Izrada 2D platformske videoigre u programskom alatu Unity

Zović, Mateo

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:120447

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mateo Zović

IZRADA 2D PLATFORMSKE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mateo Zović

Matični broj: 0016155144

Studij: Informacijski i poslovni sustavi – Umreženi sustavi i računalne igre

IZRADA 2D PLATFORMSKE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, srpanj 2024.

Mateo Zović

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

Rad obuhvaća izradu 2D platformske videoigre koristeći se programskim alatom Unity. Izrađena videoigra je platformer žanra i upravo zato je stavljen naglasak na izradu kretanja igrača kroz prostor. Uz kretanje, kao osnovnu mehaniku videoigre, javlja se i implementacija svjetla. Svjetlost je glavni i nezaobilazni element videoigre jer se cijela igra nalazi u mračnom okruženju. Kako bi igrač mogao vidjeti razinu na kojoj se nalazi, on zrači sa svjetlošću i na taj način osvjetljava sam sebi put. Međutim, prilikom kretanja u mraku, svjetlost igrača se troši, te je potrebno pronaći izvore svjetla na razini, kako bi igrač vratio izgubljenu svjetlost. U radu su prvo istaknuti programski alati i internetske stranice korištene za izradu i dobavljanje resursa videoigre. Zatim je objašnjen pojam žanra platformskih videoigara te naposljetku je dan primjer kako izraditi vlastitu videoigru. Kod izrade videoigre pretežno je stavljen naglasak na izradu programske logike. Uz programiranje, javljaju se i elementi dizajnerskih vještina, kao i tehnike primjene kreativnosti prilikom izrade same videoigre. Naposljetku, iz svega navedenog, može se zaključiti da je proces izrade videoigre vrlo zahtjevan i opsežan zadatak, ali na kraju, za pojedinca ovakva vrsta zadatka može predstavljati samo jedan novi izazov u vlastitoj karijeri.

Ključne riječi: platformer; videoigra; Unity; mehanike; kretanje; svjetlost; mrak; programiranje;

Sadržaj

1. Uvod

Tema završnog rada je predstaviti koncepte i dizajn 2D platformskih videoigara, služeći se programskim alatom Unity. Najvažnija karakteristika ovog žanra videoigara jest u kretanju igrača i savladavanju prepreka na koje igrač naiđe. Razine, odnosno leveli igre su sastavljeni pretežito od raznih platformi, po čemu je i sam žanr dobio ime. Jedan od osnovnih koncepata ovog rada bit će korištenje osvjetljenja u videoigri, pa samim time razine bi se nalazile u mračnom okruženju. Igra je zamišljena da igrač sa prikupljenom svjetlosti osvjetljava put i na taj način prelazi različite prepreke na koje putem naiđe. Polazna motivacija za odabirom izrade videoigre je ta što videoigra objedinjuje široki spektar znanja stečenog na fakultetu. Izradom videoigre objedinjuju se vještine izrade korisničkog sučelja (eng. *User Interface*), korisničkog iskustva (eng. *User Experience*), multimedije te programske vještine. Cilj korisničkog sučelja je privući pažnju korisnika, dok je cilj dobrog korisničkog iskustva zadržati korisnika. Uloga multimedije je da spoji slike i zvuk pritom kreirajući potpuno korisničko iskustvo korištenja aplikacije, ili u ovom slučaju, igranju videoigre. Programska logika je središte svake aplikacije jer ona povezuje sve elemente izrađene aplikacije. Programske vještine su ključne i preko potrebne za izradu bilo kakvog informatičkog proizvoda. Naposljetku, bitno je napomenuti da je proces izrade videoigre vrlo zahtjevan i zahtjeva dobru raspodjelu vremena. Upravljanje raspoloživim vremenom spada u organizacijsku vještinu, pa samim time ovaj projekt također uključuje i znanje stečeno na organizacijskim kolegijima.

2. Metode i tehnike rada

Istraživačke aktivnosti koje su provedene za razradu teme uključuju proučavanje videoigara koje su slične žanrom ili mehanikama igranja igre. Platforma koja je korištena za proučavanje tih videoigara jest ponajviše YouTube. Platforma za razvoj videoigre je Unity [1], koja u kombinaciji s platformom Microsoft Visual Studio [7] omogućuje implementaciju programske logike videoigre. YouTube je također korišten za dobivanje i pronalazak osnovnih ideja i koncepata implementacije koda za ostvarivanje određenih funkcionalnosti unutar videoigre. Nadalje, metode i tehnike rada obuhvaćaju istraživanje i odabir odgovarajućih resursa videoigre (eng. *asset*), uključujući vizualne i audio resurse. Platforme korištene za pronalazak resursa videoigre jesu Unity Asset Store [8], Itchi.io [9], GameDev Market [10], FontSpace [11] i Pixabay [12]. Navedene platforme bit će nešto detaljnije opisane u nastavku.

2.1. Programski alat Unity

Programski alat Unity [1] jedan je od tri danas najpopularnija programska alata za razvoj videoigara. Ostala dva alata jesu Unreal Engine i Godot. Karakteristike sva tri programska alata jesu da pokrivaju razvoj 2D i 3D videoigara, međutim Unreal Engine se primarno koristi za razvoj 3D videoigara, zbog širokih mogućnosti rada s visokom kvalitetom grafike videoigre. Godot je najnoviji programski alat od tri navedena, a karakteristika mu je što je vrlo jednostavan i pojednostavljen. Zauzima vrlo malo prostora na računalu, a primarno je namijenjen za izradu jednostavnih 2D videoigara. U usporedbi s prethodno dva navedena programska alata, Unity pruža jednako dobru podlogu za razvoj 2D i 3D videoigara. Unity također nudi mogućnost izrade videoigra za Android uređaje, od kojih su daleko najpoznatije videoigre kreirane pomoću Unity alata Pokemon GO, Subway Surfers, Crossy Road, Temple Run i druge. Iako Unity podržava izradu 2D i 3D videoigara, primarno se koristi za razvoj 2D videoigara, i videoigara namijenjenih za Android uređaje. Na sljedećoj slici je prikazana detaljna usporedba sva tri programska alata za razvoj videoigara.

Feature	Unity	Unreal Engine	Godot
Scripting and language	C#, JavaScript	C++, Blueprints	GDScript, C#, C++
2D capabilities	Strong	Excellent	Exceptional
3D capabilities	Robust	Industry-leading	Good
Graphics rendering	PBR (Physically Based Rendering)	PBR (Physically Based Rendering)	PBR (Physically Based Rendering)
Cross-platform support	Windows, MacOS, Linux, Android, iOS, and Web	Windows, MacOS, Linux, Android, and iOS	Windows, MacOS, Linux, and Web
Pricing model	Freemium with paid tiers for additional features	Freemium with a 5% royalty fee for games earning over one million USD	Completely free and open-source

Slika 1: Usporedba programskih alata za razvoj videoigara (Unity, Unreal Engine, Godot) (izvor: [5])

„Unity 1.0 je objavljen u lipnju, 2005. godine“ [4], što se može smatrati njegovim samim početkom. Danas se Unity koristi diljem svijeta te je jedan od najpopularnijih platformi za razvoj videoigara. Kako je Unity dosegao toliku popularnost, proizlazi iz činjenice da je besplatan i dostupan svima te koristi mnoštvo ugrađenih funkcija i naredbi koje olakšavaju njegovu uporabu. Također Unity je razvio svoju vlastitu digitalnu trgovinu resursa (eng. *Unity Asset Store*), gdje pojedinci mogu objavljivati i preuzimati različite resurse za videoigru. Samim time, Unity je kroz vrijeme izgradio široku zajednicu, koja je najčešće od velike pomoći pojedincima koji se tek upuštaju u područje razvoja videoigara.

Slika 2: Logotip programskog alata Unity (izvor: [6])

2.2. Microsoft Visual Studio

Microsoft Visual Studio [7] je integrirano razvojno okruženje (eng. *Integrated Development Environment – IDE*), koje se koristi za razvoj računalnih programa. Računalni programi u ovom slučaju obuhvaćaju web stranice, web aplikacije, desktop i mobilne aplikacije. Koristi se za pisanje koda u jeziku C, C++, C#, JavaScript, HTML, CSS i drugi. U ovom radu, koristi se verzija Microsoft Visual Studio 2022 za pisanje skripti u jeziku C#.

2.3. Digitalna trgovina resursa (eng. *Asset Store*)

Digitalna trgovina resursa omogućuje brzo i jednostavno dohvaćanje potrebnih resursa za razvoj videoigre. U resurse spadaju svi elementi koji doprinose razvoju igre. Najčešće pod resurse spadaju slike, pozadine, objekti, animacije i zvukovi. Za izradu rada korištene su tri digitalne trgovine resursa. Unity Asset Store [8] za pronalazak većine vizuala videoigre, te Itch.io [9] i GameDev Market [10]. Za pronalazak odgovarajućeg fonta videoigre korištena je web stranica FontSpace [11] koja nudi mnoštvo različitih fontova i pisma. Pozadinska glazba i zvukovi su preuzeti sa stranice Pixabay [12].

S obzirom na sve navedeno, bitno je istaknuti da svi vizualni i audio zapisi videoigre nisu kreirani ručno, već su preuzeti s navedenih web stranica.

3. Razrada teme

Tema rada je podijeljena u tri poglavlja, u prvom poglavlju bit će prikazani karakteristični predstavnici žanra videoigre koji se izrađuje. Uloga ovog poglavlja je da čitatelja uvede u osnovne koncepte i mehanike platformer videoigara. Drugo poglavlje ima za ulogu prikazati sučelje programskog alata Unity. Od toga kako se kreira novi projekt, do početnog prozora sučelja i osnovnih opcija koje su nezaobilazne prilikom kreiranja bilo kakve videoigre. Treće i najopsežnije poglavlje opisuje izradu 2D platformske videoigre. Ovdje će biti prikazani objekti scene, vizuali, animacije, audio zapisi, a poseban naglasak će biti stavljen na programsku logiku videoigre.

3.1. Predstavnici žanra platformer videoigara

Za početak prvo je potrebno definirati što bi to bio žanr videoigre. „Žanrovi računalnih igara su specifične kategorije igara koje su slične po mehanikama igranja.“ [13] Tako na primjer borbene videoigre (npr. Mortal Kombat) imaju mehanike posebnih udaraca, najčešće se igrač bori protiv drugog igrača, te se igra nekoliko runda dok ne istekne vrijeme ili jedan od igrača ne bude poražen. Igre preživljavanja (npr. Minecraft) najčešće imaju mehaniku skupljanja stvari i predmeta iz okoline te izgradnja novih, kompleksnijih predmeta i objekata. Ovaj žanr se ističe po tome što je igrač najčešće smješten u okolini otvorenog svijeta, a sam cilj igre je preživjeti što duže u takvom svijetu punom prepreka i izazova. Žanr platformer videoigara, također ima svoje karakteristične mehanike igranja. „Glavne karakteristike ovog žanra su korištenje skakanja i penjanja kako bi se igrač kretao okolinom u kojoj se nalazi kako bi ostvario određene ciljeve. Razine i okoline imaju neravan teren i platforme (često pomične) na različitim visinama.“ [13] Ovaj žanr zahtjeva od igrača određenu razinu spretnosti upravljanja kretanjem. U nastavku slijede poznatije platformer videoigre, koje su ukratko opisane te su predstavljene osnovne mehanike i cilj igranja same videoigre.

3.1.1. Super Mario Bros.

Kada se govori o platformer videoigrama teško je zaobići igru Super Mario Bros. Super Mario Bros. jedna je od najpoznatijih i općenito jedna od prvih videoigra platformer žanra. „Objavljena je 1985. godine (18. listopada) od kompanije Nintendo za platformu Nintendo Entertainment System.“ [14] Nintendo Entertainment System je platforma za igranje videoigra koja se sastoji od konzole u kojoj bi se učitavala videoigra te od kontrolera kojim bi se upravljalo kretanjem lika. Na slici ispod [Slika 2] prikazani su konzola i kontroler platforme putem kojem se videoigra Super Mario Bros. mogla igrati.

Slika 3: Nintendo Entertainment System - Prikaz konzole i kontrolera (izvor: [26])

Glavna radnja Super Mario Bros. videoigre je da Mario čuje za nevolju kraljevstva Gljiva, te pokušava spasiti princezu koja je zarobljena kod kralja Koopa, Bowsera. Kraljevstvo Koopa, odnosno kraljevstvo Kornjača je poznato po svojoj crnoj magiji, koju su iskoristili da napadnu kraljevstvo Gljiva.

