Izrada platformske igre u Unity okruženju

Mihin, Jurica

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
STRUČNI STUDIJ RAČUNARSTVO

JURICA MIHIN

IZRADA PLATFORMSKE IGRE U UNITY OKRUŽENJU

ZAVRŠNI RAD

ČAKOVEC, 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
STRUČNI STUDIJ RAČUNARSTVO

JURICA MIHIN

IZRADA PLATFORMSKE IGRE U UNITY OKRUŽENJU

DEVELOPMENT OF A PLATFORM GAME IN UNITY ENVIRONMENT

ZAVRŠNI RAD

MENTOR:
Nenad Breslauer, viši predavač

ČAKOVEC, 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
ODBOR ZA ZAVRŠNI RAD

Čakovec, 5. siječnja 2021.

Polje: **2.09 Računarstvo**

ZAVRŠNI ZADATAK br. 2020-RAČ-R-15

Pristupnik: **Jurica Mihin (0313022118)**
Studij: redovni preddiplomski stručni studij Računarstvo
Smjer: Inženjerstvo računalnih sustava i mreža

Zadatak: **Izrada platformske igre u Unity okruženju**

Opis zadatka:

Igra se sastoji od razina i konceptualno različitih prepreka koje igrač mora savladati kako bi uspješno napredovao do sljedeće razine.
Neke od funkcionalnosti su: kretnja igrača, dizajn razina, posebne sposobnosti kao što su dupli skok, jaki skok, hodanje po zidu, skok od zida i drugo. Potrebno je izraditi scene te sve potrebne elemente, koristiti mogućnosti koje pruža podsustav za osvjetljenje scene, animacije te koristiti simulaciju fizikalnih svojstava. Izraditi početnu scenu s izbornikom. Koristiti platformu Unity, programski jezik C# te dodatne programske alate.
Završni rad mora sadržavati sažetak, sadržaj i uvod, nakon čega slijedi poglavlje u kojem je potrebno navesti i pojasniti osnovne ciljeve rada te očekivani rezultat. U narednom poglavlju potrebno je opisati primijenjene postupke, alate i metode. Poglavlje koje slijedi obrađivati će postignute rezultate nakon čega slijedi poglavlja u kojem se kritički raspravlja o primijenjenim metodama i postupcima te se u narednom poglavlju iznose glavni zaključci rada. Rad se završava poglavljima s popisom literature te prilozima.

Zadatak uručen pristupniku: 5. siječnja 2021.
Rok za predaju rada: 20. rujna 2021.

Mentor:

Breslauer

Nenad Breslauer, v. pred.

Predsjednik povjerenstva za završni ispit:

ZAHVALA

Zahvaljujem svom mentoru Nenadu Breslaueru na strpljenju i pomoći u svakom koraku izrade završnog rada. Također zahvaljujem svim profesorima Veleučilišta na stečenom znanju koje je potrebno za početak moje nove karijere.

Jurica Mihin

SAŽETAK

Napredak tehnologije sve više raste iz dana u dan što rezultira sve većim brojem zaposlenika u IT sektoru i sve većom konkurencijom na tržištu. Na tržište dolazi sve više uređaja poput računala, mobitela, tableta i ostalih pomagala bez kojih je život današnjice teško zamisliv. Posao u IT-u nudi vrlo zadovoljavajuće plače i veliku mogućnost napredovanja.

Ovaj završni rad opisuje cjelokupan proces izrade 2D video-igre. Kod izrade koristi se razvojno okruženje Unity. Unity je jedno od najpoznatijih okruženja za izradu video-igara, a njegova najveća vrlina je jednostavnost i pregled projekta. Za funkcionalnost igre dodane su skripte u kojima se nalaze programski kodovi pisani u C# programskom jeziku. Za pisanje kodova upotrijebljeno je razvojno okruženje Visual Studio. Postoje i druga okruženja za izradu video-igara kao što su Unreal Engine i CryEngine, no Unity je i dalje najpoznatiji.

Sam proces izrade video-igara zahtijeva poprilično opsežno znanje koje obuhvaća ne samo programiranje već i dizajn i ostale segmente. Rad prikazuje izradu jednostavne video-igre u kojoj su korištene sve potrebne vještine. Veliku pomoć pruža Unityeva trgovina već unaprijed izrađenih elemenata, Unity Asset Store. Preuzimanjem besplatnih ili plaćenih elemenata sam proces izrade kreće se puno brže. Opisan je postupak kreiranja pojedinih razina koje igrač mora svladati kako bi uspješno završio igru. Na svakoj razini nalaze se određene prepreke i protivnici koji otežavaju napredak, pa je zbog toga glavnom igraču dodana sposobnost napada mačem kako bi se obranio od protivnika. Također je opisana pozadinska glazba koju je moguće čuti tijekom cijele igre. Objašnjene su komponente koje služe za fizikalna ponašanja elemenata te mogućnost njihova podešavanja. Dodane su animacije na svaki element na kojem je to potrebno kako bi se vizualno dočarao doživljaj igranja. Ako igrač izgubi sve živote, vraća se na početak trenutne razine te ponovo dobiva priliku uspješno riješiti tu razinu, a broj neuspjelih pokušaja može se cijelo vrijeme pratiti prikazom brojača smrti na ekranu.

Ključne riječi: *2D, Unity, programski kod, C#, Visual Studio, Unity Asset Store*

SADRŽAJ

1. UVOD

Završni rad opisuje cjelokupan proces razvoja 2D platformer video-igre koristeći Unity Engine razvojno okruženje. Potrebno je razviti video-igru u kojoj igrač prelazi razine od najlakše prema najtežoj upravljajući glavnim likom i izbjegavajući razne prepreke i neprijatelje. Dodani su svi potrebni elementi za pravilnu funkcionalnost igre. Igra je smještena u dvodimenzionalnom svijetu. Najveća inspiracija za ovaj projekt bila je video-igra Super Mario Bros.

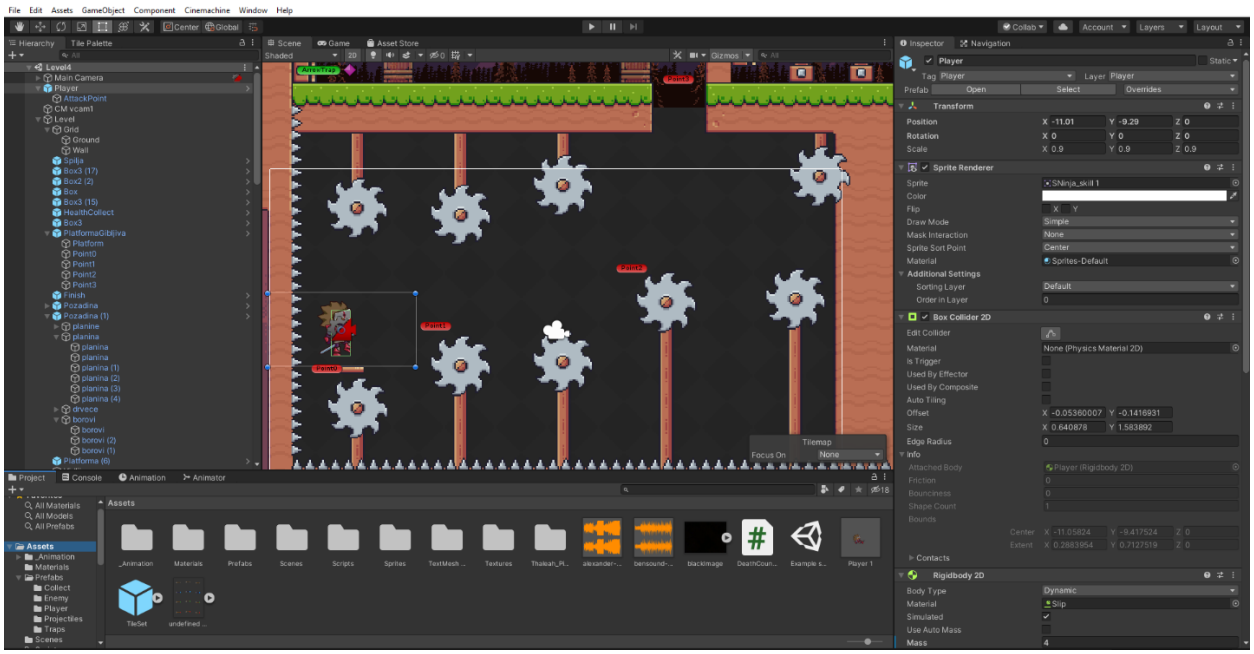
2. ALATI

U ovom dijelu su opisani alati koji su korišteni u ovom završnom radu.

2.1. Unity

Unity je najpoznatiji program koji služi za izradu video-igara za različite platforme. Po prvi puta je najavljen 2005. godine i to kao kompatibilan samo za OS X operacijske sustave. Od tada se proširio na čak 27 platforma. Vrlo je jednostavan za korištenje jer nudi mogućnost povlačenja i puštanja različitih elemenata potrebnih za izradu projekata. Najčešće se koristi programski jezik C# ali u dosta slučajeva i C++. [1].

Unity uvelike pomaže kod nekih osnovnih svojstava video-igara, na primjer ponašanje u skladu s fizikom. Potrebno je potrošiti puno vremena da bi se riješio problem fizičkih svojstava pa se zbog toga koriste već ugrađene funkcije. Unity je IDE (integrated development enviroment) što znači da nudi sav potreban alat za izradu projekta na jednom mjestu. Nudi i mogućnost pristupa datotekama projekta u samom programu.. Za pisanje koda najčešće se koristi Microsoftov Visual Studio ili Visual Studio Code. Kompatibilan je s različitim platformama poput iOS-a, PC, igraće konzole, pa čak i VR za one koji razvijaju video-igre za Oculus Rift ili HTC Vive. Jedina mana u svemu je što nema toliko dobro razvijen sustav za grafički prikaz pa grafika neće izgledati tako dobro kao što izgleda u Unreal Engineu ili Cryengineu. Instalacija je vrlo jednostavna; jednostavno se preuzme sa službene stranice te treba slijediti navedene upute. [2]



Slika 1. Unity sučelje

Izvor: autor

Dijelovi sučelja

- *Scene* prozor – Služi za vizualni prikaz trenutnog projekta kako on u stvarnosti izgleda, nebitno radi li se o 2D ili 3D projektu. U ovom prozoru postoji mogućnost slaganja elemenata i kreiranje virtualnog svijeta. Nalazi se u sredini. Može se promijeniti na *Game* ili *Asset Store* prozor klikom na određenu tablicu. *Game* prozor prikazuje što igrač vidi kada pokrene igru, dok *Asset Store* otvara trgovinu *assetima*.
- *Project* prozor – Nalazi se ispod prozora scene i prikazuje sve spremljene *assete*, slike, zvučne datoteke i ostalo što je spremljeno u projektu. Kada se doda neka nova datoteka, može se naći u ovom prozoru.
- *Hierarchy* prozor – Nalazi se lijevo od prozora scene, prikazuje hijerarhijski poredak svih objekata u trenutnoj sceni. Klikom na proširenje prikazuje sadrži li neki objekt na sebi više objekata i koji su to. U ovome prozoru koristi se tako zvani *parenting* koncept, glavni objekt je *parent* dok je svaki objekt unutar, točnije ispod njega, njegov *child objekt*.
- *Inspector* prozor – Nalazi se desno od prozora scene, služi za promatranje i uređivanje svih postavki odabranog objekta koji se odabire u *hierarchy* ili *scene*

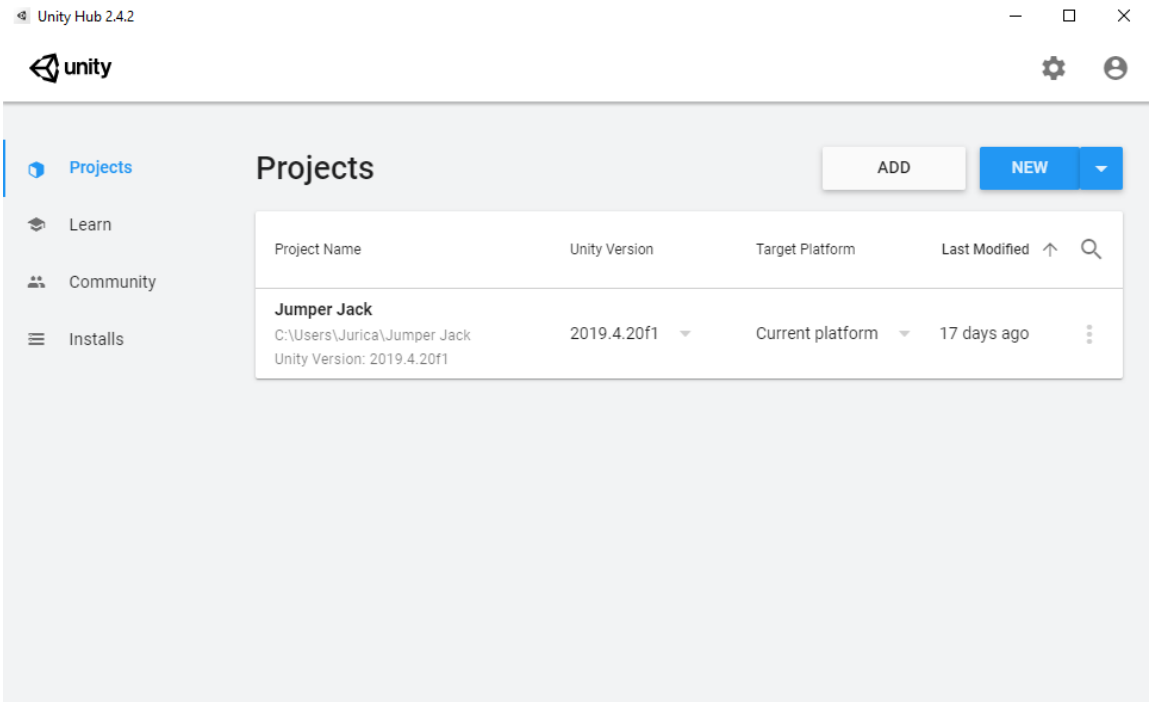
prozoru. Na primjer, kod glavnog igrača se mijenja brzina hodanja, jačina skoka i slično.

