesti jer se upravo zbog njih događao najveći napredak u smislu igara zbog čega se i danas javljaju u određenim segmentima u svim igrama.

Nastojao sam u ovoj igrici postići upravo te stvari zbog kojih su prvotno postali popularni, zbog jednostavnosti i velike igrivosti bez prevelike krivulje učenja. Neki koncepti koji su nastali u to doba danas se još uvijek koriste u velikoj mjeri i koji su prisutni i ovdje, ali u jednostavnijem obliku. Iako izgleda dosta jednostavno, postoje dosta stvari u pozadini koje se ne vide, ali upotpunjuju cjelokupnu igru, kao što je povećanje Box collidera kod igrača ili kod neprijatelja za vrijeme napada te njihovo ponovno vraćanje u prvotno stanje na kraju odvijanja te animacije. Također bitan element koji daje dobar dojam je i parallax scrolling za dobivanje prividne dubine.

Planovi za budućnost igre je dorađivati mehaniku i implementirati dodatne mogućnosti napada na neprijatelje u pogledu drugog oružja ili sličnih alata. Postoje dodatne animacije kao što je trčanje po zidovima, uklizavanje ili čak napadi s pištoljem koje nisu implementirane, za koje smatram da bi donijele stvarne novitete i podigle igricu na višu razinu. Također, napadi neprijatelja ili određeni bitniji i veći neprijatelji, bossovi, su planovi u budućnosti za implementirati.

Iznimno sam zadovoljan s postignutim u ovom projektu te se nadam kako ću nastaviti raditi u programskom okruženju Unity i dalje u budućnosti. Htio bih probati napraviti 3D igricu u kojoj će igrač trčati automatski izbjegavajući prepreke koje nastaju nasumično na ekranu, gdje će se isto skupljati što veći rezultat. Također, usporediti taj rezultat s prijateljima putem interneta, ili tablice rezultata unutar igre.

6. Literatura

[1] Red Bull (2017) *The evolution of platform games in 9 steps*. Preuzeto 14. kolovoza. 2018. s https://www.redbull.com/in-en/evolution-of-platformers

[2] Worchester Polytechnic Institute (2018) *A History Of The Unity Game Engine*. Preuzeto 27. kolovoza 2018. s https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf

[3] Unity3D (2018) *Learn*. Preuzeto 27. kolovoza 2018. s https://unity3d.com/learn

[4] AssetStore.com (2018) *Unity Asset Store*. Preuzeto 27. kolovoza 2018. s https://assetstore.unity.com/

[5] sonichehedgehog.com (2018) *Sonic The Hedgehog.* Preuzeto 27. kolovoza 2018. s https://www.sonicthehedgehog.com/

[6] supermarionrun.com (2018) *Super Mario Run.* Preuzeto 27. kolovoza 2018. s https://supermariorun.com/en/index.html?n

[7] donkeykong.com (2018) *Donkey Kong.* Preuzeto 27. kolovoza 2018. s https://donkeykong.nintendo.com/tropical-freeze/

[8] mariowiki.com (2018) *Super Mario World 2: Yoshi's Island.* Preuzeto 27. kolovoza 2018. s https://www.mariowiki.com/Super_Mario_World_2:_Yoshi%27s_Island

[9] capcom.com (2018) *Mega Man.* Preuzeto 27. kolovoza 2018. s http://megaman.capcom.com/

[10] arcade-museum-com (2018) *Congo Bongo.* 27. kolovoza 2018. s https://www.arcade-museum.com/game_detail.php?game_id=7384

[11] generation-msx.nl (2018) *Antarctic Adventure.* Preuzeto 27 kolovoza 2018. s https://www.generation-msx.nl/software/konami/antarctic-adventure/25/

[12] archive.org (2018) *Space Panic*. Preuzeto 27. kolovoza 2018. s https://archive.org/details/arcade_panic

[13] banjoo-kazooie.wikia.com (2018) *Banjoo-Kazooie.* Preuzeto 27. kolovoza 2018. s http://banjokazooie.wikia.com/wiki/Banjo-Kazooie

[14] crashbandicoot.com (2018) *Crash Bandicoot.* Preuzeto 27. kolovoza 2018. s https://www.crashbandicoot.com/

[15] tombraider.wikia.com (2018) *Tomb Raider*. Preuzeto 27. kolovoza 2018. s http://tombraider.wikia.com/wiki/Tomb_Raider_(1996_Game)

[16] unchartedthegame.com (2018) *Uncharted: The Lost Legacy*. Preuzeto 27. kolovoza 2018. s https://www.unchartedthegame.com/en-us/

[17] godofwar.playstation.com (2018) *God Of War*. Preuzeto 27. kolovoza 2018. s https://godofwar.playstation.com/

[18] cupheadthegame.com (2018) *Cuphead.* Preuzeto 27. kolovoza 2018. s http://www.cupheadgame.com/

[19] hollowknight.com (2018) *Hollow Knight.* Preuzeto 27. kolovoza 2018. s http://hollowknight.com/

[20] spyrothedragon.com (2018) Spyro The Dragon. Preuzeto 27. kolovoza 2018 s https://www.spyrothedragon.com/

[21] gameranx.com (2018) *Super Mario Odyssey*. Preuzeto 28. kolovoza 2018. s http://gameranx.com/wp-content/uploads/2017/10/MarioOdyssey3.jpg

[22] xboxachievements.com (2018) *Crash Bandicoot Review*. Preuzeto 28. kolovoza 2018. s https://www.xboxachievements.com/images/screenshots/5062/med_image%20(1).jpg

[23] gamersheroes.com (2018) *Uncharted 4: A Thieves End Review*. Preuzeto 28. kolovoza 2018. s http://www.gamersheroes.com/wp-content/uploads/2016/05/Uncharted-4-Founders-Puzzle-Guide.jpg

7. Popis slika