Slika 4: Super Mario Bros. poster (izvor: [14])

Videoigra je smještena u 2D prostoru, gdje igrač prelazi razne prepreke na koje naiđe putem. Na razinama se pojavljuju različiti neprijatelji, koji, ako ih Mario dotakne, oduzimaju život igraču. Igrač može skupiti dodatne živote tako što skuplja novčiće po razinama te kad skupi dovoljnu količinu novčića, dobiva dodatan život. Na pojedinim razinama igre pronalaze se predmeti koji igraču otključavaju neku novu mehaniku igranja videoigre. To su super gljiva, od koje igrač postaje veći i ima mogućnost izdržati jedan udarac od neprijatelja bez da izgubi život. Vatreni cvijet omogućuje igraču da ispucava vatrene kugle. Naposljetku, igrač može naići na žutu zvijezdu koja ga čini privremeno neuništivim. Još jedan zanimljivi element videoigre je, kako je Mario ustvari vodoinstalater, on kroz igru pronalazi zelene cijevi u koje može uči i otkriti neke nove razine i tajne prostore. Igra završava tako što Mario pobjeđuje Bowsera, kralja Koopa te oslobađa princezu Gljiva.

Slika 5: Super Mario Bros. - Izgled videoigre (izvor: [14])

3.1.2. Sonic the Hedgehog

Još jedan od povijesno popularnih naslova platformer videoigara jest Sonic the Hedgehog. „Sonic je objavljen 23. lipnja 1991. i odmah je postao hit. Igra je bila toliko popularna da je postala prva „killer app“ konzole Genesis.“ [15] Genesis je ustvari platforma za igranje videoigri kompanije Sega. Nalik na Nintendovu konzolu, i Segina konzola se sastojala od konzole i kontrolera.

Slika 6: Sega - Genesis konzola i kontroler (izvor: [27])

Osnovna mehanika Sonic the Hedgehog igre je prolaziti razine što je brže moguće, pritom skupljajući prstenove. Prstenovi štite igrača od neprijatelja i prepreka, a ako se igrač sudari s nečime od navedenog tada gubi prstenove. Videoigra je smještena u 2D prostoru, gdje se glavni igrač bori protiv Dr. Eggman-a i njegovih zlih izuma. Igra je stekla svoju popularnost zato jer su razine dizajnirane uzimajući u obzir brzinu kretanja igrača. Na taj način, igrač se može uz dovoljno brzine penjati po zidovima i prolaziti kroz staze koje su dizajnirane u obliku petlje.

Slika 7: Sonic the Hedgehog poster (izvor: [15])

3.1.3. Broforce

Broforce je jedna od platformer videoigara koja objedinjuje osnovne mehanike platformera – kretanje, s dodatnim mehanikama poput pucanja i mogućnost uništenja cijele razine. Igra je objavljena 2015. godine te se odvija u 2D prostoru, gdje su likovi igre poznati akcijski heroji iz filmova, koji se bore protiv terorista i naposljetku, protiv vanzemaljaca i čudovišta iz pakla. Ova videoigra nudi mnoštvo različitih mehanika pucanja jer svaki novi lik, dolazi s novim oružjem. Novi likovi se otključavaju tako što se spašavaju zarobljenici na razini. Igru može igrati najviše četiri igrača odjednom, što ju čini još zabavnijom i zanimljivijom. Ukratko rečeno, Broforce je vrlo dinamična platformer videoigra s elementima pucanja, gdje se sa svakim otključanim novim likom otključava i nova mehanika igranja igre.

Slika 8: Broforce - Prikaz videoigre (izvor: [28])

3.1.4. Hollow Knight

Hollow Knight je igra razvijena od samo dvojice ljudi, objavljena 2017. godine, a danas je jedna od svjetski poznatih 2D platformer videoigara. Igra pruža elemente skakanja i kretanja po raznim razinama i platformama, zatim pruža elemente borbe, ima duboku priču te također pruža igraču mogućnost nadogradnje i otkrivanje novih mehanika igranja. Jednom kada je sve prethodno navedeno dobro i smisleno ukomponirano, kod igrača se pobuđuje potpuno iskustvo igranja. Igra Hollow Knight je zanimljiva širokoj publici igrača iz razloga što nije potpuno linearna. Igrač ima slobodu kretanja po mapi, i ima mogućnost „izgubiti se“ u svom tom prostoru. Kako bi igrač došao do određenih dijelova mape mora otključati nove mehanike

igranja. Time se kod igrača ostvaruje kombinacija osjećaja znatiželje i oduševljenja jednom kada se otkrije nova mehanika igre. Igra je vrlo bogata sadržajem, koja kombinira istraživanje, borbu, platformer elemente, duboku priču i odgovarajuće audio sadržaje.

Slika 9: Hollow Knight - Prikaz videoigre (izvor: [29])

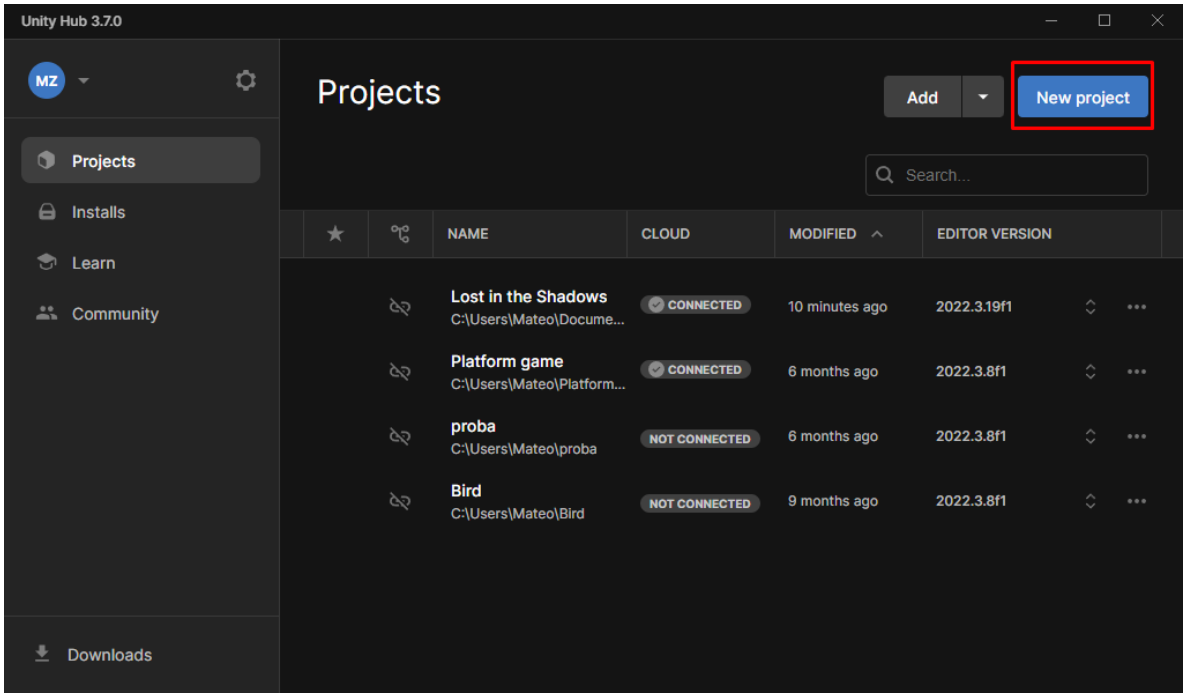
Još je potrebno istaknuti da je ova videoigra nastala u programskom alatu Unity, te da će se izrada videoigre ovog rada najviše temeljiti na igri Hollow Knight. Razlog tome je što igrač kretanjem po razinama otključava nove mogućnosti kretanja kroz prostor. U početku, to je samo hodanje i skakanje, a kasnije igrač pronalazi mehaniku naleta (eng. *Dash*), mehaniku duplog skoka i mehaniku skakanja po zidovima. Kombinacijom svih ovih vrsta kretanja igrač postaje nezaustavljiv i može pristupiti svakom dijelu mape. Slična ideja bit će i kod izrade ove igre, igrač se slobodno kreće po razini, pritom otkrivajući nove mehanike kretanja kroz prostor.

3.2. Sučelje programskog alata Unity

U ovom poglavlju bit će objašnjeno kako započeti s novim projektom u programskom alatu Unity. Nadalje, bit će predstavljene osnovne i nezaobilazne opcije kod kreiranja bilo kakvog projekta te prikaz početnog sučelja i svih njegovih bitnih komponenti i opcija.

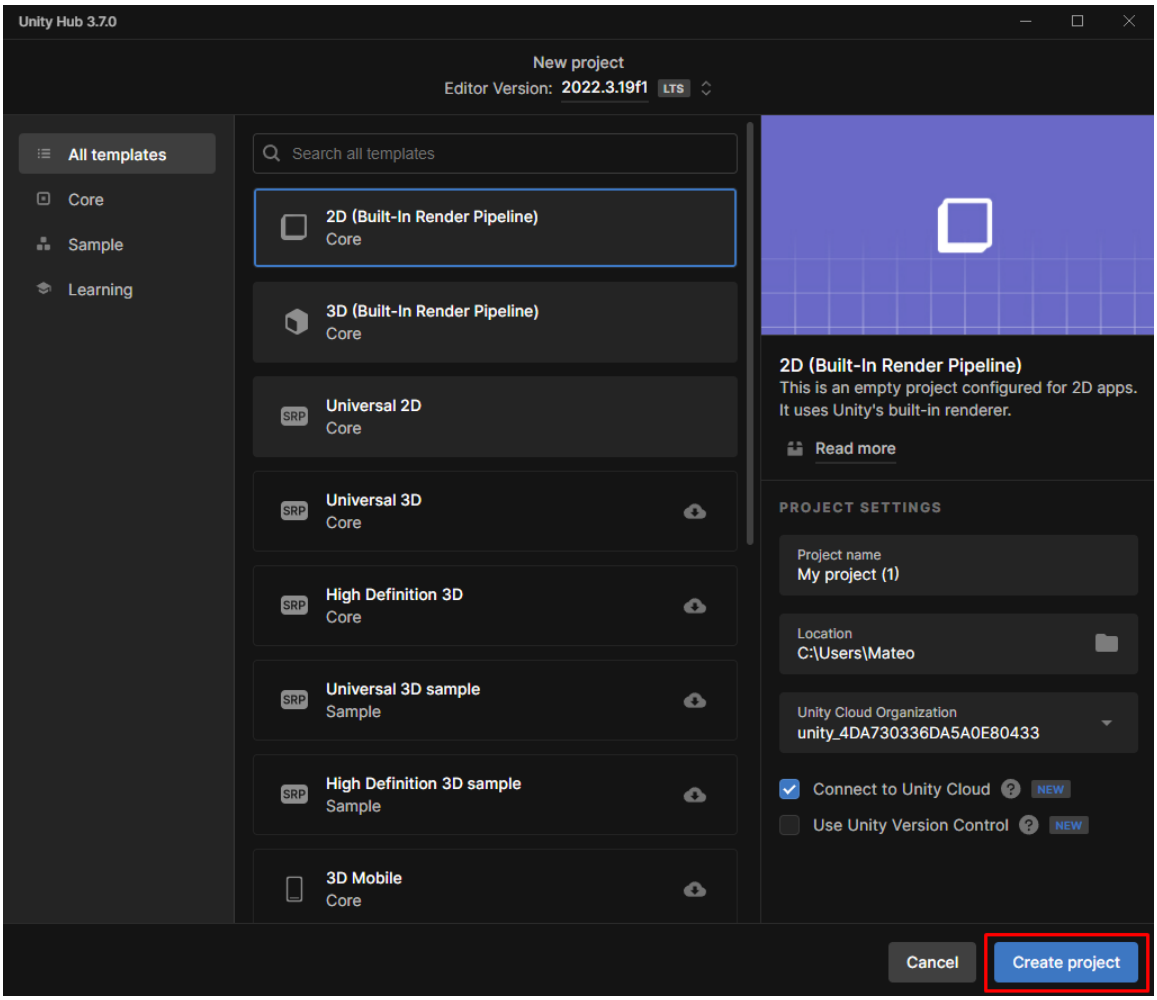
3.2.1. Kreiranje novog projekta

Za početak, potrebno je otvoriti Unity Hub aplikaciju. Otvorena aplikacija u središtu nudi prikaz svih prethodno izrađenih projekata. Na lijevoj strani sučelja korisnik može pregledavati objave zajednice, saznati nešto novo o Unity Editoru ili pak može u sekciji Installs preuzeti novije verzije Unity Editora. Za kreiranje novog projekta korisnik mora pritisnuti plavo obojenu opciju „New project“ kao što je označeno na sljedećoj slici [Slika 9].