- *Toolbar* prozor – Najvažniji prozor u *Unityu*. Nalazi se iznad prozora scene, s lijeva se vide glavni alati za manipulaciju objekata u sceni gdje se nalaze *play*, *pause i step* kontrole koje služe za pokretanje, zaustavljanje i preskakanje scene. [3]

2.2. Unity Hub

Unity Hub je aplikacija koja se koristi uz Unity okruženje. Unity Hub služi za organizaciju, pretraživanje, preuzimanje i upravljanje samim projektima; svrha je uglavnom preglednost. S Unity Hubom se može:

- upravljati *Unityem* i njegovim licencama
- izrađeni projekt u *Unityu* se može povezivati s više verzija programa
- odrediti cilj projekta bez pokretanja uređivača
- upotrebljavati *template* (predložak) za neke uobičajene projekte što pridonosi bržem razvoju projekta. [4]



Slika 2. Sučelje Unity Hub-a

Izvor: autor

2.3. Unity Asset

Unity Asset je svaki element koji je korišten u Unityu. Asset može biti kreiran izvan Unitya, na primjer u programu Blender. To može biti 3D model, nekakva zvučna datoteka, slika ili bilo koja druga vrsta datoteke koje Unity podržava. Također, assete je moguće stvarati i u samome Unityu, na primjer Animator Controller, Audio Mixer i Render Texture.

2.3.1. Unity Asset Store

Unity Asset Store je trgovina assetima. Assete mogu kreirati svi, od zaposlenika Unity Technologiesa do članova Unity zajednice. U trgovini se nalazi puno vrsta uzoraka koje je moguće preuzeti. Postoji i mogućnost preuzimanja cijelih gotovih projekata te mijenjanje određenih stavki na njima. U trgovini se osim asseta koji imaju određenu cijenu nalaze i asseti koje je moguće besplatno preuzeti. Postoji mogućnost izrade i prodavanja asseta. Pristupanje asset storeu je moguće kroz standardni web preglednik ili direktno kroz Unity.

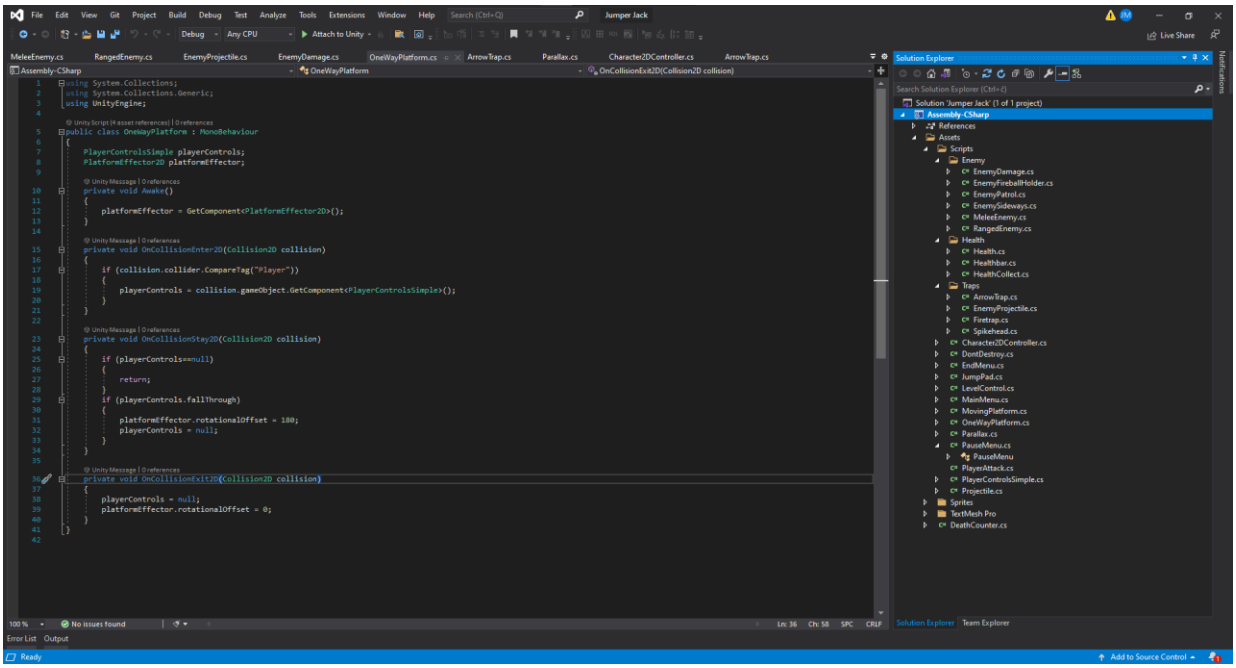
Vrste *asseta*

- 3D *asseti* - obuhvaćaju vozila, likove, rekvizite, vegetaciju i animacije. *Unityev humanoid animation retargeting* služi za miješanje likova i animacija iz različitih izvora
- 2D *asseti* – obuhvaćaju *spriteove*, *teksture*, likove, okolinu, fontove, materijale i *UI (User Interface)* elemente
- *Add-ons* (Dodaci) su napredne funkcije koje se mogu dodavati u projekt
- zvuk – zvučni efekti se mogu preuzeti s *Unity Asset Store* knjižnice zvučnih efekata ili kreirati samostalno
- *Templates* (Predlošci) – omogućavaju preuzimanje već kreiranih dijelova projekta, videa s uputama i slično. Ova vrsta *asseta* je namijenjena početnicima
- *Tools* (Alati) – obuhvaćaju širok spektar opcija, od *AI* (*artificial intelligence*) do vizualnog skriptiranja
- *VFX* – vizualni efekti, obuhvaćaju efekte čestica i sjena [5].

2.4. Visual Studio

Visual Studio je IDE (Integrated Development Enviroment) što u prijevodu znači integrirano razvojno okruženje. Napravljen je od strane Microsofta u svrhu razvijanja softverskih

rješenja kao što su programi, web stranice, mobilne aplikacije i ostalo. Napravljen je prije nešto više od 20 godina izlaskom svoje prve verzije Visual Studio 97. U ovom radu se koristi Visual Studio 2019 verzija. Podržava IntelliSense tehnologiju koja pomaže u dovršavanju programskog koda. Instalacija je vrlo jednostavna, a preuzeti se može sa službene stranice Microsofta [6]. Osnovna verzija programa je besplatna i namijenjena je studentima, open-source i invidualnim programerima. Profesionalni paket počinje od 45 američkih dolara na dalje uz probnu verziju. [7]



Slika 3. Sučelje Visual Studi-a

Izvor: autor

3. PLATFORMER

Platformer-igra je podžanr action igre kod koje sam igrač upravlja likom ili objektom u dvodimenzionalnom ili trodimenzionalnom svijetu i nailazi na razne prepreke kao što su rupe, izbočine, penjalice i slično. Svijet u kojem se igrač kreće može biti pozicioniran vodoravno ili okomito, ovisi o vrsti igre.

Platformer-igre su jedne od najstarijih video-igara, desetljećima su vodile titulu najpopularnijih video-igara upravo zbog svoje jednostavnosti i dostupnosti. Iako su one ispisale povijest video-igara, još uvijek postoje tvrtke koje se bave kreiranjem baš takve vrste igara što pokazuje koliko su zapravo kvalitetno osmišljene. Njihova popularnost je naglo pala 1990-ih

godina zbog brzog razvoja novih vrsta igara. Mnogi obožavatelji platformer-igara tvrde da je arkadna igra Space Panic koja je izašla 1980. godine bila prva platformer video-igra, dok većina ljudi smatra da je prva bila Donkey Kong (iako je izašla 1981. godine). Vodi se dilema o tome je li platformer generacija započeta izlaskom Space Panica ili jedne od najpoznatijih igara Donkey Kong. Donkey Kong je izašao samo godinu dana nakon Space Panica, oko 1981. godine. Platformer-igre su platforme s jednim zaslonom što znači da se na zaslonu vidi samo jedno statično okruženje, te ako se igrač miče na krajeve mape, okruženje se neće promijeniti. Kada je razina jednog zaslona završena, prelazi se na sljedeći zaslon ili se ostaje na istom, ali uz težu razinu levela, ovisno o igrici koja se igra. Osim platformi s jednim zaslonom postoje također i platforme kod kojih je moguće bočno i vertikalno pomicanje okruženja. [8]

4. O IGRI

Jumper Jack je igra u kojoj igrač upravlja glavnim junakom Jackom. Jack je glavni lik koji prolazi kroz razne prepreke boreći se protiv raznih neprijatelja kako bi uspješno napredovao do sljedeće razine. Prolaskom kroz razine, težina igre postaje sve zahtjevnija. Jack ima mogućnost skakanja, dvostrukog skoka, napada mačem i penjanja po zidu. Svaka razina ima svoje određene prepreke pa tako, na primjer, neprijatelje susrećemo tek na trećoj razini. Prepreke se sastoje od pila, bodljikavih oštrica, zamki koje ispaljuju strelice i zamki koje ispaljuju plamen. Svaka zamka je podešena tako da glavnom liku oduzme određenu snagu koja se ponovo napuni prijelazom na sljedeću razinu; primjerice, bodljikave oštrice momentalno ubiju glavnog igrača dok pila u nekim slučajevima oduzima samo po pola srca od moguća četiri. Neprijatelji se kreću između dviju određenih točaka, te kada im se glavni lik dovoljno približi, napadaju ga mačem i svakim zamahom uzimaju mu po jedno srce. Ako glavni lik prođe kraj neprijatelja i uspije se izvući prije nego što ga neprijatelj ozlijedi, oduzima mu se po pola srca jer ga je dotaknuo. Također, igrač može naletjeti na drugu vrstu neprijatelja koji će ga slijediti ako im se jedanput dovoljno približi. Oni su leteći te mogu putovati i kroz zidove. Postoje i neprijatelji čija uloga nije oduzimanje života već samo odguravanje glavnog lika prema zamkama ili smjeru koji je zadan. Njih nije moguće ni ubiti niti gurati, već ih treba pokušati izbjeći. Da bi se olakšao put do sljedeće razine, dodane su raznolike kutije, platforme po kojima se može hodati i skakati, platforme koje se gibaju između određenih točaka, trampolin i zidovi po kojima se glavni igrač penje. Igra ima ukupno sedam razina na kraju

čega dolazi scena u kojoj se čestita igraču na uspješno prijeđenoj igri. Na završnoj sceni je vidljiv jedan zanimljiv detalj, a to je takozvani death counter ili brojač smrti koji govori koliko puta je igrač ukupno umro tijekom cijele igre. Njega je tijekom cijelog vremena igranja moguće vidjeti u gornjem desnom kutu. Kada glavni igrač umre, brojač se ponovo stvara na početku razine i započinje ponovo.

5. GRAFIKA

2D računalna grafika je generacija digitalnih slika koje su vidljive na računalu. Sastoji se od dvodimenzionalnih modela kao što su geometrijski modeli, tekst i digitalne slike. Koristi se najčešće kod crtanja, tipografije, kartografije, oglašavanja itd. U odnosu na 3D grafiku, 2D grafika je poželjnija kod crtanja i sličnih radnji zbog toga jer ima jasniji uvid na sliku nego kod 3D grafike kod koje slike više liče na fotografiju nego na tipografijski pristup. Vrlina 2D grafike jest u tome što se neki sadržaj može prikazati u više rezolucija, tako da se može izrađivati onaj sadržaj koji odgovara različitim uređajima. Povijest 2D grafike započela je 1950-ih godina i temeljila se na vektorskim grafičnim uređajima.

Moderne grafične kartice koriste raster tehniku, dijeleći zaslon u pravokutnu mrežu piksela zbog relativno niske cijene u usporedbi s vektorskim grafičnim hardverom.

Neka od poznatih grafičnih sučelja kao što su Max OS, Microsoft Windows ili X Window System temelje se na 2D grafičnim konceptima. Korisnička sučelja unutar većine aplikacija su uobičajeno 2D uglavnom zbog toga što su ulazni uređaji (kao što je računalni miš) ograničeni na dvije dimenzije. Osim što se koristi za vektorska crtanja i ostalo, također se koristi kod video -igara kao što su kartaške ili strategijske igre kod kojih 3D prostor nije potreban. Najpoznatiji programi za crtanje Adobe Illustrator i CorelDraw koriste upravo takvu grafiku za rad s vektorima što omogućava odabir rezolucije po želji. [9]

5.1. Sprite

Sprite je 2D objekt koji sadrži teksture (slike koje vidimo na ekranu). U dvodimenzionalnom svijetu Unity će sam odrediti nekakav sprite koji je vidljiv na ekranu, dok u trodimenzionalnom svijetu sprite neće biti oblik već ploha koja izgleda kao neka vrsta papira, jer Z koordinata neće imati neku vrijednost. Kako bi izabrani sprite bio vidljiv na ekranu, game objectu

se mora dodati komponenta sprite renderer. Spritove je moguće preuzimati s Asset Storea ili samostalno izraditi pomoću dodatnih programa kao što su Adobe Photoshop, Gimp i ostali. U ovom radu su preuzeti besplatni spriteovi Ninja Sprite Sheet, Knight Sprite Sheet, Pixel Adventure 1 i drugi. Dodavanje spriteova se vrši dodavanjem željene slike u Sprites mapu u Assets mapi. Kada je slika dodana u mapu, objektu kojem se želi dodati sprite, dodaje se nova komponenta sprite renderer. [10] Dodavanje spriteova se može vrlo lako izvršiti jednostavnim povlačenjem iz mape Sprites u polje Sprite u sprite rendereru. U ovom završnom radu spriteovi su upotrijebljeni kod igrača, neprijatelja, terena i svega što igrač, kada igra, može vidjeti.

Jedan od problema kod glavnog spritea jest to da kada se sprite kreće po horizontalnom pravcu, okrenut je uvijek u defaultnom smjeru kako je bio dodan u sprite renderer. Da bi se sprite okretao u smjeru kretanja, mora se dodati dio programskog koda koji to omogućuje.

```
void Flip()
    {
           //okretnanje igraca u smjeru kretanja
           if (movement < 0 && facingRight)
           {
                 flip();
           }
           else if (movement > 0 && !facingRight)
           {
                 flip();
           }
     }
     void flip() //okret igraca
    {
        facingRight = !facingRight;
        transform.Rotate(0f, 180f, 0f);
    }
```

Kod 1. Zakret igrača

Izvor: autor

6. KOMPONENTE

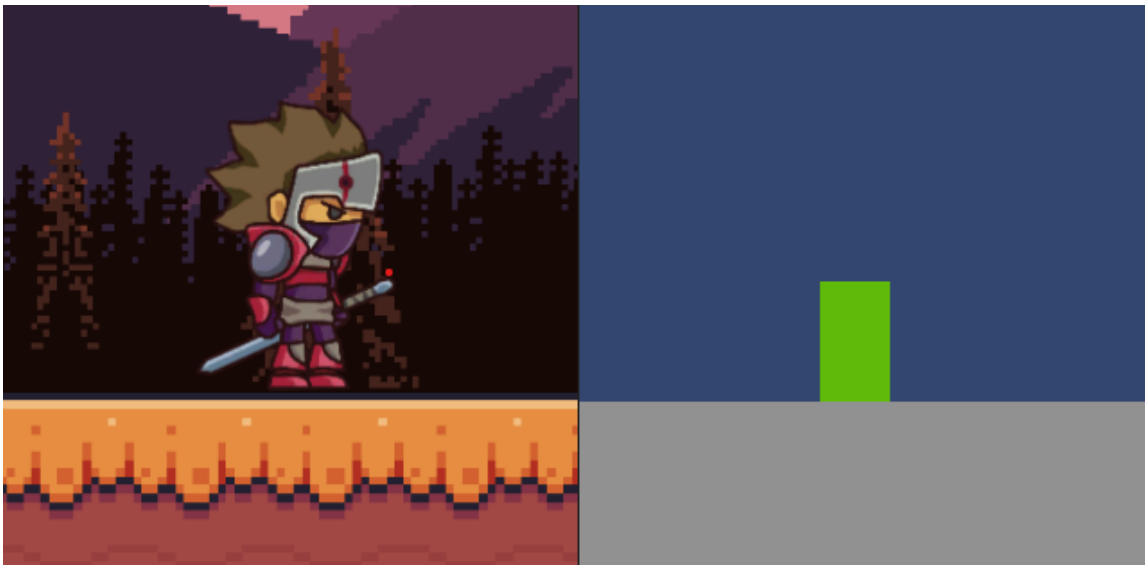
U ovom dijelu su opisane komponente koje su ključne za sva fizikalna i slična svojstva elemenata.

6.1. Transform

Svaki objekt na sebi obavezno ima transform komponentu. Transform služi za određivanje objektove pozicije na X, Y i Z osima, za rotiranje objekta po X, Y i Z osima, te za prilagodbu veličine objekta po X, Y i Z osima. Glavni lik na svakoj sceni ima drugačiju poziciju zbog različitih veličina scena, početne pozicije i slično. Rotacija kod glavnog lika je po svim osima jednaka nuli jer nema potrebe za zakretanjem lika. Što se tiče scale vrijednosti, odnosno veličine, određeno je da je veličina glavnog lika 0.9 prema čemu su i dizajnirane razine tako da se glavni lik može provlačiti kroz prepreke i ostalo.

6.2. Sprite Renderer

Sprite renderer komponenta služi za prikazivanje spriteova kao što je već objašnjeno ranije u tekstu.



Slika 4. Glavni lik prije i nakon dodavanja spriteova

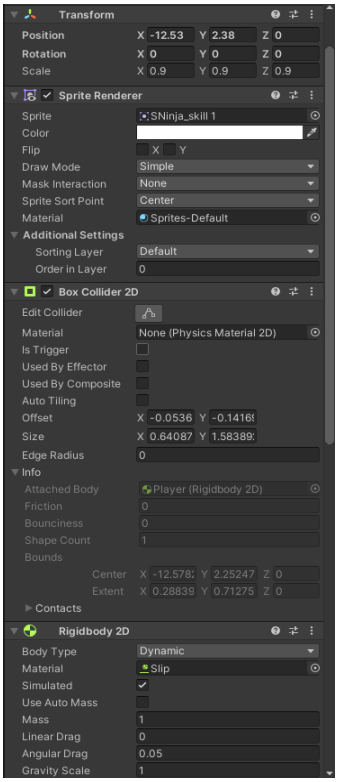
Izvor: autor

6.3. Box Collider

Collideri (engl. sudarači) se koriste kako bi se objektima dodalo fizičko svojstvo krutosti. Dok svojstvo rigidbody omogućava objektima da se ponašaju u skladu s fizikom, collideri omogućuju da se objekti sudare ili dotaknu. Svaki objekt može imati na sebi collider neovisno o rigidbodyu i obrnuto. Vrlo korisno svojstvo jest da je prilikom sudara dvaju ili više collidera moguće dobiti obavijest o sudaranju te se s time mogu programirati razne radnje poput uništavanja neprijatelja i slično. Box colliderovu veličinu je moguće mijenjati klikom na ikonu za uređivanje collidera te prilagoditi željenu veličinu.

6.4. Rigidbody

Rigidbody se koristi u svrhu korištenja fizikalnih svojstava kao što su gravitacija, masa, otpor i slično. Vrijednostima se može jednostavno upravljati u inspector prozoru te isprobavati igru sve do željene vrijednosti. U polje material je dodan slip materijal kako se glavni lik ne bi zaglavio sa zidom ili slično, već bi klizio uza nj.



Slika 5. Komponente glavnog lika

Izvor: autor

7. RAZVOJ IGRE

U ovom dijelu se postupno opisuje proces izrade video-igre. Počinje se s dizajnom i vizualnim prikazom cijele igre dodavanjem pozadine, trampolina, platformi i pločica. Sljedeće se objašnjava kamera koja prati igrača, a nakon toga i sama mehanika glavnog igrača.

7.1. Pozadina

Pozadina je vizualni dodatak kojim se dodatno dočarava razina igre. Moguće ju je samostalno kreirati pomoću raznih programa poput Adobe Photoshopa ili preuzeti već kreiranu pozadinu po dijelovima s asset storea kako je napravljeno u ovom završnom radu. Dodavanjem pozadine kreirana je samo pozadinska slika koja izgleda poprilično dosadno, pa je u ovom radu dodan parallax efekt.

Parallax efekt je posebna tehnika koja služi za pomicanje određenih dijelova pozadine. Svrha ove tehnike je stvoriti dinamičniji vizualni doživljaj video-igre. U ovome završnome radu se pozadina sastoji od dviju vrsti planina, drveća i borova. Neki elementi pozadine su duplicirani zbog ljepšeg vizualnog prikaza, npr. više borova, planina itd. Svaka grupa elemenata koji čine pozadinu ima svoju određenu vrijednost parallax efekta što znači vrijeme zakašnjenja u odnosu na glavni lik i obrnuto; dakle svakom dijelu pozadine moguće je odrediti drugačiju brzinu pomicanja. Kako bi to bilo moguće, mora se za njega dodati posebna skripta koja će obavljati taj posao.

Deklaracija varijabli za *parallax efekt*:

```
private float length, startpos;
public GameObject cam;
public float parallaxEffect;
```

Da bi se odredio objekt u odnosu na koji će se vršiti parallax efekt, u polje varijable cam dodaje se glavna kamera (engl. MainCamera). Brzinu pomicanja određenih dijelova određujemo varijablom parallaxEffect.

```
    void Start()
    {
        startpos = transform.position.x;
        length = GetComponentInChildren<SpriteRenderer>().bounds.size.x;
    }

    void Update()
    {
        float temp = (cam.transform.position.x * (1 - parallaxEffect));
        float distance = (cam.transform.position.x * parallaxEffect);
        transform.position = new Vector3(startpos + distance, transform.position.y, transform.position.z);

        if (temp > startpos + length)
        {
            startpos += length;
        }
        else if (temp < startpos - length)
        {
            startpos -= length;
        }
    }
}
```

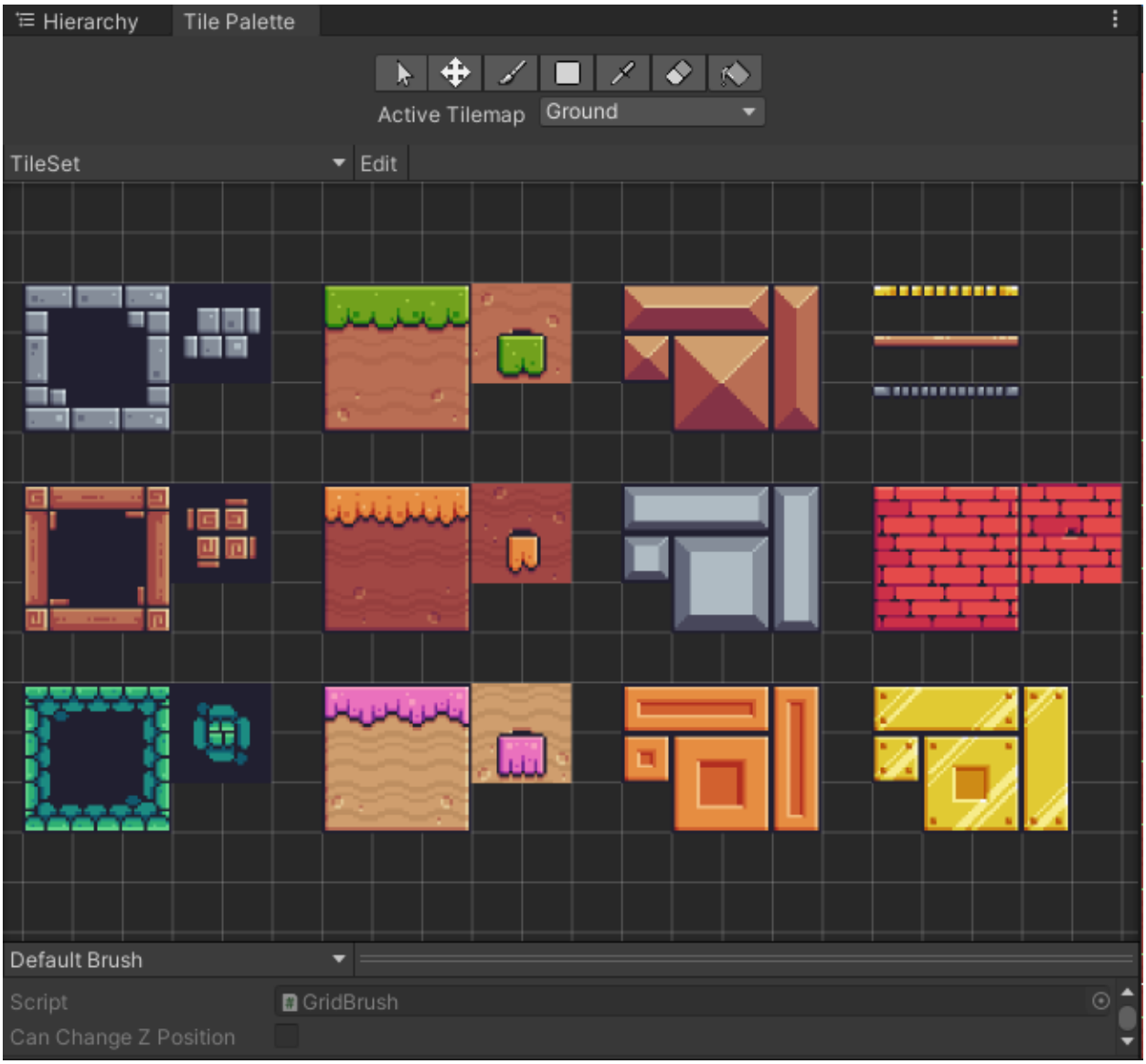
Kod 2. Parallax efekt

Izvor: autor

7.2. Dizajn

Dizajn razina je vrlo važan dio svake video-igre jer o njemu ovisi zanimljivost igre. Dizajnom razina se smatra fizički svijet kojim se igrač kreće prolazeći razine. Što je dizajn složeniji i smisleniji, to je vizualni doživljaj igranja zanimljiviji. Dizajn je moguće kreirati dodavanjem

objekata u scenu jedan po jedan, no ta tehnika zauzima prilično puno vremena pa postoji mogućnost korištenja pločica (engl. tiles). Pločice su zapravo spriteovi kojima je dodana komponenta Tilemap Collider 2D koja omogućuje sudaranje i dodirivanje igrača s podlogom ili bilo kojim dijelom pločica. Pločice su smještene u paletu pločica (engl. Tile Palette) iz koje se jednostavno odabire potrebna pločica i dodaje u scenu.