Slika 10: Kreiranje novog projekta – Prvi korak

Zatim se na zaslonu korisnika pojavljuje novo sučelje koje traži od korisnika da odabere način renderiranja scene videoigre (eng. *Render Pipeline)* [Slika 10].



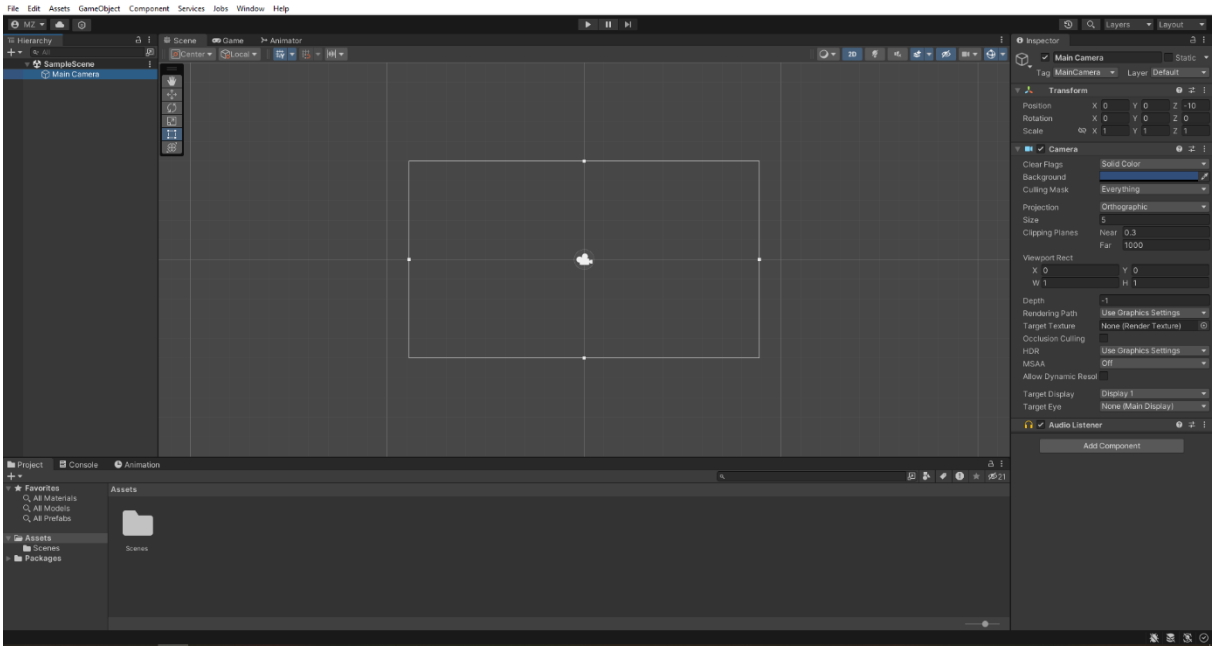
Slika 11: Kreiranje projekta - Drugi korak

Korisniku su dostupne na odabir različite vrste renderiranja. „Renderiranje izvodi niz operacija koje preuzimaju sadržaj scene i prikazuje ih na ekranu.“ [18] Prvo što korisnik mora odlučiti jest želi li raditi na 2D ili 3D videoigri te hoće li se ta igra pokretati na računalu ili pametnom mobitelu. Zatim se mora odlučiti koliko će zahtjevnu grafiku videoigre imati i prema tome bira želi li običan sustav renderiranja scene i objekata ili neki napredniji. U većini slučajeva upravo je običan sustav renderiranja scene optimalno rješenje.

U desnoj sekciji sučelja pojavljuje se kratko objašnjenje odabranog sustava renderiranja scene. Tu se, također unose informacije novog projekta, poput naziva, mjesto spremanja projekta i mogućnost uključivanja nekih dodatnih opcija kao opcija povezivanja na Unity Cloud i opcija korištenja Unity Version Control. Kada je korisnik zadovoljan s odabranim, tada pritišće na plavi gumb kreiranje projekta (eng. *Create project*). Nakon nekog vremena, dok se projekt postavi, otvara se početno sučelje Unity Editora čime završava proces kreiranja novog projekta.

3.2.2. Prikaz početnog sučelja Unity Editora

Prilikom prvog pokretanja novo kreiranog projekta, prikaz početnog sučelja izgledat će nešto slično sučelju prikazanom na slici [Slika 11].

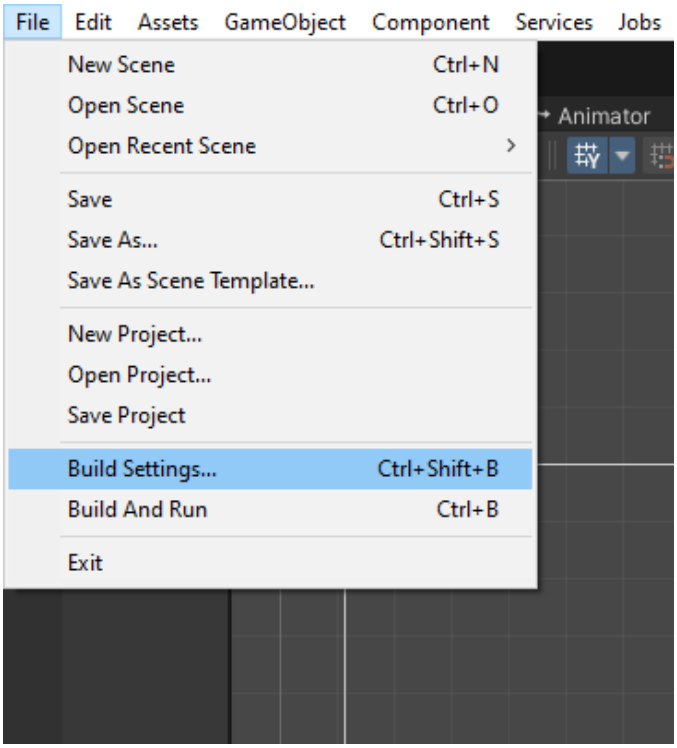


Slika 12: Prikaz početnog sučelja Unity Editora

Početno sučelje Unity Editora sastoji se od nekoliko osnovnih komponenti to jest dijelova koji ubrzavaju i olakšavaju rad na projektu. To su opcija za pregled scene, opcija game, animator, inspektor, projekt, konzola, animacije te opcija pregleda hijerarhije objekata. Opcija za pregled scene nalazi se na pri vrhu sučelja, a služi za pregled svih objekata na sceni. Ukoliko korisnika zanima kako bi se svi ti elementi prikazivali u videoigri tada jednostavno odabirom opcije Game, korisnik ima pregled igre iz perspektive igrača. Prema tome može se onda reći da je opcija Scene pogled iz perspektive programera (eng. *developer*). Opcija Animator služi za postavljanje veza između različitih animacija. Dakle, jednom kada se u opciji Animation (na dnu sučelja) kreira više animacija određenog objekta scene, tada se te animacije mogu međusobno jednostavno povezati u sekciji Animator. Popis svih objekata na sceni može se vidjeti na lijevoj strani sučelja. Ova sekcija se zove hijerarhija (eng. *Hierarchy*). Kod ovog djela, bitno je istaknuti da se cijela struktura Unity sustava radi na principu objekata. Nadalje, tim se objektima dodaju određena svojstva ili proizvoljno i drugi objekti, podobjekti. Ta obilježja i svojstva određenog objekta vidljiva su na desnoj strani sučelja u sekciji Inspector. Na prikazanoj slici [Slika 11] označen je objekt glavne kamere te se mogu uočiti neka njegova svojstva. To su svojstvo Transform koje određuje poziciju i položaj objekta na sceni, zatim kamera koja definira boju, veličinu, udaljenost i ostala svojstva, te Audio Listener, koji služi za

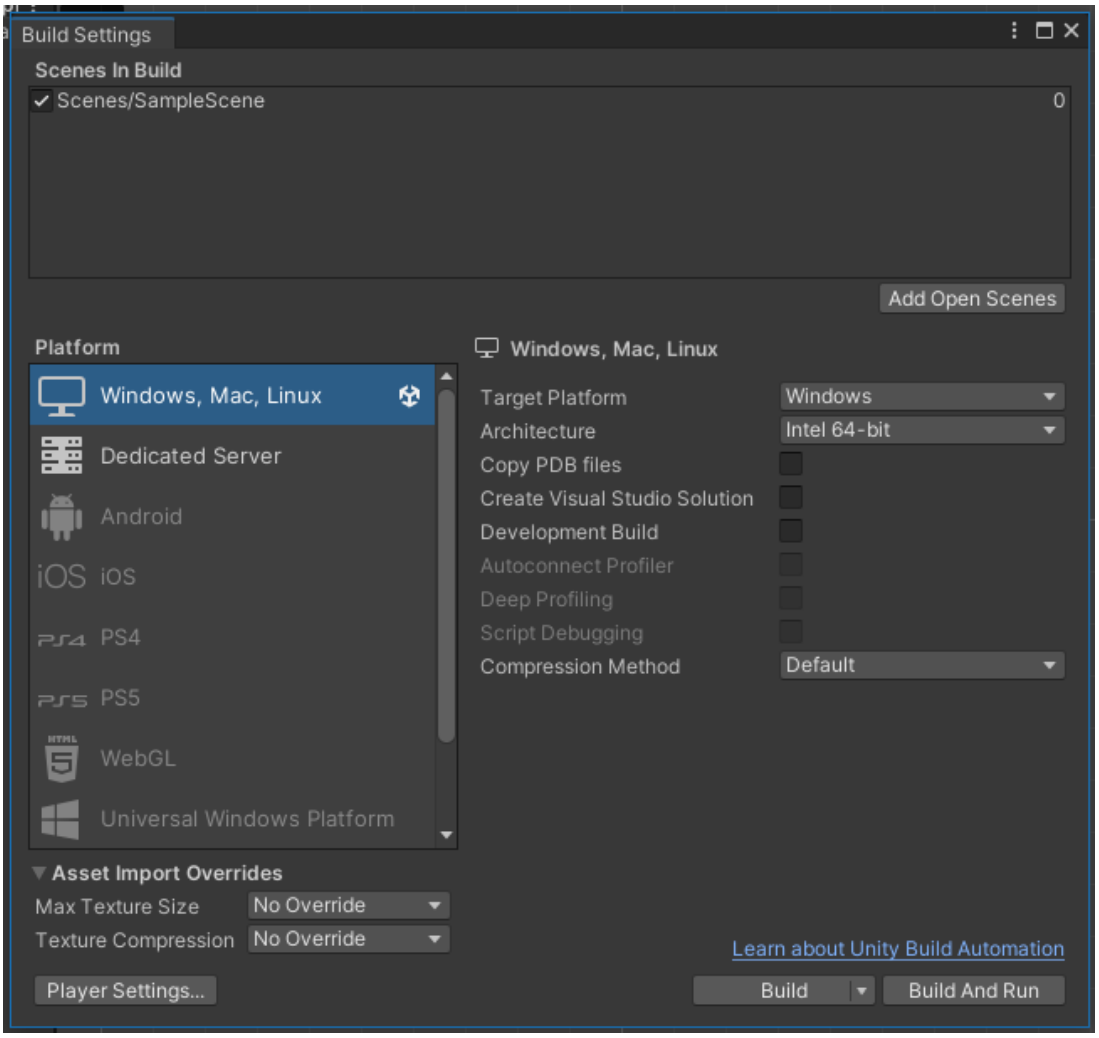
rad sa zvučnim zapisima videoigre. Na dnu sučelja osim opcije Animation nalaze se i opcije Project i Console. Projekt služi za pregled svih resursa videoigre i za grupiranje sadržaja u mape. Ovdje se nalaze svi vizualni elementi videoigre, audio zapisi, teksture, animacije, skripte i drugo. Opcija konzole prikazuje ima li kakvih logičkih grešaka u projektu i služi za pronalazak grešaka (debugiranje) prilikom izvođenja same videoigre.

Još jedna bitna opcija Unity Editora je kako izvesti/izgraditi videoigru jednom kada je kreirana. U tom slučaju korisnik klikom na opciju File (u lijevom gornjem kutu) otvara padajući izbornik iz kojeg odabire opciju Build Settings.



Slika 13: Pronalazak opcije izgradnje videoigre

Opcija Build Settings vodi do posebnog prozora u kojem se definiraju osnovne i napredne mogućnosti izgradnje videoigre. U tom prozoru [Slika 13] može se uočiti da korisnik odabire platformu i arhitekturu za koju izgrađuje videoigru, te neke dodatne opcije, kao i mogućnost podešavanja postavki igrača. Jednom kada korisnik izvrši odabir postavki tada klikom na Build pokreće proces izgradnje videoigre.