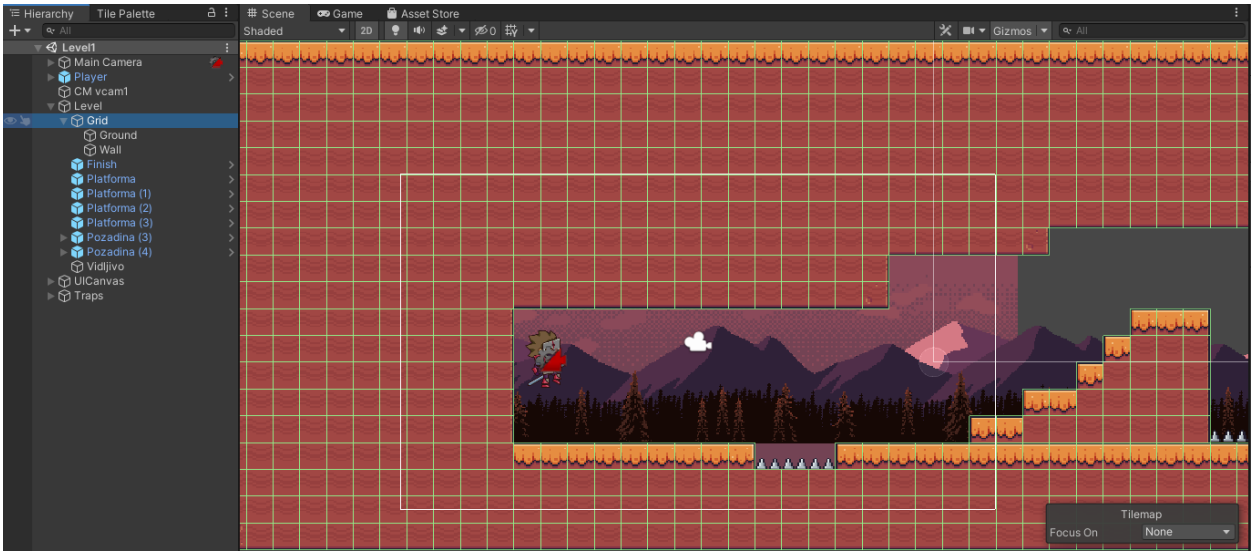
Slika 6. Tile Palette

Izvor: autor

U ovaj projekt je dodano više različitih spriteova zbog ljepšeg dizajna. Mogu se međusobno kombinirati ili kao u ovom projektu svaku scenu urediti određenim pločicama. Spriteovi su

besplatno preuzeti s asset storea. Kod preuzetih pločica može se primijetiti da su sve spojene zajedno pa ih treba izrezati (engl. slice). Za ovaj projekt korištene su pločice veličina 16 x 16 piksela.

Zbog mogućnosti kao što su skok od zida, penjanje po zidu ili skok od tla važno je odrediti što je tlo (engl. ground), a što zid (engl. wall). Dodavanjem pločica u projekt stvara se mreža (engl. grid) koja olakšava dodavanje pločica, u jedno polje u gridu može se postaviti po jedna pločica. Da bi se odredilo što je tlo a što zid, odabirom opcije Active Tilemap u Tile Palette prozoru odabire se sloj (engl. layer), odnosno sloj koji se želi staviti u scenu, tlo ili zid.



Slika 7. Grid sistem

Izvor: autor

U Hierarchy prozoru se može primijetiti da su objektu grid dodana 2 child objekta, Ground i Wall te klikom na jednog od njih prikazuju se samo pločice koje obuhvaća koji objekt.

7.3. Platforme

Platforme su objekti koji pomažu igraču da lakše prođe razinu. Igrač po platformama može skakati, hodati i gibati se jednako kao i po svakoj drugoj podlozi. Kada se platforma doda u scenu, može se postaviti po želji, može se čak i mijenjati veličina. U ovom projektu korištene su statičke i gibljive platforme. Svaka platforma, nebitno je li statička ili gibljiva, mora na sebi sadržavati collider kako bi poprimila određena fizička svojstva. Na platforme je dodan sprite koji je besplatno preuzet s asset storea. Dodana je i Platform Effector 2D komponenta koja omogućava igraču da

prolazi kroz platforme kada to želi. Prolaženje kroz platformu je moguće isključivo u jednome smjeru, i to od tla prema gore.

7.4. Gibljive platforme

Platforme koje stoje u mjestu pomalo su dosadne pa su u ovaj projekt dodane platforme koje se neprestano gibaju. Gibanje se vrši kretanjem platforme do objekata odnosno točaka koje su dodane na željena mjesta. Za svaku gibljivu platformu su dodane po 4 točke.

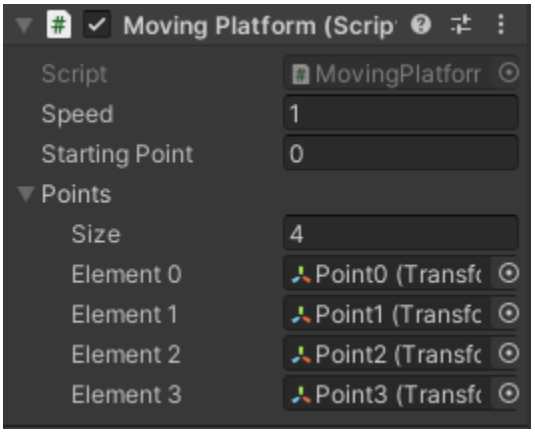
Slika 8. Putanja gibljive platforme

Izvor: autor

Dodana je određena skripta koja upravlja platformom od točke do točke.

```
public float speed; //brzina platforme
public int startingPoint; //pocetna tocka
public Transform[] points; //zavrsna tocka kretanja
```

Deklarirane su varijable kojima se u inspector prozoru određuje brzina platforme, početna i završna točka putanje. Varijabla points je lista u koju su dodani objekti odnosno sve 4 točke koje određuju putanju platforme.



Slika 9. Varijable za podešavanje gibljivih platformi

Izvor: autor

```
    private void Update()
    {
        if (Vector2.Distance(transform.position,
points[i].position)<0.02f)
        {
            i++;
            if (i==points.Length)
            {
                i = 0;
            }
        }
        //kretanje platforme do pozicije sa indexom i
        transform.position =
Vector2.MoveTowards(transform.position, points[i].position, speed
* Time.deltaTime);

    }
```

U Update metodu je dodan kod koji određuje putanju kretanja platforme preko indexa, odnosno oznaka svake točke koje se kreću od 0 do 3.

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
```

```
            collision.transform.SetParent(transform);
        }
```

Kod 3. Kretanje gibljivih platformi

Izvor: autor

Kod kontakta, odnosno sudara platforme s nekom od točaka, mijenja se index i prelazi se na sljedeću točku.

7.5. Jump pad

Trampolin je objekt koji igraču omogućuje odskok po koordinati Y. Kada igrač stupi u fizički kontakt s colliderom trampolina, izvršava se skripta za odskok. Skripta za ovaj objekt je vrlo jednostavna zbog same jednostavnosti objekta.

```
public class JumpPad : MonoBehaviour
{
    [SerializeField] private float bounce;

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {

collision.gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2
.up * bounce, ForceMode2D.Impulse);

        }
    }
}
```

Kod 4. Jump pad

Izvor: autor

Varijablom bounce u inspectoru određuje se jačina skoka, odnosno povećava veličina koordinate Y.

7.6. Kamera

Kamera je najbitniji element svakog projekta. Ona određuje što se vidi na ekranu te ako je loše konfigurirana, može doći do gubitka slike ili nedovoljno dobrog pregleda igre. Postoji više vrsta konfiguriranja kamere, no u ovom završnom radu fokus je na kameri koja prati glavnog igrača kada prolazi kroz razne prepreke i ostalo. Cilj je prilagoditi kameru tako da glavni igrač vidi neke prepreke unaprijed, ali ne previše, kako bi ga svaka prepreka mogla ponovo iznenaditi. Igrač se može vraćati i u suprotnom smjeru, kamera će ga i dalje slijediti. Glavna kamera je po početnim postavkama poprilično loša i ne zadovoljava potrebe projekta, pa je na nju dodan dodatak Cinemachine.

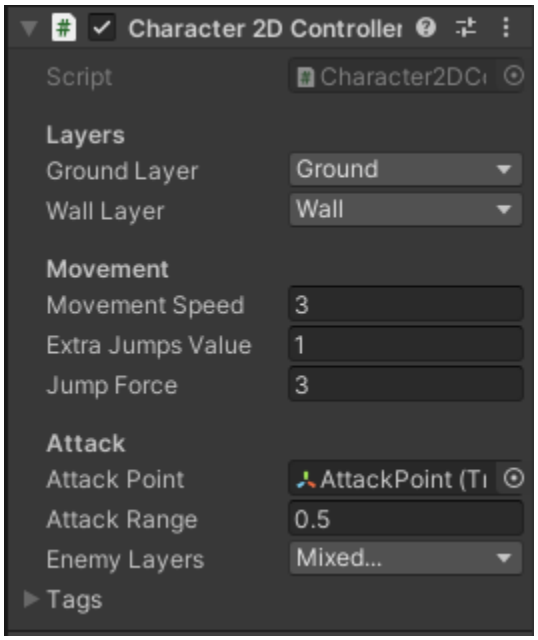
7.6.1. Cinemachine

Cinemachine je alat koji Unity nudi kao pomoć oko glavne kamere. Kameru je moguće ručno programirati pisanjem kodova, no ta metoda je prilično kompliciranа i zauzima previše vremena, dok se cinemachine može instalirati u nekoliko klikova. Instalacija je prilično jednostavna, klikne se na prozor Window i zatim odabire opcija Package Manager. Otvara se dijaloški okvir u kojem se vide različiti dodatci koji se mogu instalirati u projekt. Otvaranjem Cinemachine paketa prikazuje se opcija Install. Nakon uspješne instalacije u hierarchy prozoru kreira se novi objekt pod nazivom CM vcam1 što je zapravo cinemachine kamera. U glavnu kameru se dodaje CinemachineBrain komponenta koja zapravo kontrolira cinemachine kameru. Ako cinemachine kamera nije dodana na glavnu kameru, neće se primijeniti na projekt. U objektu CM vcam1 nalazi se mnoštvo opcija koje je moguće podešavati. Potrebno je potrošiti ponešto vremena da se postigne željeni efekt kamere pošto ima poprilično dosta opcija, no uz malo truda može se stvoriti vrlo dobar rezultat. Dok CinemachineVirtualCamera komponenta služi za prilagođavanje opcija glavne kamere, Cinemchine Confiner komponenta služi za određivanje granica do kojih kamera može ići. U ovom projektu dodan je polygon collider čija je veličina prilagođena svakoj razini posebno zbog različitih veličina razina pa samim time i različitim poljima vidika.

7.7. Mehanika

Mehanika podrazumijeva kretanje igrača, neprijatelja, kamere i ostale elemente koji imaju neku mehaničku svrhu što ju čini jednim od najbitnijih elemenata svake video-igre. Svaki objekt u igri na sebi sadrži komponentu script u kojoj se nalazi određena skripta koja upravlja tim objektom i omogućuje mu obavljanje određenih radnji. Bez mehanike igra ne bi imala smisla jer igrač ne bi mogao prisustvovati igri.

Glavni igrač ima mogućnost skakanja, kretanja lijevo, desno, dvostruki skok, napad mačem, propadanje i penjanje kroz platforme i penjanje po zidu. Za svaki dio igračevih mehaničkih sposobnosti dodan je određeni dio koda.



Slika 10. Sučelje varijabli za mehanizam glavnog igrača

Izvor: autor

U inspector prozoru mogu se mijenjati vrijednosti za mehanizam glavnog igrača. Da bi to bilo moguće, moraju se deklarirati određene varijable.

Za svaku radnju koja je navedena dodana je određena funkcija koja ju izvršava, pa je dodana Update metoda koja poziva potrebnu funkciju.

```
 private void Update()
{
        WallJump();
        Attack();
```

```
            Movement();
            Flip();
            ExtraJump();
```

Ovdje se nalaze i parametri za pokretanje animacija za trčanje i stajanje na miru.

```
        //animator parametri
            anim.SetBool("run", movement != 0);
            anim.SetBool("grounded", isGrounded());
    }
```

Kod 5. Update metode mehanike igrača

Izvor: autor

7.7.1. Provjere podloga

Da bi igrač imao sve sposobnosti koje su ranije navedene, mora se konstantno provjeravati gdje se igrač nalazi, je li na podu, na zidu ili negdje drugdje. Dodane su varijable kojima je označen zid ili tlo.

```
    [Header("Layers")]
    [SerializeField] private LayerMask groundLayer;
    [SerializeField] private LayerMask wallLayer;
```

Umetanjem koda za provjeru tla i zida dobivaju se informacije koje kasnije mogu pomoći u programiranju određenih mehaničkih funkcijia igrača.

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

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

Kod 6. Provjera podloge i zida

Izvor: autor

Ovdje se po prvi puta pojavljuje pojam raycast. Raycast su nevidljive zrake koje vraćaju informacije čega se igrač dodiruje; ako zraku nešto presječe, vraća informaciju da se na tom mjestu nešto nalazi pa se s tom informacijom mogu manipulirati razne fizičke funkcije.

7.7.2. Horizontalno kretanje

Horizontalno kretanje je vrlo jednostavna funkcija čiju svrhu kazuje samo ime. Igrač se kreće u lijevom i desnom smjeru, a to omogućuje držanje tipki A ili D ili strelica lijevo (engl. left) ili strelica desno (engl. right).

```
    public float MovementSpeed = 1;
        bool facingRight = true;
    private float movement;
```

Deklaracijom varijabli određuje se brzina kretanja i smjer gledanja igrača.

```
    void Movement()
   {
        //horizontalno gibanje
        movement = Input.GetAxis("Horizontal");
        transform.position += new Vector3(movement, 0, 0) *
Time.deltaTime * MovementSpeed;
    }
```

Kod 7. Horizontalno gibanje igrača

Izvor: autor

Kako bi sam sprite igrača bio okrenut u smjeru kretanja, dodana je funkcija flip.

```
 void flip() //okret igraca
{
    facingRight = !facingRight;
    transform.Rotate(0f, 180f, 0f);
}
```

Potrebno je dodati funkciju koja će se izvršavati promjenom smjera kretanja igrača i u trenutku promjene smjera pozvati funkciju flip da se okretanje spritea izvrši.

```
 void Flip()
{
      //okretnanje igraca u smjeru kretanja
      if (movement < 0 && facingRight)
      {
            flip();
      }
      else if (movement > 0 && !facingRight)
      {
            flip();
      }
 }
```

Kod 8. Okret sprite-a igrača

Izvor: autor

7.7.3. Skok

Bitan element kretanja je skok (engl. jump). Pošto u igri postoje platforme na raznim visinama i prepreke koje se moraju preskočiti, potrebno je igraču dodati funkciju skoka.

```
private int extraJumps;
public int extraJumpsValue;
public float JumpForce = 1;
```

```
    private float wallJumpCooldown;
```

Deklaracija varijabli za skok određuje jačinu skoka, broj višestrukih skokova i skok od zida.

```
    private void Jump() //skok
    {
        if (isGrounded())
        {
            _rigidbody.velocity = new
Vector2(_rigidbody.velocity.x, JumpForce);
            anim.SetTrigger("jump");
        }
```

Ako je igrač prizemljen, moguće je izvršiti kod za skok. U ovoj funkciji nalazi se i parametar koji pokreće animaciju skoka.

```
        else if (onWall() && !isGrounded()) //penjanje po zidu
        {
            if (movement == 0)
            {
                _rigidbody.velocity = new Vector2(-
Mathf.Sign(transform.localScale.x) * 10, 0);
            }
            else
            {
                _rigidbody.velocity = new Vector2(-
Mathf.Sign(transform.localScale.x) * 3, 6);
            }
        }
    }
```

Kod 9: Funkcija za skok igrača

Izvor: autor

7.7.4. Penjanje po zidu

Ako igrač nije prizemljen i ako je u kontaktu sa zidom, moguće je penjanje po zidu. Dodana je također i funkcija odskoka od zida.

```
    void WallJump()
    {
            //skok od zida
            if (wallJumpCooldown > 0.2f)
            {
                  _rigidbody.velocity = new Vector2(movement *
MovementSpeed, _rigidbody.velocity.y);

                  if (onWall() && !isGrounded())
                  {
                        _rigidbody.gravityScale = 0;
                        _rigidbody.velocity = Vector2.zero;
                  }
                  else
                        _rigidbody.gravityScale = 1;

                  if (Input.GetKey(KeyCode.Space))
                        Jump();
            }
            else
                  wallJumpCooldown += Time.deltaTime;
      }
```

Kod 10. Funkcija za skok od zida

Izvor: autor

Dakle, ako igrač nije prizemljen i u kontaktu je sa zidom, pritiskom na tipku Space izvršava se funkcija Jump i igrač odskače od zida.

7.7.5. Višestruki skok

Kako bi igra bila još zanimljivija, dodana je funkcija za višestruki skok. U inspector prozoru može se odrediti broj višestrukih skokova; u ovom radu je to stavljeno na jedan jer se prvi skok ne računa pošto to nije višestruki skok.

```
    void ExtraJump()
    {
            //visestruki skok
            if (isGrounded() == true)
            {
                  extraJumps = extraJumpsValue;
            }

            if (Input.GetKeyDown(KeyCode.Space) && extraJumps > 0)
            {
                  _rigidbody.velocity = Vector2.up * JumpForce;
                  extraJumps--;
            }
            else    if    (Input.GetKeyDown(KeyCode.UpArrow)    &&
extraJumps == 0 && isGrounded() == true)
            {
                  _rigidbody.velocity = Vector2.up * JumpForce;
            }
```

U ovoj funkciji se nalazi i poziv funkcije Jump s tipkom Space. Kada se jednom pritisne Space, izvršava se samo Jump funkcija, a kada se višestruko pritisne, funkcija Jump će se izvršiti više puta, a to omogućava gornji dio koda.

```
            if (Input.GetKey(KeyCode.Space) && isGrounded())
            {
                  Jump();
            }

      }
```

Kod 11. Funkcija za višestruki skok

Izvor: autor

7.7.6. Napad

Dodana je mogućnost napada glavnog igrača kako bi se zaštitio i napao neprijatelje. Potrebno je stvoriti novi objekt koji je centar napada te deklarirati varijable potrebne da bi se napad izvršio.

```
[Header("Attack")]
public Transform attackPoint;
public float attackRange = 0.5f;
public LayerMask enemyLayers;
private GameObject ozljeda;
[SerializeField] string[] tags;
```

U inspector prozoru određena je udaljenost napada, dakle daljina do koje igrač prilikom izvršenja napada vrši određene ozljede, odnosno oduzimanje života neprijateljima. Također se moraju odrediti slojevi neprijatelja kako bi ih Unity uopće prepoznao prilikom napada. Varijabla tags je lista oznaka neprijatelja koje je moguće napasti.

Da bi bilo moguće izvršiti napad, dodana je funkcija attack.

```
void attack()
{
   //radijus u kojem mač oštećuje
            Collider2D[]            hitEnemies            =
Physics2D.OverlapCircleAll(attackPoint.position,     attackRange,
enemyLayers);
```

Prostor u kojem igrač ozljeđuje protivnika određen je varijablom hitEnemies. Prostor je određen veličinom radijusa kada igrač zamahne mačem.

```
   //animacija napada
if (!isGrounded())
{
```

```
                anim.SetTrigger("jumpAttack");
        }
        else
        {
                anim.SetTrigger("attack");
           }
```

Postoji i mogućnost napada u zraku kada igrač nije prizemljen, pa su tako dodani parametri za izvršavanje animacija kada je igrač prizemljen, a kada nije.

```
            //damage
            foreach (Collider2D enemy1 in hitEnemies)
            {
                  for (int i = 0; i < tags.Length; i++)
                  {
                  if (enemy1.CompareTag(tags[i]))
                  {
                              ozljeda                                    =
GameObject.FindWithTag(tags[i]);

      ozljeda.GetComponent<Health>().TakeDamage(1);

                        }
                  }
            }
      }
```

Kod 12. Napad igrača

Izvor: autor

Kao što je prije spomenuto, svaki neprijatelj je označen svojom oznakom te kada igrač napadne određenog neprijatelja, oduzima mu određen broj života. Dodana je petlja foreach koja pregledava listu oznaka neprijatelja, te ako se napadnuti neprijatelj nalazi na listi, oduzima mu po jedan život.

Pritiskom lijeve tipke miša poziva se funkcija attack i izvršava se napad.

```
void Attack()
    {
           //klikom misa pozivamo funkciju attack
           if (Input.GetKeyDown(KeyCode.Mouse0))
           {
                 attack();
           }
     }
```

Kod 13. Pozivanje funkcije za napad klikom miša

Izvor: autor

8. NEPRIJATELJI

Neprijatelji (engl. enemy) su negativci koji su dodani u ovaj projekt kako bi otežali glavnom liku sam prelazak razina. Na svakog neprijatelja dodane su slične komponente kao i kod glavnog lika, samo što su skripte potpuno drugačije zbog same svrhe neprijatelja. Neprijatelji se kreću između dvije zadane točke koje se određuju u samom editoru, označene su s LeftEdge i RightEdge objektima. Sprite neprijatelja preuzet je s asset storea, a izgleda kao mali vitez s mačem. Za kretanje neprijatelja dodana je sljedeća skripta.

Deklaracija varijabli za kretanje neprijatelja:

```
[Header("Patrol Points")]
[SerializeField] private Transform leftEdge;
[SerializeField] private Transform rightEdge;

[Header("Enemy")]
[SerializeField] private Transform enemy;

[Header("Movement parameters")]
[SerializeField] private float speed;
private Vector3 initScale;
private bool movingLeft;

[Header("Idle Behaviour")]
```

```
    [SerializeField] private float idleDuration;
    private float idleTimer;

    [Header("Enemy Animator")]
    [SerializeField] private Animator anim;
```

Varijable pod naslovom Patrol Points omogućavaju određivanje dviju točaka između kojih će se kretati neprijatelj. Varijabla enemy označuje neprijatelja na kojeg se ova skripta odnosi. Movement Parameters varijablama je određena neprijateljeva brzina kretanja i smjer kretanja. Idle Behaviour varijable nude vremensko podešavanje kako dugo će neprijatelj stajati na mirnoj poziciji, do kada se neće okrenuti u suprotnom smjeru i nastaviti hodati (engl. patrolling). Anim ima svrhu određivanja neprijateljevog animatora.

U Update metodu je dodan kod za kretanje neprijatelja.

```
private void Update()
{
    if (movingLeft)
    {
        if (enemy.position.x >= leftEdge.position.x)
            MoveInDirection(-1);
        else
            DirectionChange();
    }
    else
    {
        if (enemy.position.x <= rightEdge.position.x)
            MoveInDirection(1);
        else
            DirectionChange();
    }
}
```

Kod 14. Kretanje neprijatelja

Izvor: autor

Kako bi neprijatelj mijenjao smjer i kretao se u pravom smjeru, dodana je funkcija DirectionChange.

```
private void DirectionChange()
    {
        anim.SetBool("moving", false);
        idleTimer += Time.deltaTime;

        if (idleTimer > idleDuration)
            movingLeft = !movingLeft;
    }
```

Kod 15. Funkcija promjene smjera kretanja neprijatelja

Izvor: autor

Kada neprijatelj odredi smjer kretanja, počinje s kretnjom.

```
private void MoveInDirection(int _direction)
    {
        idleTimer = 0;
        anim.SetBool("moving", true);

        //Make enemy face direction
        enemy.localScale = new Vector3(Mathf.Abs(initScale.x) *
_direction,
            initScale.y, initScale.z);

        //Move in that direction
        enemy.position = new Vector3(enemy.position.x +
Time.deltaTime * _direction * speed,
            enemy.position.y, enemy.position.z);
    }
```

Kod 16 Funkcija za kretanje neprijatelja u određenom smjeru

Izvor: autor

8.1. Napad neprijatelja

Neprijatelji glavnog igrača ozljeđuju mačem. Kada se glavni igrač dovoljno približi neprijatelju, neprijatelj staje na mjestu i vrši se animacija napadanja mačem. Svaki pogođen napad glavnom igraču oduzima po jedno srce, dok svaki prelazak igrača pokraj neprijatelja oduzima po pola srca. Kako bi to bilo moguće, dodana je potrebna skripta.

Deklaracija varijabli unutar MeleeEnemy skripte.