Slika 14: Opcije izgradnje videoigre

3.3. Izrada 2D platformske videoigre

Izrada videoigre je vrlo opsežan i zahtjevan posao. Vrlo često zahtijeva od osobe da se posveti mnogočemu, od osmišljavanja priče videoigre, kreiranje likova i predmeta igre, snimanje zvukova, do pisanja programske logike, reklamiranja i prodaje videoigre. U ovom radu nisu vlastoručno kreirani likovi, predmeti i zvukovi videoigre, ali je uloženo određeno vrijeme za pronalazak odgovarajućih resursa, koji će zajedno tvoriti jednu zaokruženu cjelinu. Sljedeći navedene korake i principe izrade videoigre, daljnji tekst je podijeljen u poglavlja kojim redom se odvijala izrada videoigre.

3.3.1. Ideja videoigre

Početna točka bilo čega novoga, pa tako i nove videoigre jest ideja. Sve kreće s nekom inicijalnom idejom, s nekom zamisli. Ideja je mnogo, ali bitno je odabrati onu pravu ideju, izraditi dobar plan te se posvetiti provedbi odabrane ideje u djelo.

Isto tako je bilo u ovom slučaju. Početna ideja je bila predstaviti borbu između tame i svjetla, istaknuti kontrast koji se javlja između ta dva elementa. Tama, koja je sama po sebi mračna i puna tajni, ali i strahova trebala bi videoigri dodati osjećaj mističnosti i pobuditi u igraču osjećaj znatiželje. Nasuprot tami nalazi se svjetlost. Svjetlost je nešto poznato, nešto sigurno. Svjetlost osvjetljava put u tami, i odjednom tama više nije toliko tajanstvena i mistična. Dakle, svjetlost je ključan i vrijedan resurs videoigre. Upravo iz ovog razmišljanja je nastala ideja da igrač zrači svjetlo oko sebe, pritom osvjetljavajući put kuda prolazi. Međutim, kako je svjetlost vrijedan resurs, potrošna je. Tu je nastala ideja iza osnovne mehanike igranja videoigre. Igrač zrači svjetlo i osvjetljava put, međutim to svjetlo se može potrošiti i kada igrač ostane bez svjetla, umire. Iz tog razloga, igrač se mora kretati kroz mračna područja razine ne bi li našao put do izlaza, a uz to mora voditi brigu da pronađe izvor svjetlosti kako bi nazad prikupio izgubljenu svjetlost. Samim time životni bodovi igrača se očituju u količini svjetlosti kojom igrač zrači.

Jednom kada je nastala osnovna ideja i osnovna mehanika videoigre, bilo je potrebno osmisliti određene prepreke i smjer u kojem će se videoigra dalje razvijati. Glavna prepreka je kroz cijelu igru ostala ista, a to je tama, na kojoj su se dalje gradili i povezivali novi koncepti i novi elementi same videoigre.

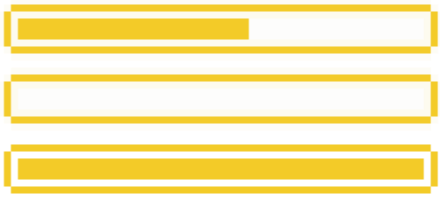
3.3.2. Izbor vizualnih elemenata videoigre

Vizualni elementi uključuju sve što igrač vidi na ekranu videoigre. Prvo što igrač primjećuje kod bilo koje videoigre jesu vizuali, odnosno slike. Ovisno o izgledu samih slika i grafike videoigre igrač kreira prve dojmove o videoigri. To mogu biti pozitivni dojmovi, ali i negativni dojmovi, kao na primjer osjećaj odbojnosti. Iz tog razloga, bitno je uložiti dovoljno vremena za pronalazak (ili kreiranje) odgovarajućih vizualnih resursa igre, kako bi takva igra privukla što širu skupinu potencijalnih igrača.

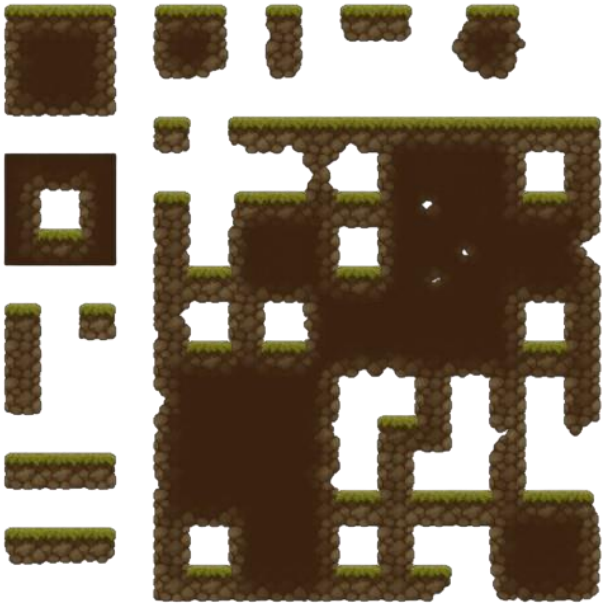
Vizualni elementi korišteni u videoigri su primarno preuzeti s Unity Asset Store-a [8], dok su također korištene i platforme poput GameDevMarket [10] i Itch.io [9], koji su opisani na početku rada. U vizualne elemente također spadaju i fontovi. U nastavku će biti prikazani neki od prepoznatljivih vizualnih elemenata koji su korišteni za izradu videoigre.

Slika 15: Prikaz slika (eng. sprite) glavnog lika (izvor: [20])

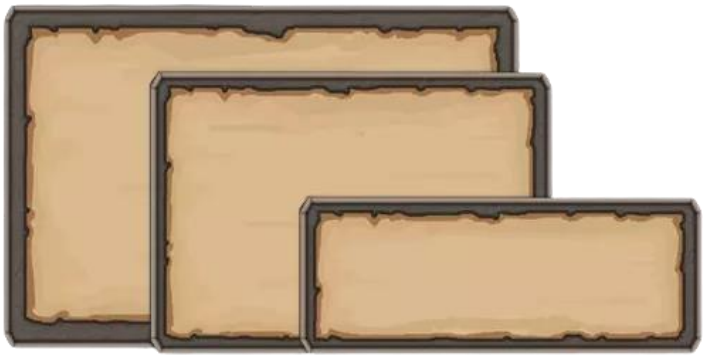
Slika 16: Prikaz objekata na sceni (izvor: [21])



Slika 17: Životna traka (eng. healthbar) igrača (izvor: [22])



Slika 18: Prikaz različitih vrsta tla videoigre (izvor: [21])



Slika 19: Okviri za tekst igre (izvor: [23])

LOST IN THE SHADOWS

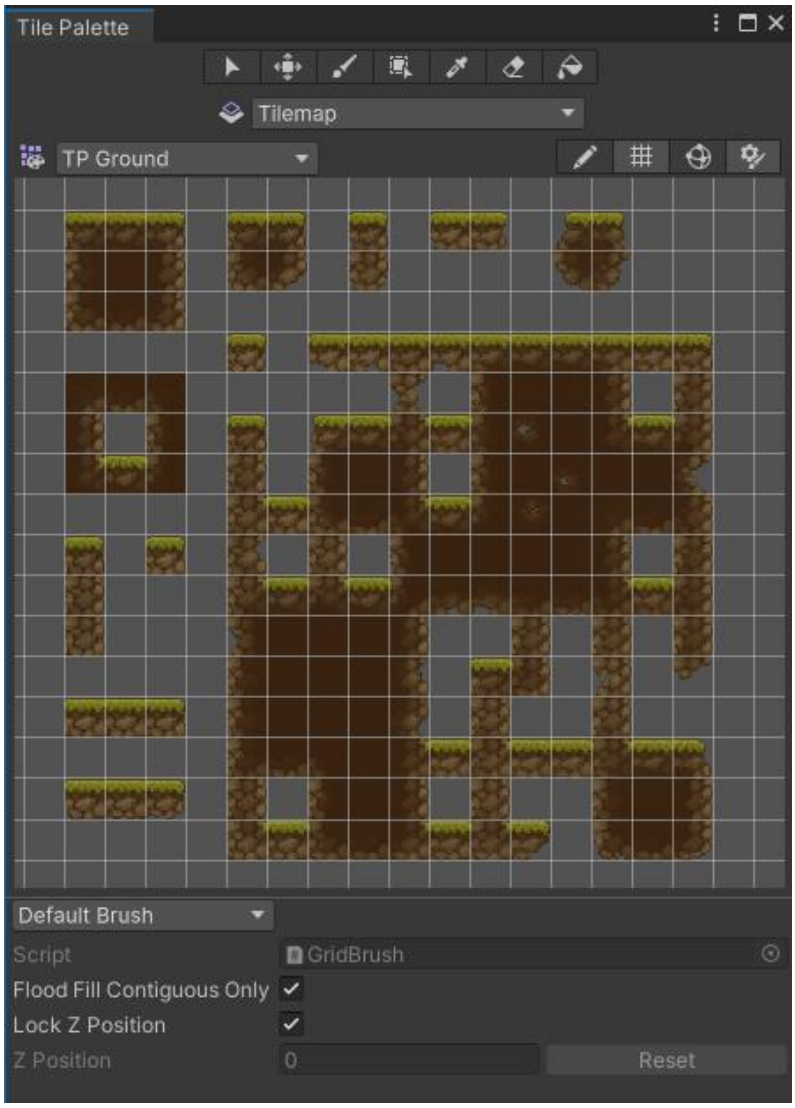
Lost in the Shadows

Slika 20: Prikaz fontova (izvor: [24] i [25])

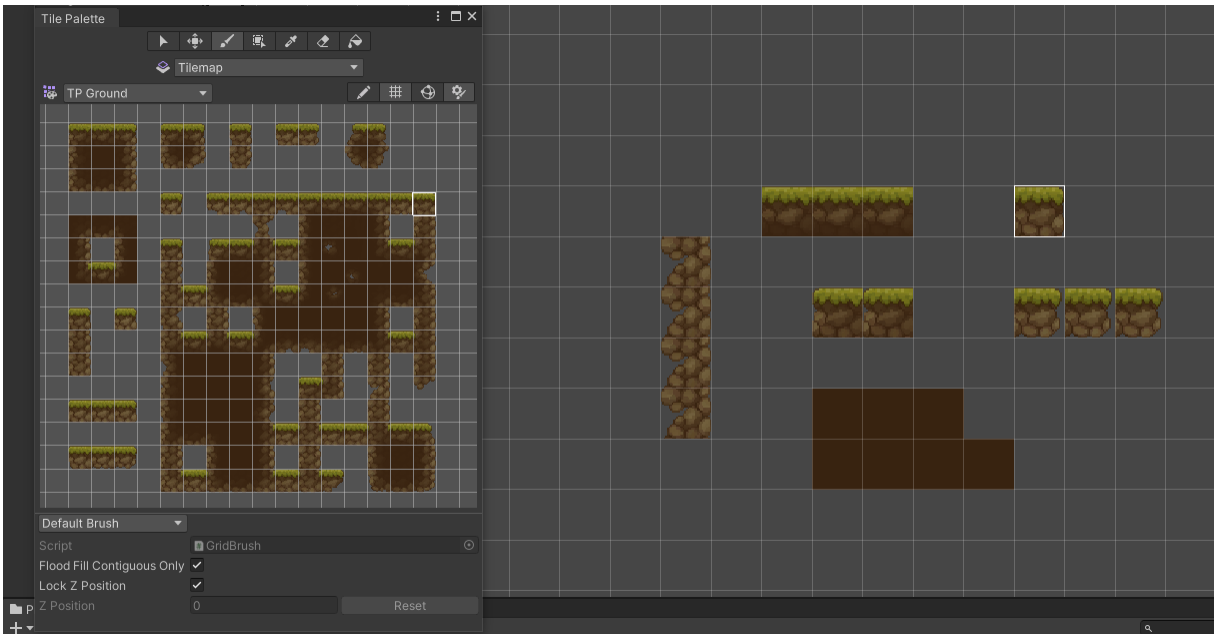
3.3.3. Postavljanje scene videoigre

Kao što je već prije rečeno, na sceni se mogu vidjeti svi objekti igre. Ovdje će biti prikazan način kreiranja tla videoigre. Za postavljanje tla na scenu korištena je opcija Grid i Tilemap. Ova ugrađena komponenta Unity-a, omogućuje brže i jednostavnije postavljanje terena na scenu, umjesto da se svaki pojedinačni objekt tla postavlja odvojeno. U paletu pločica (eng. *Tile palette*) potrebno je dodati sliku sa svim objektima tla. Kao što je prikazano na slici [Slika 20] dodana je slika TP Ground koja sadrži objekte tla. Može se uočiti da je slika automatski podijeljena na pločice, kao zasebne objekte tla.