```
[Header("Attack Parameters")]
[SerializeField] private float attackCooldown;
[SerializeField] private float range;
[SerializeField] private int damage;

[Header("Collider Parameters")]
[SerializeField] private float colliderDistance;
[SerializeField] private BoxCollider2D boxCollider;

[Header("Player Layer")]
[SerializeField] private LayerMask playerLayer;
private float cooldownTimer = Mathf.Infinity;

private Animator anim;
private Health playerHealth;
private EnemyPatrol enemyPatrol;
```

Attack Parameters varijablama određena je jačina, udaljenost i pauza između višestrukog napada. Collider Parameters varijable određuju neprijateljev box collider i udaljenost napada od samog neprijatelja u kojem se ozljeđuje glavni igrač kada stoji unutar njega. Player Layer je samo sloj kojim je označen glavni igrač, tako da Unity zna kada glavnom igraču oduzeti živote prilikom napada. Ako se pogrešno označi sloj, na primjer ako je označeno da je Player Layer nešto drugo, tada se glavnom igraču neće oduzimati životi, nego će se oduzimati onome što je označeno pod tim slojem.

U Update metodu je dodan kod potreban da bi se napad pravilno izvršio.

```
private void Update()
{
    if (enemyPatrol!=null)
```

```
            {
                enemyPatrol.enabled = !PlayerInSight();
            }
            cooldownTimer += Time.deltaTime;

            //napad samo kada je igrac na vidiku
            if (PlayerInSight())
            {
                if (cooldownTimer >= attackCooldown)
                {
                    cooldownTimer = 0;
                    anim.SetTrigger("meleeAttack");
                }

            }

        }
```

Kod 17. Napad neprijatelja

Izvor: autor

Kako bi neprijatelj uočio glavnog igrača, dodana je funkcija PlayerInSight. Pomoću ove funkcije neprijatelj zna kada treba izvršiti napad.

```
        private bool PlayerInSight()
        {
            RaycastHit2D hit =
                Physics2D.BoxCast(boxCollider.bounds.center + transform.right * range * transform.localScale.x * colliderDistance,
                new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y, boxCollider.bounds.size.z),
                0, Vector2.left, 0, playerLayer);

            if (hit.collider!=null)
            {
```

```
            playerHealth = hit.transform.GetComponent<Health>();
        }

        return hit.collider!=null;
    }
```

Kod 18. Funkcija za uočavanje glavnog igrača

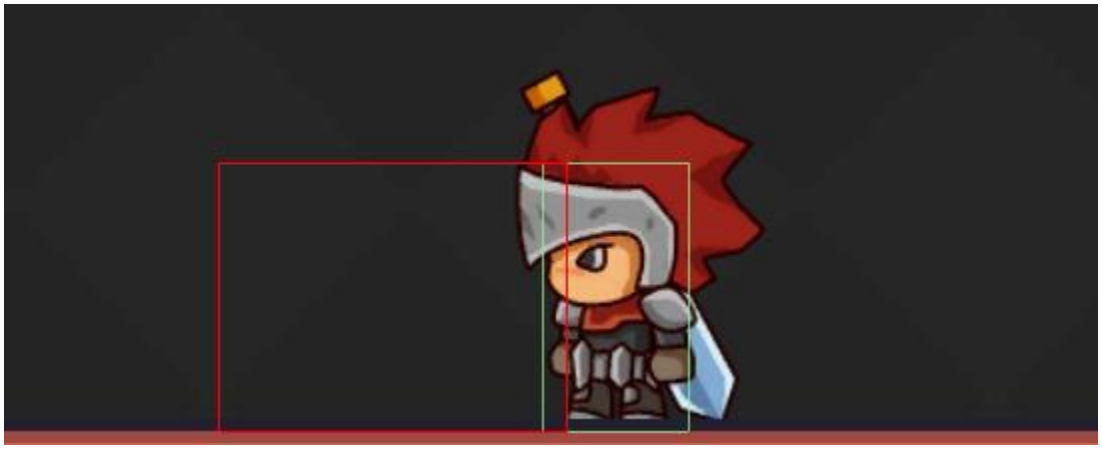
Izvor: autor

Dodana je funkcija OnDrawGizmos koja vizualno prikazuje mjesto i veličinu ranije spomenutog polja koje služi neprijatelju za prepoznavanje glavnog igrača kako bi ga mogao napasti.

```
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(boxCollider.bounds.center + transform.right * range * transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y, boxCollider.bounds.size.z));
    }
```

Kod 19. Funkcija za vizualni prikaz collidera za napad

Izvor: autor



Slika 11. Collider neprijatelja i područje napada

Izvor: autor

Kada je neprijatelj uočio glavnog igrača i glavni igrač se nalazi u collideru za napad, funkcija DamagePlayer omogućava da se glavnom igraču oduzme određen broj života koji se može odrediti u inspectoru.

```
private void DamagePlayer()
{
    //ako je igrac u range-u enemy mu zadaje damage
    if (PlayerInSight())
    {
        playerHealth.TakeDamage(damage);

    }
}
```

Kod 20. Funkcija za ozljedu glavnog igrača

Izvor: autor

9. ZAMKE

Zamke (engl. traps) su objekti koji igraču otežavaju prelazak razina sa svojim određenim funkcijama. Svaka zamka obavlja svoju zadaću koja je određena kroz skripte. U ovom radu postoje bodlje (engl. spikes), pila (engl. saw), vatrena zamka (engl. firetrap), zamka sa strelicama (engl. arrow trap) i zamka koja se može svrstati i u kategoriju neprijatelja Spikehead. Dodan je i neprijatelj Maskman koji ne ozljeđuje igrača, već mu smeta na putu. Zamke su na svakoj razini drugačije raspoređene, sukladno dizajnu određene razine. Svaka zamka oduzima onoliko života igraču koliko je određeno u inspector prozoru.

9.1. Pila

Pila (engl. saw) je objekt koji izgleda kao kružna pila koja se vrti, te kada igrač stupi u kontakt s njezinim colliderom, oduzima mu određeni broj života.

```
[SerializeField] private float movementDistance;
```

```
    [SerializeField] private float speed;
    private bool movingLeft;
    private float leftEdge;
    private float rightEdge;
    private Animator anim;
```

Deklaracija varijabli omogućava podešavanje brzine kretanja, duljinu kretanja i broj života koje oduzima igraču.

```
    private void Update() //kamo se krece pila
    {
        if (movingLeft)
        {
            if (transform.position.x > leftEdge)
            {
                transform.position = new
Vector3(transform.position.x - speed * Time.deltaTime,
transform.position.y, transform.position.z);
            }
            else
            {
                movingLeft = false;
                if (CompareTag("Enemy1"))
                {
                    transform.Rotate(0f, 180f, 0f);
                }
            }
        }
        else
        {
            if (transform.position.x < rightEdge)
            {
                transform.position = new
Vector3(transform.position.x + speed * Time.deltaTime,
transform.position.y, transform.position.z);
```

```
            }
            else
            {
                movingLeft = true;
                if (CompareTag("Enemy1"))
                {
                    transform.Rotate(0f, 180f, 0f);
                }
            }

        }
    }
```

U Update metodu je dodan dio koda zaslužan da se pila giba lijevo i desno, te kada dostigne dovoljnu udaljenost koja je zadana u inspector prozoru, pila se rotira za 180 stupnjeva te kreće u suprotnom smjeru. Kada u suprotnom smjeru dostigne ponovo tu određenu udaljenost, opet se okreće za 180 stupnjeva u prethodnom smjeru i tako nastavlja s kretanjem. Nekim pilama nije potrebno kretanje pa je varijabla movementDistance postavljena na 0.

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        base.OnTriggerEnter2D(collision);
    }
}
```

Kod 21. Kretanje pile

Izvor: autor

Kada pila dotakne igrača, izvršava se skripta EnemyDamage i igraču se oduzima određeni broj života.

Slika 12. Pila

Izvor: autor

9.1.1. Inheritance

Inheritance je mogućnost programskog jezika C# da naslijedi polja i metode iz jedne u drugu klasu. Podjela inheritance koncepta dijeli se na izvedbenu klasu (engl. derived class) koja je dijete (engl. child) i osnovnu klasu (engl. base class) koja je roditelj (engl. parent). Izvedbena klasa je klasa koja se nasljeđuje iz neke druge klase, dok je osnovna klasa ona iz koje se nasljeđuje. Za nasljeđivanje se koristi : simbol. [11] U ovom radu se inheritance koristi kod projektila koji predstavljaju strelice u zamkama, pile i neprijatelja Spikehead koji igrača ozljeđuje sudaranjem.

```
public class EnemyDamage : MonoBehaviour
{
    //glavnom igracu oduzimamo health
    //INHERITANCE
    [SerializeField] protected float damage;

    protected void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag=="Player")
        {
            collision.GetComponent<Health>().TakeDamage(damage);
        }
    }
}
```

Kod 22. Oduzimanje zdravlja glavnom liku

Izvor: autor

Na početkku skripte za strelice EnemyProjectile pojavljuje se slučaj inheritance i kasnije u metodi za sudaranje strelica u neki objekt gdje se koristi ključna riječ base da bi se identificirala baza, odnosno osnovna funkcija u EnemyDamage skripti.

```
public class EnemySideways : EnemyDamage
        base.OnTriggerEnter2D(collision);
```

Kod 23. Inheritance kod skripte za strelice

Izvor: autor

Dakle, kada strelica dotakne glavnog igrača, počinje se koristiti EnemyDamage skripta koja tada preko skripte Health oduzima određeni broj života glavnom igraču. Isto vrijedi i za ostale objekte na kojima se primjenjuje inheritance.

9.2. Spikehead

Spikehead je zamka, odnosno neprijatelj koji prati igrača kada stupi u njegovo vidno polje. Na sebi sadrži bodlje koje ozljeđuju igrača prilikom fizičkog kontakta.

```
[Header("Spikhead postavke")]
[SerializeField] private float speed;
[SerializeField] private float range;
[SerializeField] private float checkDelay;
[SerializeField] private LayerMask playerLayer;
```

Deklarirane su varijable kojima se određuje spikeheadova jačina ozljede, brzina kretanja, udaljenost koja je zapravo polje u koje, kada dođe igrač, spikehead ga počinje slijediti, i checkDelay koji određuje vrijeme reagiranja. Da bi spikehead prepoznao igrača, dodana je varijabla playerLayer koja određuje sloj igrača.

```
private void CheckForPlayer()
{
```

```
            CalculateDirections();

            //provjera da li spikhead vidi igraca u sva 4 smjera
            for (int i = 0; i < directions.Length; i++)
            {
                Debug.DrawRay(transform.position,       directions[i],
Color.red);
                RaycastHit2D                    hit                    =
Physics2D.Raycast(transform.position,    directions[i],    range,
playerLayer);

                if (hit.collider!=null && !attacking)
                {
                    attacking = true;
                    destination = directions[i];
                    checkTimer = 0;
                }
            }
        }

        private void CalculateDirections()
        {

            directions[0] = transform.right * range; //u desno
            directions[1] = -transform.right * range; //u lijevo
           directions[2] = transform.up * range; //prema gore
           directions[3] = -transform.up * range; //prema dolje

        }
```

Funkcija CheckForPlayer služi da bi spikehead uočio igrača. Kod toga mu pomaže funkcija *CalculateDirections* koja provjerava smjerove lijevo, desno, gore, dolje.

```
        private void Stop()
        {
            {
```

```
            destination = transform.position;
            attacking = false;
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        base.OnTriggerEnter2D(collision);
        Stop();//zaustavi spikheada kada se dotakne neceg
    }
```

Funkcija Stop zaustavlja spikeheada kada se dotakne igrača.

```
    private void Update()
    {
        //samo ako spikhead napada samo onda ide na destination
        if (attacking)
        {
            transform.Translate(destination * Time.deltaTime *
speed);

        }
        else
        {
            checkTimer += Time.deltaTime;
            if (checkTimer>checkDelay)
            {
                CheckForPlayer();
            }
        }
    }
```

Kod 24. Spikehead

Izvor: autor

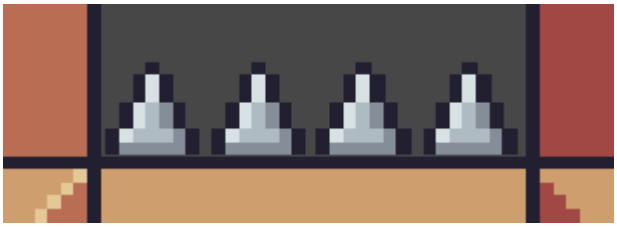
U Update metodu dodan je kod koji upravlja spikeheadom. Kada vidi igrača, ide na njegovo odredište i napada ga, a kada nije u njegovom dometu, provjerava sva 4 smjera.

Slika 13. Spikehead

Izvor: autor

9.3. Bodlje

Bodlje (engl. spikes) su najjednostavnija zamka u ovom projektu. Kada igrač stupi u fizički kontakt s bodljama, odnosno collideri se dotaknu, oduzimaju mu određeni broj života. Nema posebne skripte, već se koristi samo skripta EnemyDamage i u inspector prozoru se unosi broj koliko života će oduzeti glavnom igraču.



Slika 14. Bodlje

Izvor: autor

9.4. Vatrena zamka

Vatrena zamka ispušta plamen koji oštećuje igrača cijelo vrijeme kada je u fizičkom kontaktu s njime. Ako igrač stoji u plamenu, plamen će ga ozljeđivati sve dok se ne ugasi.

```
    [SerializeField] private float damage;
    [Header("Firetrap Timers")]
    [SerializeField] private float activationDelay;
    [SerializeField] private float activeTime;
```

Varijablama je određena jačina ozljede, vrijeme potrebno da se plamen aktivira i vrijeme trajanja plamena.

```
    private IEnumerator ActivateFireTrap()
    {
        //zamka postaje crvena da znamo da smo ju upalili
        triggered = true;
        sprite.color = Color.red;

        //ceka se delay, aktivira se zamka, upali se animacija,
boja se vraca u normalno stanje
        yield return new WaitForSeconds(activationDelay);
        sprite.color = Color.white;
        active = true;
        anim.SetBool("activated",true);

        //pricekamo sekunde, deaktiviramo zamku i resetiramo
animacije i sve varijable
        yield return new WaitForSeconds(activeTime);
        active = false;
        triggered = false;
        anim.SetBool("activated", false);

    }
```

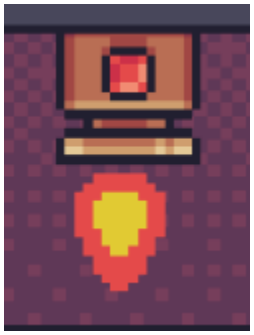
Ovdje se po prvi puta koriste korutine (eng. Coroutine); one služe da se funkcija zaustavi dok se ne ispuni zadani uvjet, u ovom slučaju je to efekt čekanja na aktivaciju paljenja plamena i ponovnog gašenja. Također zamka postaje crvena kada je u upotrebi. Za izvođenje korutina koristi se funkcija IEnumerator.

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            playerHealth = collision.GetComponent<Health>();
            if (!triggered)
            {
                //zamka se ukljuci
                StartCoroutine(ActivateFireTrap());
            }
            if (active)
            {

collision.GetComponent<Health>().TakeDamage(damage);
            }
        }
    }
```

Kod 25. Zamka sa plamenom

Izvor: autor

Kada je igrač u kontaktu sa zamkom, izvršava se funkcija ActivateFireTrap i traje sve dok se plamen ne ugasi ili se igrač ne odmakne.



Slika 15. Zamka sa plamenom

Izvor: autor

9.5. Zamka sa strelicama

Prije postavljanja same zamke dodana je skripta koju će koristiti strelice koje zamka ispaljuje da bi uopće imale svoju funkciju.

```
[SerializeField] private float speed;
[SerializeField] private float resetTime;
```

Varijablama se određuje jačina ozljede, brzina strelice i vrijeme putovanja strelice ukoliko se ne dotakne nečeg i nestane.

```
public void ActivateProjectile()
{
    hit = false;
    lifetime = 0;
    gameObject.SetActive(true);
    boxCollider.enabled = true;
}
```

Aktivacija strelice vrši se funkcijom ActivateProjectile.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    hit = true;
```

```
            base.OnTriggerEnter2D(collision);
            //zanemari collider od pollygon collidera, Vidljivo(game
    object)
            Physics2D.IgnoreLayerCollision(11, 2);
            boxCollider.enabled = false;


            if (anim != null)
            {
                anim.SetTrigger("explode");
            }
            else
            {
                //unisti strelicu
                gameObject.SetActive(false);
            }

        }
```

Kada strelica dotakne igrača, oduzima mu određeni broj života i nestaje, točnije, vrši se animacija eksplozije i objekt nestaje.

```
        private void Update()
        {
            if (hit)
            {
                return;
            }

            float movementSpeed = speed * Time.deltaTime;
            transform.Translate(movementSpeed, 0, 0);


            lifetime += Time.deltaTime;
```

```
        if (lifetime>resetTime)
        {
            gameObject.SetActive(false);
        }
    }
```

Kod 26. Strelice

Izvor: autor

U Update metodu dodan je kod koji služi za računanje brzine strelice i duljinu putovanja.

Kada su strelice isprogramirane, dodaje se zamka koja u sebi sadrži strelice kao child objekte, odnosno ArrowHolder u kojem su one pohranjene.

```
    [SerializeField] private float attackCooldown;
    [SerializeField] private Transform firePoint;
    [SerializeField] private GameObject[] arrows;
```

Varijablama se određuje vremenska pauza između ispaljivanja strelica, smjer pucanja i lista u koju su strelice dodane.

```
    private void Attack()
    {
        cooldownTimer = 0;
        arrows[FindArrow()].transform.position                    =
firePoint.position;

arrows[FindArrow()].GetComponent<EnemyProjectile>().ActivateProje
ctile();

    }

    private int FindArrow()
    {
        for (int i = 0; i < arrows.Length; i++)
        {
            if (!arrows[i].activeInHierarchy)
```

```
            {
                return i;
            }
        }
        return 0;
    }
```

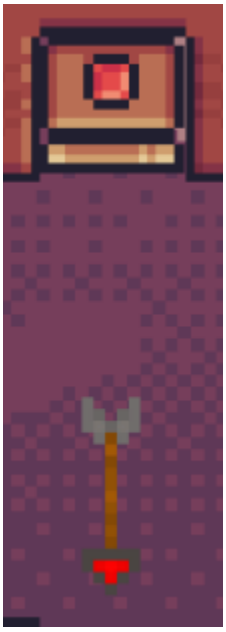
Funkcija Attack služi za ispaljivanje strelica preko funkcije FindArrow koja provjerava listu u kojoj su pohranjene strelice te ih ispucava.

```
private void Update()
{
    cooldownTimer += Time.deltaTime;
    if (cooldownTimer>=attackCooldown)
    {
        Attack();
    }
}
```

Kod 27. Zamka sa strelicama

Izvor: autor

U Update metodi određuje se vremenska stanka između napada.



Slika 16. Zamka sa strelicama

Izvor: autor

9.6. Maskman

Maskman je protivnik koji svojim kretanjem otežava put igraču. Njegova svrha je samo odguravanje igrača ponekad u neke zamke, a ponekad samo ometanje na putu. Na sebi sadrži prilagođen Box Collider 2D koji je iznad glave malo izdužen tako da je Maskmana teže preskočiti. Skripta je jednaka kao i kod pile, ali je varijabla damage postavljena na nulu tako da se igrač ne ozljeđuje, već samo gura.

Slika 17. Maskman

Izvor: autor

10. ZDRAVLJE

Kako bi sukob neprijatelja i glavnog igrača (isto kao i sukob zamki i glavnog igrača) imao smisla, svi živi likovi moraju biti ograničeni nekakvim zdravljem (engl. health). Glavni igrač nosi 4 srca koja označavaju njegovo trenutno zdravlje, dok neprijatelji nose po 5 srca. U inspectoru se određuje broj srca za pojedini objekt. Bez skripte za zdravlje glavni lik ne bi mogao umrijeti i ne bi postojala mogućnost eliminacije protivnika, pa je zbog toga taj dio vrlo važan.

Deklaracija varijabli za Health skriptu:

```
    [Header("iFrames")]
    [SerializeField] private float iFramesDuration;
    [SerializeField] private int numberOfFlashes;
    private SpriteRenderer spriteRend;

    [Header("Health")]
    [SerializeField] private float startingHealth;

    [Header("Components")]
    [SerializeField] private Behaviour[] components;
    private bool invulnerable;

    [SerializeField] string[] tags;

    public float currentHealth { get; private set; } //get=mozemo
pristupiti trenutnom zdravlju iz bilo koje druge skripte
    //private, set=trenutno zdravlje mozemo postaviti samo u ovoj
skripti
    private Animator anim;
    private bool dead;
```

iFrames varijable se koriste za određivanje bliještanja glavnog lika kada ga nešto ozlijedi. Na primjer, ako ga neprijatelj pogodi mačem ili ga oštrica pile dotakne, glavni lik će početi bliještati bojom koja je zadana u inspectoru. Određuje se boja, vrijeme i broj ponavljanja bliještanja, isto vrijedi i za neprijatelje. Varijabla startingHealth je jedna od najkorisnijih u ovoj skripti. Pomoću nje se u inspektor prozoru određuje koliko života, odnosno srca će na početku imati glavni lik, a

koliko pojedini neprijatelj. Components varijable služe za stvaranje liste objekata na kojima će se vršiti određena radnja. Pomoću tih varijabli se određuje skripta koja će se ugasiti za određen objekt kada on umre. Na primjer, kada glavni lik eliminira neprijatelja, na neprijateljevom objektu se gasi skripta za napad i on umire. Kod neprijatelja postoji lista koja se sastoji od oznaka (engl. tags) koja služi za identifikaciju svakog neprijatelja, pa su tako redom dodane oznake Enemy2, Enemy3, Enemy4 i tako sve do Enemy25. U ovom radu su dodane 23 moguće oznake neprijatelja; to se vrlo lako može dodati ili umanjiti brisanjem ili dodavanjem oznaka u Tag prozoru u Unityu i zatim dodavanjem na tu listu. CurrentHealth varijabla služi za pohranjivanje trenutne vrijednosti zdravlja te se kasnije koristi u programskom kodu za daljnje manipuliranje koje je potrebno za određene radnje. Dead varijabla je podatkovnog tipa bool što znači istinito ili lažno (engl. true or false), a služi za određivanje je li neki objekt živ ili mrtav.

```
    public void TakeDamage(float _damage)
    {

        if (invulnerable)
        {
            return;
        }

        currentHealth = Mathf.Clamp(currentHealth - _damage, 0,
startingHealth);

        if (currentHealth > 0)
        {
            //igrac ozljeda
            anim.SetTrigger("hurt");
            StartCoroutine(Invunerability());
        }
        else
        {
            //ako igrac umre
            if (gameObject.CompareTag("Player"))
            {
```

```
                if (!dead)
                {
                    anim.SetTrigger("die");

                    //restarta level

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

                    //Deathcounter +1
                    DeathCounter.deathValue += 1;

                    dead = true;
                }
            }

            //ako neprijatelj umre
            for (int i = 0; i < tags.Length; i++)
            {
               if(gameObject.CompareTag(tags[i])) {

                    if (!dead)
                    {
                        anim.SetTrigger("die");

                        //deaktiviramo sve skripte za kretanje koje
su u komponentama
                        foreach    (Behaviour    component    in
components)
                            component.enabled = false;

                        dead = true;
                    }
```

```
                }
            }
        }
    }
```

Kod 28. Zadavanje ozljeda

Izvor: autor

Ozljeđivanje protivnika ili obrnuto vrši se funkcijom TakeDamage koja vraća vrijednost ozljede (engl. damage) koju igrač ili protivnik nanosi.

```
    private IEnumerator Invunerability()
    {
        invulnerable = true;
        Physics2D.IgnoreLayerCollision(10, 11, true);
        for (int i = 0; i < numberOfFlashes; i++)
        {
            spriteRend.color = new Color(1, 0, 0,0.5f);
            yield return new
WaitForSeconds(iFramesDuration/(numberOfFlashes*2));
            spriteRend.color = Color.white;
            yield return new WaitForSeconds(iFramesDuration /
(numberOfFlashes * 2));

        }
        //invurnebaility duration
        Physics2D.IgnoreLayerCollision(10, 11, false);
        invulnerable = false;
    }
```

Kod 29. Korutine za blještanje

Izvor:autor

Ako glavnom liku ili neprijatelju ponestane života, varijabla currentHealth padne na 0; izvršava se resetiranje scene pomoću SceneManager alata i koristi se DeathCounter skripta za zapisivanje broja umiranja. Ako neprijatelj umre, izvršava se prilično identična radnja osim što

nema resetiranja scene, brojenja umiranja i ubačena je for petlja koja pregledava listu oznaka neprijatelja te određuje koji neprijatelj je eliminiran.