Na vrhu prozora nalaze se akcijski gumbi koji se koriste za različite mogućnosti postavljanja, brisanja i uređivanja tla, odnosno pločica. Najjednostavniji princip korištenja ovakvog sustava je kliknuti na željenu pločicu, u ovom slučaju na objekt tla te na sceni nanijeti objekt tla prema vlastitim željama.



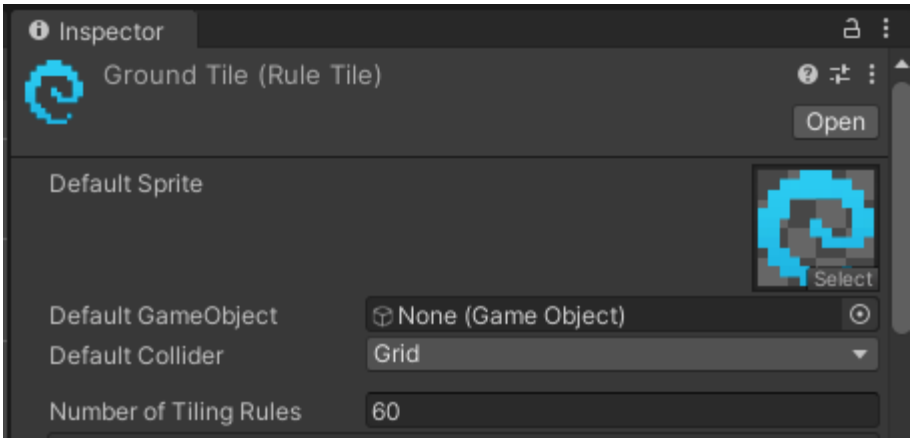
Slika 21: Prikaz palete pločica (eng. Tile Palette)



Slika 22: Postavljanje tla na scenu

Ovaj proces ubrzava i pojednostavljuje izradu podloge za daljnji razvoj igre, međutim postoji jedan problem. Potrebno je svaki objekt tla ručno prilagoditi da vizualno odgovara cjelini. Svaki put treba odabrati objekt tla, postaviti ga na scenu, zatim odabrati drugi objekt, opet ga postaviti, pa treći, četvrti objekt i tako u nedogled. Iz tog razloga, zbog beskonačnih iteracija između palete pločica i scene kreirano je bolje, optimalnije rješenje.

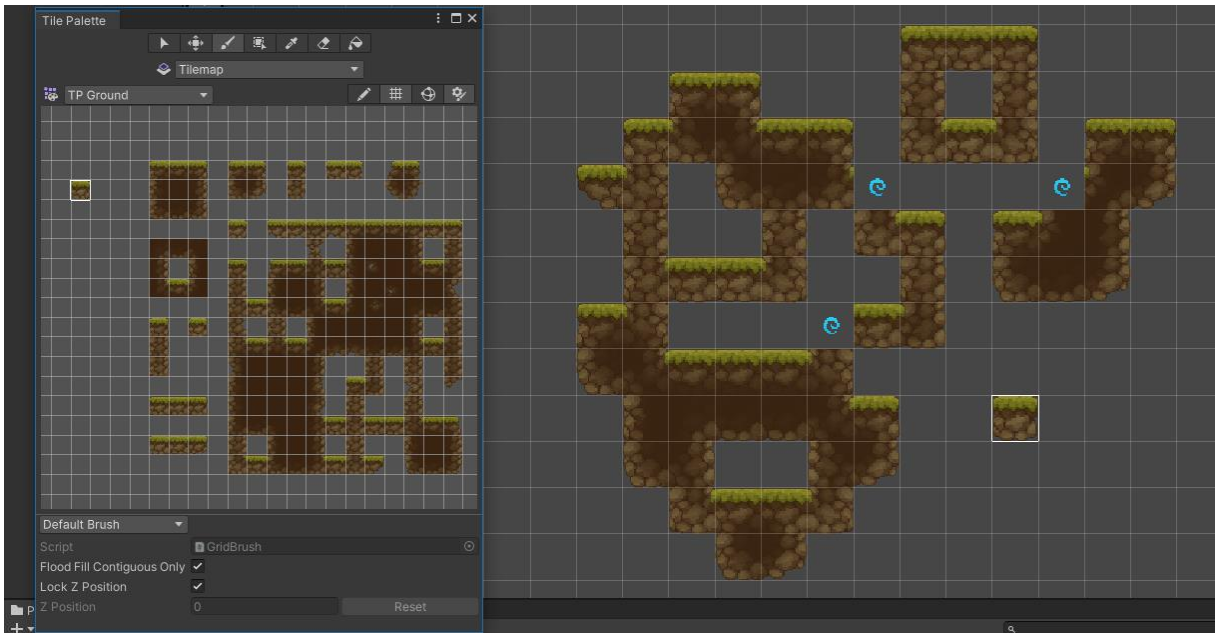
Moguće je kreirati objekt pločice koja sadrži pravila. Postavljanjem takve pločice na scenu, pločica sama određuje koja slika objekta bi trebala ići na koje mjesto. Ova metoda uvelike skraćuje vrijeme postavljanja scene i omogućuje brzo i jednostavno mijenjanje i prilagođavanje terena.



Slika 23: Prikaz pločice s pravilima

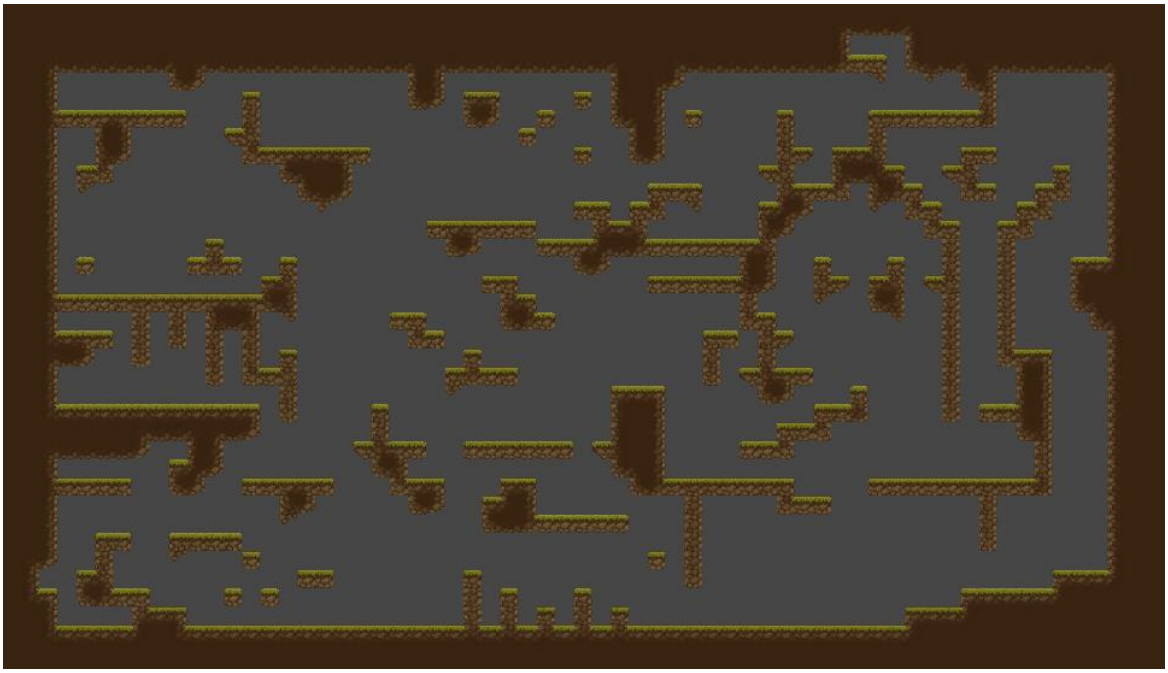
Za ovu pločicu definirano je ukupno 60 različitih pravila, odnosno 60 situacija kada se koristi koja slika tla. U desnom dijelu prozora se nalazi slika tla koja se postavlja, a uvjet je vizualno prikazan odmah s lijeve strane te slike. Tako prema slici ispod [Slika 23], se može uočiti da će se slika crnog tla postaviti samo u slučaju kada je taj objekt okružen sa svih strana s drugim objektima. Vizualno je to predstavljeno korištenjem zelenih strelica koje govore u kojem se smjeru objekt nalazi, a crveni X označava gdje objekta nema. Postoji još mogućnost pustiti polje prazno, u tom slučaju, na tom mjestu može i ne mora biti drugi objekt. U slučaju da za neku situaciju nije definirano niti jedno pravilo tada se postavlja zadana (eng. *default*) slika pločice.

Slika 24: Prikaz definiranih pravila



Slika 25: Postavljanje pločica na scenu

Na ovaj način, sve što korisnik mora učiniti je odabrati pločicu s definiranim pravilima i postaviti je na scenu. Ne mora se brinuti o tome koja slika pločice bi najbolje odgovarala, jer pločica sama određuje koja slika tla bi trebala ići na koje mjesto. Ovime je jako optimiziran i pojednostavljen proces izrade i kreiranje scene. Kreiranjem ovakvog automatskog sustava postavljanja slika pločica na scenu, korisniku je lakše i zabavnije kreirati i prilagođavati izgled scene prema vlastitim željama.



Slika 26: Konačni izgled terena scene

3.3.4. Kreiranje osnovnih mehanika videoigre

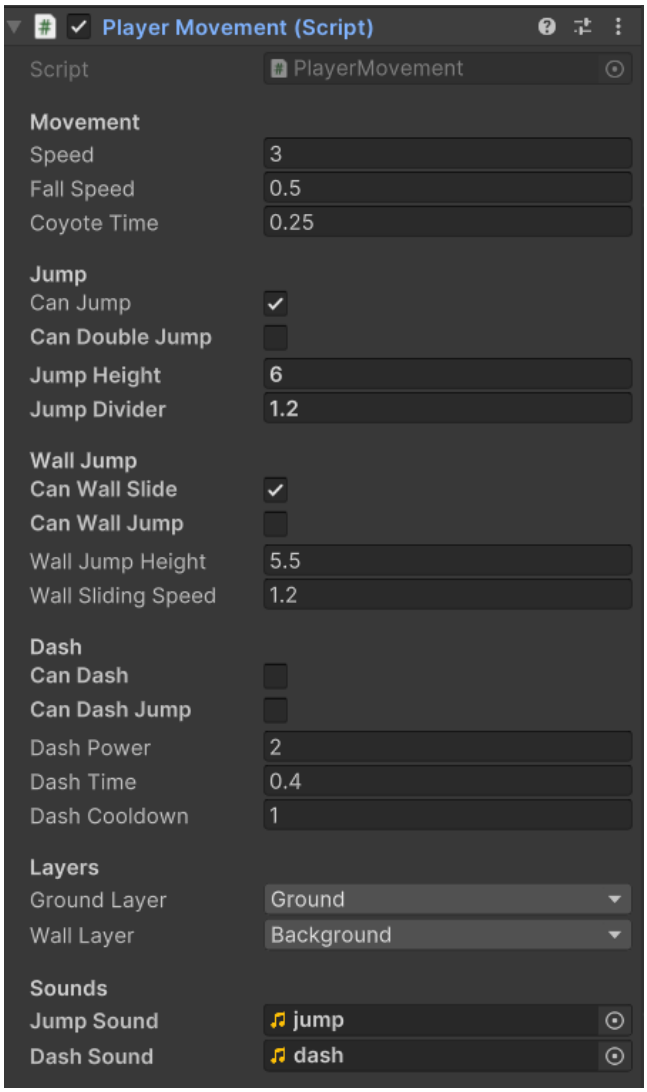
Osnovne mehanike videoigre su ključne za pravilno izvođenje same videoigre. Zbog tih mehanika igra dobiva smisao. Mehanike koje će ovdje biti predstavljane uključuju mehanike kretanja igrača, mehanika promjene životnih bodova igrača, kretanje kamere, izmjena scena i slično. Za izradu svih tih mehanika potrebno je koristiti komponente skripti koje se dodaju na željeni objekt. Skripte se pišu u programskom jeziku C#, u Visual Studio razvojnoj okolini. Sve skripte, imena i nazivi varijabli su pisane na engleskom jeziku, kako bi programski kod bio dostupan i razumljiv široj populaciji. U nastavku neće biti prikazane cijele skripte programskog koda, već samo njeni najvažniji dijelovi.

3.3.4.1. Kretanje igrača

Skripta `PlayerMovement` zadužena je za upravljanje akcijama kretanja igrača. Na vrhu skripte deklariraju se varijable koje će se dalje u programu pozivati i koristiti. Iz priloženog koda može se uočiti da postoje dvije vrste varijabli, privatne (eng. *private*) i javne (eng. *public*). Privatne varijable su vidljive i dostupne samo na razini trenutne skripte, dok su javne varijable dostupne drugim skriptama, te im druge skripte mogu mijenjati vrijednosti. Opcija `[SerializeField]` se koristi da bi se u inspektoru određene varijable pojavile i na taj način da im se jednostavnijim i bržim putem mogu promijeniti i testirati nove vrijednosti.

```
[Header("Movement")]
[SerializeField] private float speed;
[SerializeField] private float fallSpeed;
[SerializeField] private float coyoteTime;
private float playerMovement;
private Vector2 vecGravity;
private float coyoteCounter;

[Header("Jump")]
[SerializeField] private bool canJump;
[SerializeField] public bool canDoubleJump;
[SerializeField] private float jumpHeight;
[SerializeField] private float jumpDivider;
private float timer;
private bool doubleJump;
```

Na slijedećoj slici [Slika 26] je prikazan popis svih varijabli koje se mogu mijenjati izravno iz Unity Editora.



Slika 27: Prikaz parametara kretanja igrača

Funkcija `Awake()` se koristi za inicijalizaciju varijabli ili stanja prije pokretanja aplikacije. Ova funkcija se samo jednom poziva, te je vrlo slična funkciji `Start()`, gdje se `Awake()` poziva kada se objekt skripte inicijalizira, bez obzira je li skripta omogućena ili nije.

```
private void Awake()
{
    body = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    boxCollider = GetComponent<BoxCollider2D>();
    trail = GetComponent<TrailRenderer>();

    vecGravity = new Vector2(0, -Physics2D.gravity.y);
    playerGravity = body.gravityScale;
}
```

Funkcija `Update()` zadužena je za konstantno izvođenje programa. Ona se beskonačno vrti u petlji, od početka pokretanja skripte. Ova funkcija je prikladna za dohvaćanje ulaza od strane igrača i izvođenje određenih radnji na temelju dobivenih ulaza. Prikazana funkcija zadužena je za izračun brzine i smjera kretanja igrača, okretanje slike igrača da gleda u dobrom smjeru, određivanje brzine padanja igrača, postavljanje parametara za aktivaciju određene animacije te postavljanje uvjeta za mehanike kretanja.

```
    private void Update()
    {
        // player movement
        if (!isDashing)
            playerMovement = Input.GetAxisRaw("Horizontal") * speed *
Time.deltaTime;

        flipPlayer();
        transform.Translate(playerMovement, 0f, 0f);

        // falling speed
        if (body.velocity.y < 0)
            body.velocity -= vecGravity * fallSpeed * Time.deltaTime;

        // animation parameters
        anim.SetBool("run", playerMovement != 0);
        anim.SetBool("grounded", isGrounded());
        anim.SetFloat("yVelocity", body.velocity.y);

        // jumping mechanics
        if (canJump)
        {
            // to prevent from early resseting counters
            timer += Time.deltaTime;

            if (timer > 0.05f)
            {
                // reset jump counter and coyote counter
                if (isGrounded() && (!onWall() || canWallJump))
                {
                    coyoteCounter = coyoteTime;
                    if (canDoubleJump) doubleJump = true;
                }

                timer = 0;
            }
            coyoteCounter -= Time.deltaTime;

            if (Input.GetKeyDown(KeyCode.Space))
            {
                timer = 0;
                Jump();
            }
```

```
            // adjustable jump height
            if (Input.GetKeyUp(KeyCode.Space) && body.velocity.y > 0)
                body.velocity = new Vector2(body.velocity.x, body.velocity.y /
jumpDivider);
        }

        // wall sliding
        if (onWall() && canWallSlide)
        {
            body.velocity = new Vector2(body.velocity.x,
Mathf.Clamp(body.velocity.y, -wallSlidingSpeed, float.MaxValue));
        }

        if (canDash)
        {
            if (Input.GetKeyDown(KeyCode.K))
            {
                SoundManager.instance.PlaySound(dashSound);
                isDashing = true;
                if (!canDashJump) canJump = false;
                StartCoroutine(Dash());
            }
        }
    }
```

Ono što bi trebalo istaknuti je kako program određuje je li igrač na tlu ili na zidu. To se odvija u posebnim funkcijama koje vraćaju vrijednost 1 ili 0. Unity kreira nevidljivu lasersku zraku koja je određena vektorom i vraća vrijednost, ako je pogodila/presjekla objekt ispod ili pored igrača.

```
    private bool isGrounded()
    {
        RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center,
boxCollider.bounds.size, 0, Vector2.down, 0.1f, groundLayer);
        return raycastHit.collider != null;
    }

    private bool onWall()
    {
        RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center,
boxCollider.bounds.size, 0, new Vector2(transform.localScale.x, 0), 0.1f,
groundLayer);
        return raycastHit.collider != null;
    }
```

Slika 28: Detekcija kolizije igrača s tlom i zidom

3.3.4.2. Životni bodovi igrača

Logika iza životnih bodova igrača sastavljena je u skripti `PlayerHealth`. Deklaracija skripte je provedena na isti način kao i u prošloj skripti. U `Update()` funkciji se koristi provjera je li tijelo igrača „spava“, u slučaju da spava, program ga budi. To se događa jer svakog trenutka igrač ili dobiva životne bodove ili troši. Problem je taj što se funkcije `TakeDamage` ili `AddHealth` toliko puta pozovu u sekundi, da ih Unity nakon nekog vremena počne zanemarivati. Zato je potrebno probuditi tijelo kako bi obavljalo potrebnu radnju.

```
public void Update()
{
    if (body.IsSleeping()) body.WakeUp();
    if (isRegenerating || currentHealth <= 0) return;
    TakeDamage(healthWaste);
}
```

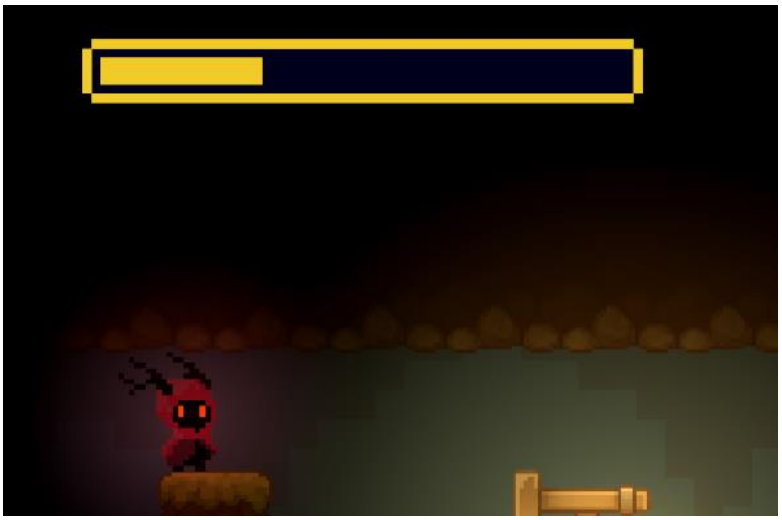
Funkcije `TakeDamage` i `AddHealth` primaju po jedan argument, vrijednost koja se igraču dodaje odnosno oduzima od životnih bodova. Također, kako se igraču smanjuju ili povećavaju životni bodovi, tako se proporcionalno smanjuje i povećava razina svjetlosti kojom igrač zrači te se kamera koja prati igrača sužava, odnosno povećava. U slučaju da životni bodovi igrača padnu ispod vrijednosti 0 tada se onemogućuje kretanja igrača te se pokreće animacija umiranja igrača i poziva funkcija `Respawn()` koja ponovno stvara igrača na mjestu `Checkpointa`.

```
public void TakeDamage(float value)
{
    if (value < 1) value *= Time.deltaTime;
    currentHealth = Mathf.Clamp(currentHealth - value, 0, startingHealth);
    spotLight.pointLightOuterRadius =
Mathf.Clamp(spotLight.pointLightOuterRadius - value * 5, 0.5f,
startingLightRange);
    mainCamera.orthographicSize = Mathf.Clamp(mainCamera.orthographicSize -
value * 1.5f, 4f, cameraSize);
```

```
        if (currentHealth <= 0 && !dead)
        {
            SoundManager.instance.PlaySound(deadSound);
            GetComponent<PlayerMovement>().enabled = false;

            anim.SetBool("grounded", true);
            anim.SetTrigger("die");
            dead = true;
        }
    }

    public void AddHealth(float value)
    {
        if (value < 1) value *= Time.deltaTime;
        currentHealth = Mathf.Clamp(currentHealth + value, 0, startingHealth);
        spotLight.pointLightOuterRadius =
Mathf.Clamp(spotLight.pointLightOuterRadius + value * 5, 0.5f,
startingLightRange);
        mainCamera.orthographicSize = Mathf.Clamp(mainCamera.orthographicSize +
value * 2, 3.5f, cameraSize);
    }

    public void Respawn()
    {
        dead = false;
        AddHealth(startingHealth);
        anim.ResetTrigger("die");
        anim.Play("Idle");

        GetComponent<PlayerMovement>().enabled = true;
    }
```

Skripta je sastavljena tako da se cijelo vrijeme igraču oduzimaju životni bodovi, a samo u trenutku kada se nalazi pored izvora svjetla, tada se igrač regenerira. U programsku kodu je ovaj dio prepoznat kao okidač (eng. *trigger*).

```
    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.tag == "HealthArea" || collision.tag == "Checkpoint")
        {
            isRegenerating = true;
            if (currentHealth == startingHealth) return;
            AddHealth(healthRegeneration);
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        isRegenerating = false;
    }
```



Slika 29: Prikaz poluprazne životne trake

3.3.4.3. Ponovno stvaranje igrača (eng. *respawn*)

Kada igrač slučajno stane na zamku (šiljke), ili ostane bez svih životnih bodova tada se pokreće skripta `PlayerRespawn`. Ona je zadužena za kreiranje male čestice svjetla na mjestu gdje je igrač umro. To je vidljivo iz sljedećeg primjera koda gdje su definirana sva svojstva novonastale čestice svjetla.

```
public void SpawnLight()
{
    GameObject light = new GameObject();
    light.name = "Light of the past";
    light.transform.position = transform.position;
    light.layer = 10;
    light.AddComponent<Light2D>();
    light.GetComponent<Light2D>().pointLightOuterRadius = 2;
    light.GetComponent<Light2D>().color = GetComponent<Light2D>().color;
    light.transform.parent = lightOfPast.transform;
}
```

Kada igrač umre, ponovno se stvara na poziciji zadnjeg `Checkpointa`. Igrač može aktivirati `Checkpoint` tako što prođe pokraj njega i tada se aktivira okidač. Jednom kada se aktivira okidač, pozicija prošlog `Checkpointa` se zanemaruje i pohranjuje se vrijednost novog `Checkpointa`.

```
public void Respawn()
{
    transform.position = currentCheckpoint.position;
    playerHealth.Respawn();
}
```

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.tag == "Checkpoint")
        {
            if(currentCheckpoint)
currentCheckpoint.GetComponent<Light2D>().enabled = false;
            currentCheckpoint = collision.transform;
            collision.GetComponent<Light2D>().enabled = true;
        }
    }
```

3.3.4.4. Kamera

Postoje tri vrste kamera u videoigri. Glavna kamera koja prikazuje cijelu scenu videoigre, zatim kamera koja prikazuje cijelu mapu videoigre odjednom, te kamera male mape koja se nalazi u desnom donjem kutu videoigre. Sve kamere koriste istu skriptu. Sve kamere prate igrača, njegovo kretanje i poziciju. Kod pomicanja kamere, dohvaća se pozicija igrača na sceni, te se koristi funkcija `Mathf.Clamp` za postavljanje granica kamere. Granice služe da kamera ne otiđe predaleko.

```
    private void Update()
    {
        transform.position = new Vector3(Mathf.Clamp(player.position.x +
lookAhead, xLimit.x, xLimit.y), Mathf.Clamp(player.position.y + cameraHeight,
yLimit.x, yLimit.y), transform.position.z);
        lookAhead = Mathf.Lerp(lookAhead, (aheadDistance * player.localScale.x),
Time.deltaTime * cameraSpeed);
    }
```

Slika 30: Prikaz male mape

3.3.4.5. Promjena scena videoigre

Promjena scena u videoigri se odvija putem skripte `SceneController`. U ovoj skripti su definirane funkcije za pokretanje određene scene videoigre. Također, kod pozivanja nekih funkcija postavlja se vrijeme na vrijednost 1 jer se može dogoditi ako igrač želi iz zaustavljene (pauzirane) igre doći do neke druge scene videoigre. U tom slučaju igra bi ostala i dalje pauzirana.

```
    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        Time.timeScale = 1;
    }

    public void MainMenu()
    {
        SceneManager.LoadScene(0);
        Time.timeScale = 1;
    }

    public void Quit()
    {
        Application.Quit(); // works only on build option

        #if UNITY_EDITOR
                UnityEditor.EditorApplication.isPlaying = false; // exits play
mode in editor
        #endif
    }

    public void Play()
    {
        SceneManager.LoadScene(2);
    }

    public void Credits()
    {
        SceneManager.LoadScene(1);
    }
```

Slika 31: Prikaz početnog ekrana videoigre

3.3.5. Dodavanje ostalih mehanika videoigre

Pod ostale mehanike spadaju sve dodatne mehanike videoigre koje poboljšavaju iskustvo igranja. To su dodatni sadržaji, zabavni elementi koji produljuju vrijeme trajanja same videoigre i nastoje učiniti iskustvo igranja što ugodnijim i kvalitetnijim.

3.3.5.1. Zvučni efekti videoigre

Za bolje iskustvo igranja svakako su korisni zvučni efekti jer oni naglašavaju određene radnje u igri. Ne samo to, već podižu igračevo iskustvo i dojmove na novu razinu igranja same videoigre. U skripti `SoundManager` kreirana je statična varijabla, što znači da na razini cijelog projekta može postojati samo jedna varijabla imena instance. Na taj način se omogućuje lakše i brže dohvaćanje varijable iz vanjskih skripti. Igra koristi pozadinsku glazbu i iz tog razloga je važno da prilikom prelaska iz scene u scenu, glazba ostane ista, to jest nastavi svirati od djela gdje je na prošloj sceni stala. Iz tog razloga se poziva funkcija da se ovaj objekt ne uništi ni kod prelaska iz scene u scenu, već ako postoji duplikat, da se obriše taj novonastali duplikat. To je korisno u slučaju kada dvije scene imaju istu komponentu `SoundManager`, pa se odbacuje duplikat, jer u suprotnom svirale bi dvije pozadinske glazbe.

```
public static SoundManager instance {  get; private set; }
private AudioSource source;
```

```
private void Awake()
{
    source = GetComponent<AudioSource>();

    // Keep this object even when we go to new scene
    if(instance == null)
    {
        instance = this;
        DontDestroyOnLoad(gameObject);
    }
    // Destroj duplicate objects
    else if (instance != null && instance != this)
        Destroy(gameObject);
}

public void PlaySound(AudioClip sound)
{
    source.PlayOneShot(sound);
}
```

3.3.5.2. Korisničko sučelje videoigre

Korisničko sučelje videoigre je definirano u skripti `UIManager`. Ova skripta se koristi za otvaranje skočnih prozora, prikaz fiksnih elemenata na ekranu igrača, kao na primjer prikaz mape, prikaz valute u igri i slično. `Update()` funkcija provjerava je li aktivna scena naziva „`TestScene`“ i ako je, tada čeka neki od unosa igrača. Na slovo M otvara se cijela mapa videoigre, te se pauzira sama videoigra. Slovo P služi za pauziranje videoigre.

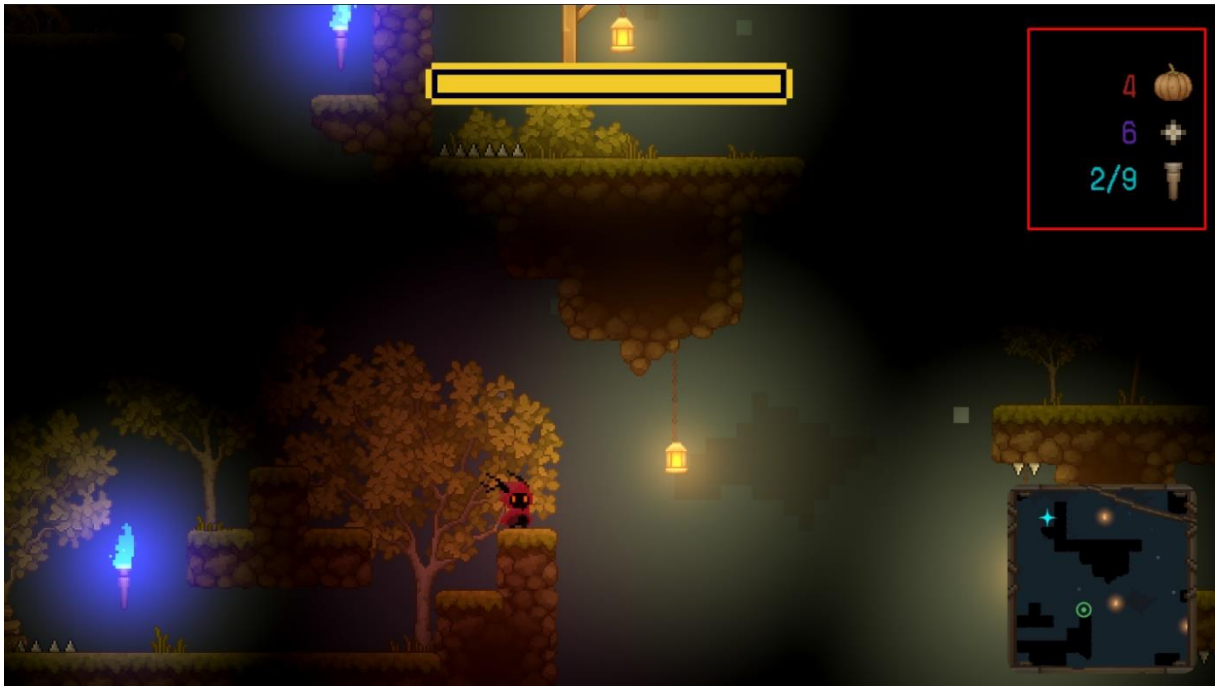
```
private void Update()
{
    if (SceneManager.GetActiveScene().name == "TestScene")
    {
        if (Input.GetKeyDown(KeyCode.M))
        {
            SoundManager.instance.PlaySound(mapSound);

            if (isMapOpen)
            {
                Time.timeScale = 1;
                map.SetActive(false);
                minimap.SetActive(true);
            }
            else
            {
                map.SetActive(true);
                minimap.SetActive(false);
                Time.timeScale = 0;
            }
```

```
            isMapOpen = !isMapOpen;
        }

        if (Input.GetKeyDown(KeyCode.P) || Input.GetKeyDown(KeyCode.Escape))
        {
            SoundManager.instance.PlaySound(mapSound);

            if (isPauseScreenOpen)
            {
                pauseScreen.SetActive(false);
                Time.timeScale = 1;
            }
            else
            {
                pauseScreen.SetActive(true);
                Time.timeScale = 0;
            }

            isPauseScreenOpen = !isPauseScreenOpen;
        }
    }
}
```

Slika 32: Prikaz pauziranog ekrana videoigre

Na kraju skripte se nalaze funkcije za izračun količine valute koju igrač ima. Ove funkcije pozivaju vanjske skripte.

```
public void addPumpkin(int value)
{
    pumpkinAmount += value;
    pumpkinText.text = pumpkinAmount.ToString();
}
public void addShard(int value)
{
    shardAmount += value;
    shardText.text = shardAmount.ToString();
}
public void removePumpkin(int value)
{
    pumpkinAmount -= value;
    pumpkinText.text = pumpkinAmount.ToString();
}
public void removeShard(int value)
{
    shardAmount -= value;
    shardText.text = shardAmount.ToString();
}
public void countBlueTorch()
{
    blueTorchAmount++;
    blueTorchText.text = blueTorchAmount.ToString();
    blueTorchText.text = string.Format("{0}/9", blueTorchAmount);
}
```



Slika 33: Prikaz valuta u desnom kutu videoigre

3.3.5.3. Posebne kutije

Kutije u igri služe za otključavanje novih mehanika kretanja igrača. Igrač, kako bi došao do krajnjeg cilja, mora pronaći kutije koje su skrivene po mapi. Svaka kutija mu otključava novu mehaniku kretanja po sceni. Kutija se aktivira kada igrač prođe pokraj nje. Neke kutije se odmah mogu otvoriti, dok neke zahtijevaju poseban predmet, koji je u ovom slučaju čekić. Ako igrač prođe pokraj kutije i po potrebi ima sa sobom poseban predmet tada se aktivira animacija otvaranja kutije i igraču se prikazuje tekst na ekranu da je otključao novu mehaniku kretanja.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player" && hammer)
    {
        SoundManager.instance.PlaySound(chestSound);
        anim.SetBool("IsOpened", true);
        player.GetComponent<PlayerMovement>().canDashJump = true;
        if (!unlocked)
        {
            text.SetActive(true);
            unlocked = true;
        }

    }
}

private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.tag == "Player" && hammer)
    {
        anim.SetBool("IsOpened", false);
        text.SetActive(false);
    }
}
```

Slika 34: Zatvorena i otvorena kutija

3.3.5.4. Dijalozi u videoigri

Na početku videoigre ili tijekom igranja igre, stalno se pojavljuju različiti dijalozi. Dijalozi u obliku poruka, uputa, ponuda ili pak govora drugih likova. Uloga dijaloga je da scenu videoigre oživi i kreira dojam dinamičnosti igre. Skripta `Dialogue` napravljena je tako da dohvaća prethodno definirane elemente u kojima je pohranjen tekst, te ispisuje slovo po slovo na ekran igrača. Ako igrač želi ubrzati proces ispisa teksta na ekran tada jednostavnim pritiskom na tipku razmaknice (eng. *Space*) omogućuje instant ispis cijelog teksta elementa.

```
private void Update()
{
    if(Input.GetKeyDown(KeyCode.Space))
    {
        if(textComponent.text == lines[index])
        {
            NextLine();
        }
        else
        {
            StopAllCoroutines();
            textComponent.text = lines[index];
        }
    }
}

private void StartDialogue()
{
    player.GetComponent<PlayerMovement>().enabled = false;
    index = 0;
    StartCoroutine(TypeLine());
}

private void NextLine()
{
    if(index < lines.Length - 1)
    {
        index++;
        textComponent.text = string.Empty;
        StartCoroutine(TypeLine());
    }
    else
    {
        gameObject.SetActive(false);
        player.GetComponent<PlayerMovement>().enabled = true;
    }
}

IEnumerator TypeLine()
{
    foreach(char c in lines[index].ToCharArray())
```

```
        {
            textComponent.text += c;
            yield return new WaitForSeconds(textSpeed);
        }
    }
```

Slika 35: Prikaz dijaloga

3.3.5.5. Trgovina u igri / ponude

Kroz igru se javljaju određene ponude, gdje igrač može zamijeniti svoje resurse, kako bi otključao neki novi predmet. Ako ima nedovoljan broj resursa, igrač ne može otključati predmet. Trgovine, odnosno ponude u igri se aktiviraju jednom kada igrač prođe pokraj mjesta ponude, koje su u igri označene s uskličnikom. Igrač pritiskom na tipku Y izvršava kupnju određene stvari, odnosno tipkom N odbija ponudu. U slučaju da igrač nema dovoljno resursa i pritisne tipku Y, tada se provjerava je li uvjet ispunjen (ima li igrač dovoljan broj resursa) te ako nije tada se događa isti proces kao da je stisnuo tipku N.

```
    private void Update()
    {
        if (shopText.activeInHierarchy)
        {
            player.GetComponent<PlayerMovement>().enabled = false;
            if (Input.GetKeyDown(KeyCode.Y) || Input.GetKeyDown(KeyCode.Z))
            {
                if(UICanvas.GetComponent<UIManager>().pumpkinAmount >=
removePumpkin)
                {
                    SoundManager.instance.PlaySound(yesChoice);
```

```
UICanvas.GetComponent<UIManager>().removePumpkin(removePumpkin);
                    UICanvas.GetComponent<UIManager>().addShard(addShards);
                }
                else
                    SoundManager.instance.PlaySound(noChoice);

                shopText.SetActive(false);
                player.GetComponent<PlayerMovement>().enabled = true;
            }

            if (Input.GetKeyDown(KeyCode.N))
            {
                SoundManager.instance.PlaySound(noChoice);
                shopText.SetActive(false);
                player.GetComponent<PlayerMovement>().enabled = true;
            }

        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
            shopText.SetActive(true);
       }
```

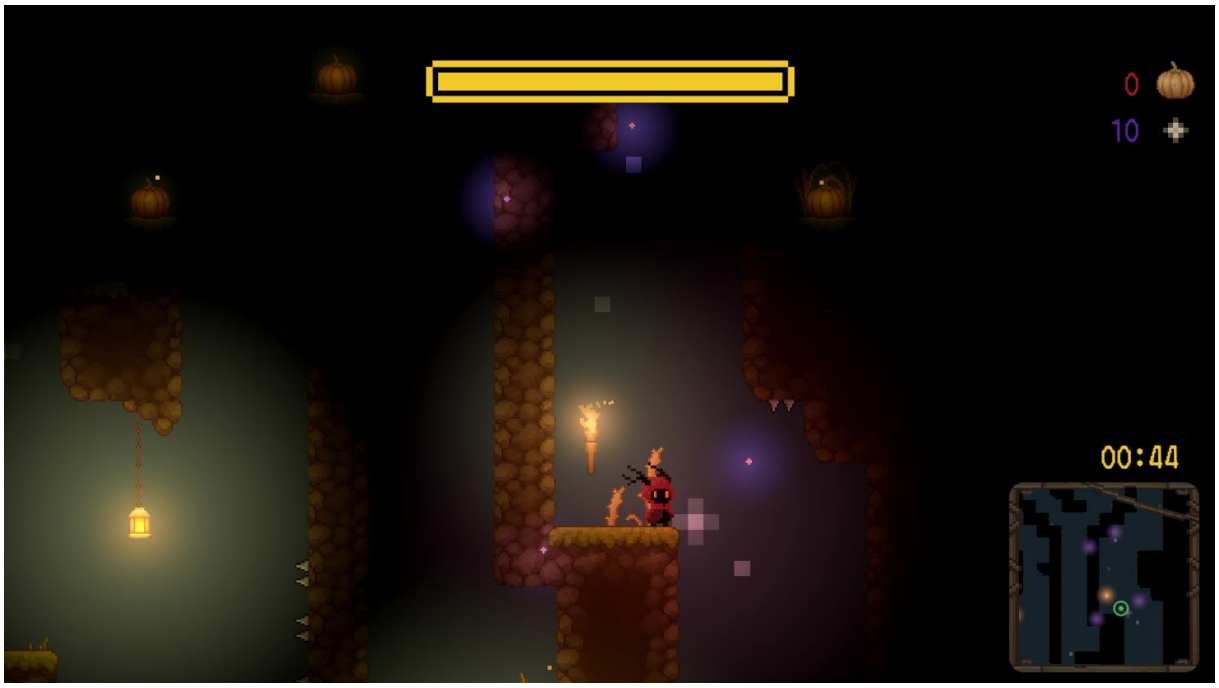
Slika 36: Primjer ponude za razmjenu resursa

3.3.5.6. Nasumično stvaranje objekata

U jednom dijelu videoigre javlja se ponuda koja, kada se prihvati, stvara nasumične objekte na sceni. Stvaraju se komadići svjetla na mapi koje igrač mora pokupiti, ti komadići svjetla su ujedno i valuta u igri. Jednom kada se kreira definiran broj tih objekata, brojač vremena započinje s odbrojavanjem. Nakon isteka vremena, svi nepokupljeni objekti se uništavaju i nestaju sa scene.

```
    public void Update()
    {
        if (activated)
        {
            for(int i = 0; i < numberOfShards; i++)
                SpawnObject();

            activated = false;
        }


        if (timeRemaining < 0)
        {
            Destroy(GameObject.Find("Shard of Light(Clone)"));
            text.SetActive(false);
            timeIsRunning = false;
        }

        if (timeIsRunning)
        {
            timeRemaining -= Time.deltaTime;
            DisplayTime(timeRemaining);
        }

    }

    private void SpawnObject()
    {
        Vector3 randomPosition = Random.insideUnitCircle * radius;

        Instantiate(itemPrefab, randomPosition, Quaternion.identity);
        timeIsRunning = true;
        text.SetActive(true);
        itemPrefab.GetComponent<ShardOfLight>().UICanvas =
GameObject.Find("UICanvas");
        timeRemaining = 75;

    }

    private void DisplayTime(float timeToDisplay)
    {
        timeToDisplay += 1;
        float minutes = Mathf.FloorToInt(timeToDisplay / 60);
        float seconds = Mathf.FloorToInt(timeToDisplay % 60);
        timeText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
    }
```

Slika 37: Nasumično stvaranje objekata na sceni – Prikaz mape



Slika 38: Nasumično stvaranje objekata na sceni – Prikaz u igri

4. Zaključak

Proces izrade videoigre zahtijeva mnoga znanja i vještine. Traži od osobe primjenu kreativnosti, dizajna i programiranja. Za izradu bilo kakve videoigre potrebni su programski alati. Kao glavni programski alat za izradu videoigre korišten je Unity. Uz dodatak Unity-u korišten je i Microsoft Visual Studio za uređivanje skripti i pisanje programskih kodova koje koriste objekti Unity sustava. Za pomoć pri pronalasku odgovarajućih vizualnih resursa korištene su digitalne trgovine resursa (eng. *Asset Store*). Internetske stranice poput FontSpace [11] i Pixabay [12] poslužile su za pronalazak fonta videoigre, kao i odgovarajućih zvučnih efekata.

Danas postoji mnoštvo različitih vrsta platformer videoigara, međutim kada se priča o prvim platformer videoigrama teško je zaobići poznate naslove poput igre Super Mario Bros [14] i Sonic the Hedgehog [15]. Oni su utemeljili osnovne principe i mehanike platformer videoigara, koje su danas široko prepoznate u svim novijim igrama tog žanra. Kao baza i motivacija za izradu igre za ovaj projekt, uzeta je videoigra Hollow Knight [17]. Ova igra objedinjuje osnovne i napredne mehanike kretanja po mapi, pritom dajući igraču osjećaj slobode kretanja i nelinearnosti igre.

Usvajanjem dobrih temelja i pozadine platformskih videoigara, prikazana je izrada vlastite 2D videoigre naslova „Lost in the Shadows“. Opis procesa izrade videoigre započinje s idejom, a nastavlja se kroz pronalazak i izbor vizualnih elemenata videoigre. Zatim je opisan proces postavljanja scene videoigre pomoću sustava pločica (eng. *Tile Palette*). Najopsežniji i najbitniji dio rada je izrada programske logike koja povezuje sve prethodno kreirane elemente. Kreiranje programskog rješenja podijeljeno je u dva dijela, kreiranje osnovnih mehanika videoigre, te dodavanje dodatnih mehanika koje znatno poboljšavaju postojeće iskustvo videoigre.

Ova videoigra je nastala kao želja za primjenom stečenih znanja i vještina na fakultetu. Razvojem igre obuhvaćena su različita područja akademskog obrazovanja, od organizacije i raspodjele vremena i resursa, do primjene kreativnosti i dizajnerskih vještina. Poseban naglasak bih stavio na razvoj programskog rješenja i programske logike same videoigre, jer bez toga, igra ne bi bila povezana, a samim time ni igriva.

Popis literature

[1] Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. (bez dat.). Pristupano 20. svibnja, 2024, sa https://unity.com/

[2] The Ultimate Beginners Guide To Game Development In Unity | by HD | We've moved to freeCodeCamp.org/news | Medium. (21. ožujak 2019.). Pristupano 20. svibnja, 2024, sa https://medium.com/free-code-camp/the-ultimate-beginners-guide-to-game-development-in-unity-f9bfe972c2b5

[3] History of Unity Game Engine. (1. lipanj 2023.). Pristupano 20. svibnja, 2024, sa https://agate.id/history-of-unity-game-engine/

[4] What Makes Unity So Popular in Game Development? - Arnia Software. (14. rujan 2022.). Pristupano 20. svibnja, 2024, sa https://www.arnia.com/what-makes-unity-so-popular-in-game-development/

[5] Unreal vs Unity vs Godot - Ruah Tech Solutions. (22. studeni 2023.). Pristupano 20. svibnja, 2024, sa https://ruahtech.com.au/unreal-vs-unity-vs-godot/

[6] How to Implement and Use Game Analytics in Unity | by Jamie Camera | Medium. (23. listopad 2019.). Pristupano 20. svibnja, 2024, sa https://medium.com/@camera.j/how-to-implement-and-use-game-analytics-in-unity-548c66063513

[7] Visual Studio: IDE and Code Editor for Software Developers and Teams. (bez dat.). Pristupano 20. svibnja, 2024, sa https://visualstudio.microsoft.com/

[8] Unity Asset Store - The Best Assets for Game Making. (bez dat.). Pristupano 20. svibnja, 2024, sa https://assetstore.unity.com/

[9] Download the latest indie games - itch.io. (bez dat.). Pristupano 20. svibnja, 2024, sa https://itch.io/

[10] Game Assets for Indie Developers | GameDev Market. (bez dat.). Pristupano 20. svibnja, 2024, sa https://www.gamedevmarket.net/

[11] Free Fonts | 120.000+ Font Downloads | FontSpace. (bez dat.). Pristupano 20. svibnja, 2024, sa https://www.fontspace.com/

[12] 4.3 million+ Stunning Free Images to Use Anywhere - Pixabay - Pixabay. (bez dat.). Pristupano 20. svibnja, 2024, sa https://pixabay.com/

[13] M. Konecki, „Žanrovi računalnih igara", nastavni materijali na predmetu Razvoj računalnih igara [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2023.

[14] The official home of Super MarioTM – History. (bez dat.). Pristupano 28. lipnja, 2024, sa https://mario.nintendo.com/history/

[15] D.S. Cohen, History of Sonic the Hedgehog by Sega Genesis. (22. ožujak 2019.). Pristupano 28. lipnja, 2024, sa https://www.lifewire.com/history-of-sonic-the-hedgehog-729671

[16] Broforce | Broforce Wiki | Fandom. (bez dat.). Pristupano 28. lipnja, 2024, sa https://broforce.fandom.com/wiki/Broforce

[17] Hollow Knight. (bez dat.). Pristupano 28. lipnja, 2024, sa https://www.hollowknight.com/

[18] Unity - Manual: Render pipelines. (bez dat.). Pristupano 28. lipnja, 2024, sa https://docs.unity3d.com/Manual/render-pipelines.html

[19] Understanding Unity and how to create a new Unity project | by Jessica Granzow | Medium. (20. lipanj, 2023). Pristupano 28. lipnja, 2024, sa https://medium.com/@jessica.r.mathe/understanding-unity-and-how-to-create-a-new-unity-project-e82be8b86965

[20] Free character - Satyr by Luck. (bez dat.). Pristupano 29. lipnja, 2024, sa https://lucky-loops.itch.io/character-satyr

[21] Pixel Art Platformer - Village Props | 2D Environments | Unity Asset Store. (11. svibanj 2023.). Pristupano 29. lipnja, 2024, sa https://assetstore.unity.com/packages/2d/environments/pixel-art-platformer-village-props-166114

[22] Pixel Neon UI Asset pack by Asrigu. (bez dat.). Pristupano 29. lipnja, 2024, sa https://asrigu.itch.io/neon-ui

[23] Fantasy Wooden GUI : Free | 2D GUI | Unity Asset Store. (18. veljača 2019.). Pristupano 29. lipnja, 2024, sa https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811

[24] Decay Font | LJ Design Studios | FontSpace. (26. srpanj, 2019.). Pristupano 29. lipnja, 2024, sa https://www.fontspace.com/decay-font-f23727

[25] Fs jenson 1 Font | opipik | FontSpace. (16. veljača, 2015.). Pristupano 29. lipnja, 2024, sa https://www.fontspace.com/fs-jenson-1-font-f21617

[26] Nintendo Classic Mini: Nintendo Entertainment System | Misc. | Nintendo. (bez dat.). Pristupano 30. lipnja, 2024, sa https://www.nintendo.com/en-gb/Misc-/Nintendo-Classic-Mini-Nintendo-Entertainment-System/Nintendo-Classic-Mini-Nintendo-Entertainment-System-1124287.html

[27] Sega Consoles | Gaming console | Record Head. (bez dat.). Pristupano 30. lipnja, 2024, sa https://recordhead.biz/history-of-sega-consoles/

[28] Save 80% on Broforce on Steam. (bez dat.). Pristupano 30. lipnja, 2024, sa https://store.steampowered.com/app/274190/Broforce/

[29] Save 50% on Hollow Knight on Steam. (bez dat.). Pristupano 30. lipnja, 2024, sa https://store.steampowered.com/app/367520/Hollow_Knight/

Popis slika