10.1. Dodavanje života

U igri postoje srca koja se mogu skupljati kako bi se vratili izgubljeni životi. Da bi to funkcioniralo, varijabli currentHealth je dodan po jedan život za svako prikupljeno srce pa je dodana nova funkcija AddHealth.

```
    public void AddHealth(float _value)
    {
        currentHealth = Mathf.Clamp(currentHealth + _value, 0,
startingHealth);

    }
```

Kod 30. Funkcija za dodavanje života

Izvor: autor

Dodana je mala skripta koja dodaje srca igraču ukoliko dođe do kontakta između collidera igrača i srca za skupljanje. Svakim kontaktom se u varijablu currentHealth u Health skripti dodaje po vrijednost 1.

```
public class HealthCollect : MonoBehaviour
{
    [SerializeField] private float healthValue;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {

collision.GetComponent<Health>().AddHealth(healthValue);
            gameObject.SetActive(false);
```

```
            }
        }
    }
```

Kod 31. Skripta za dodirivanje igrača i srca

Izvor: autor

Funkcija Deactivate omogućuje nestajanje neprijateljskog objekta.

```
    private void Deactivate()
    {
        gameObject.SetActive(false);
    }
```

Kod 32. Deaktivacija objekta neprijatelja

Izvor: autor

10.2. Healthbar

Prikaz života (engl. Healthbar) prikazuje trenutno stanje života tijekom igre. Smješten je u gornjem lijevom kutu u obliku niza srdaca. To je zapravo child objekt objekta UICanvas te u sebi sadrži objekte HealthbarTotal i HealthbarCurrent. HealthbarTotal je konstantno prikazan kao maksimalan broj srca koja je moguće imati, pa je boja tog objekta nešto tamnija od HealthbarCurrent. Trenutna vrijednost srca ima punu crvenu boju radi lakšeg raspoznavanja. Healthbar objekt sadrži jednostavnu skriptu Healthbar koja sadrži samo tri varijable: playerHealth koja obavještava o vrijednosti života čitajući podatke iz Health skripte, totalhealthBar varijabla koja obavještava kolika je maksimalna vrijednost života i currenthealthBar varijabla koja obavještava o trenutnoj vrijednosti života.

```
public class Healthbar : MonoBehaviour
{
    [SerializeField] private Health playerHealth;
    [SerializeField] private Image totalhealthBar;
    [SerializeField] private Image currenthealthBar;
```

```
    void Start()
    {
        totalhealthBar.fillAmount = playerHealth.currentHealth /
10;
    }

    void Update()
    {
        currenthealthBar.fillAmount = playerHealth.currentHealth /
10;
    }
}
```

Kod 33. Prikaz trenutne i maksimalne vrijednosti života

Izvor: autor

10.3. Brojač smrti

Brojač smrti (eng. Deathcounter) služi za brojenje koliko puta je glavni lik bio eliminiran. Brojač se nalazi u gornjem desnom kutu u obliku natpisa DEATHS: i broj umiranja. To je child objekt objekta UICanvas te na sebi sadrži samo Text komponentu i DeathCounter skriptu. Za prikaz teksta koristi se besplatan font ThaleahFat koji je prethodno besplatno preuzet [12]. DeathCounter skripta služi samo za brojenje smrti glavnog igrača i podatke uzima iz skripte Health unutar TakeDamage funkcije.

```
public class DeathCounter : MonoBehaviour
{
    public static int deathValue = 0;
    Text death;

    void Start()
    {
        death = GetComponent<Text>();
    }
```

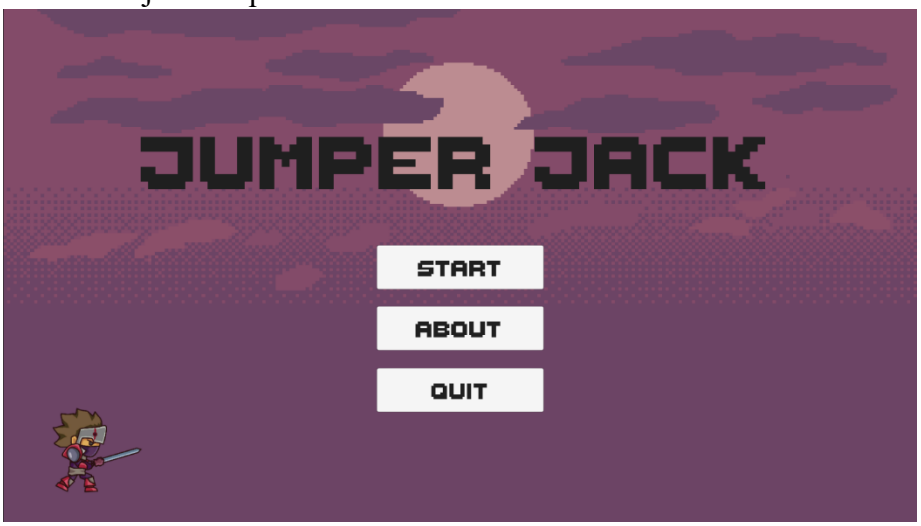
```
    void Update()
    {
        death.text = "Deaths: " + deathValue;
    }
}
```

Kod 34. Deathcouner

Izvor: autor

11. IZBORNICI

Velika većina video-igara sadrži neku vrstu izbornika. Izbornici služe kako bi se igrač lakše orijentirao kada otvori igru i pročitao upute koje će ga voditi kroz samu igru. U većini igara ne postoji samo jedan, već više izbornika, a glavni izbornik (engl. Main Menu) je uvijek onaj koji se otvara prije nego što igrač započne s igranjem. U ovom projektu postoje tri izbornika. To su: glavni izbornik (engl. Main Menu), izbornik kod pauziranja igre (engl. Pause Menu) i završni izbornik (engl. End Screen). Glavni izbornik se pokreće pokretanjem same igre, na njemu se nalaze ponuđene opcije kao što su Start za početak igranja, About za više informacija o samoj igri i upute o igranju i Quit koja omogućuje izlazak iz igre. Kod početnog i završnog izbornika se zbog ljepšeg vizualnog prikaza u donjem lijevom kutu nalazi slika glavnog lika odnosno Jacka. Svaka tipka mora imati određenu funkciju u skripti.



Slika 18. Glavni izbornik

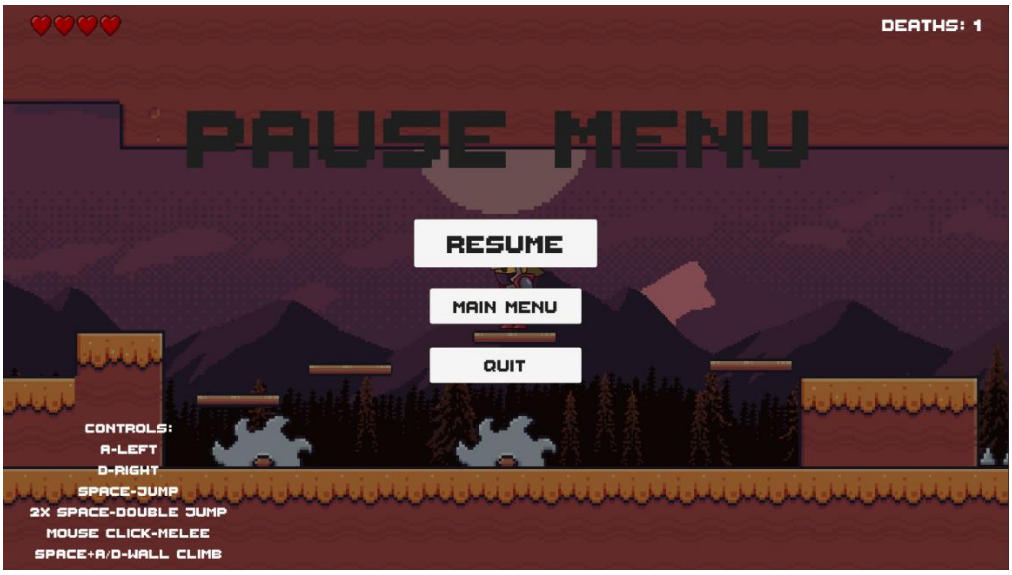
Izvor: autor

```
public class MainMenu : MonoBehaviour
{
    //Početak igre - START
    public void StartGame()
    {

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex  +
1);
    }

    public void Mainmenu()
    {
        SceneManager.LoadScene("Main Menu");
    }

    //Izlazak - QUIT
    public void EndGame()
    {
        Application.Quit();

        //u consoli nam javlja da li se igra iskljucuje
        Debug.Log("Igra se isključuje.");
    }

    //O igri - ABOUT
    public void Aboutt()
    {
        SceneManager.LoadScene("About");
    }
}
```

Kod 35. Glavni meni

Izvor: autor

Pauziranje igre vrši se pritiskom tipke Esc na tipkovnici i samim time se otvara izbornik koji nudi mogućnost nastavka igranja, izlazak u početni izbornik ili izlazak iz cijele igre. U donjem desnom kutu su navedene kontrole radi brzog informiranja ukoliko igrač zaboravi neku kontrolu. Otvaranjem izbornika pozadina se neće promijeniti kao kod početnog i završnog, već će se zatamniti trenutna scena. Pritiskom na opciju resume, scena se vraća u prvobitnu svjetlinu i igra se nastavlja.



Slika 19. Pause menu

Izvor: autor

```
public class PauseMenu : MonoBehaviour
{
    public static bool GamePaused = false;
    public GameObject pauseMenuUI;

    private void Update()
    {
        //Pritiskom Esc pauziramo igru
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GamePaused)
            {
                Resume();
            }
```

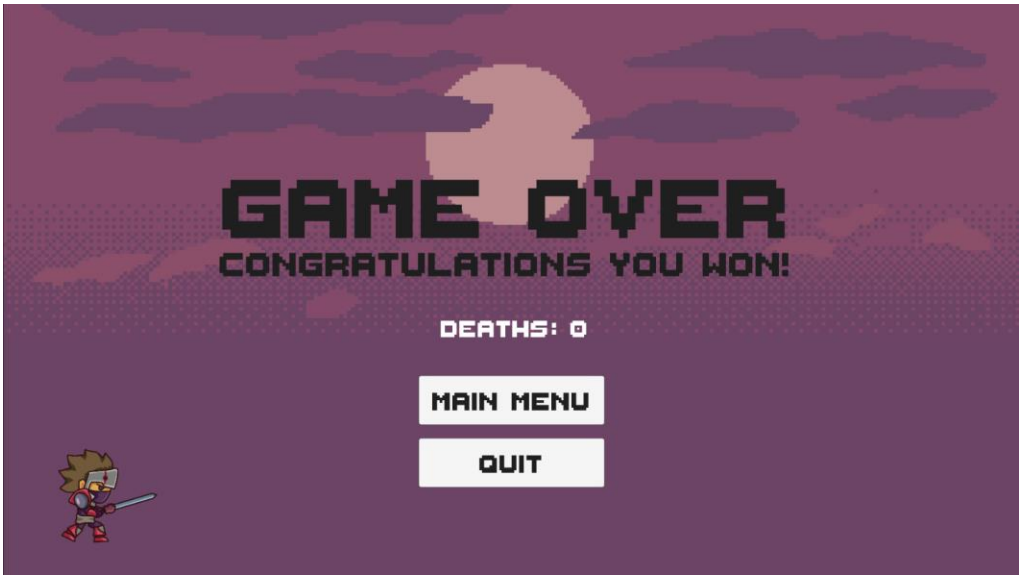
```
            else
            {
                Pause();
            }
        }
    }

    //Nastavak - RESUME
    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GamePaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GamePaused = true;
    }

    //Glavni meni - MAIN MENU
    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Main Menu");
    }

    //Izlazak - QUIT
    public void QuitGame()
    {
        Debug.Log("Igrica se iskljucuje");
        Application.Quit();
    }
```

```
}
```

Kod 36. Pause menu

Izvor: autor

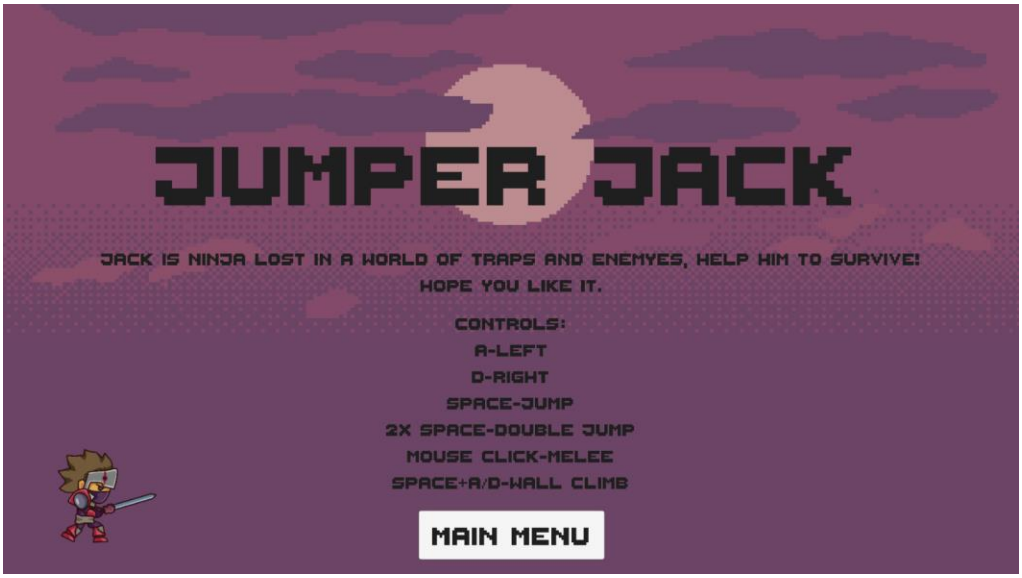
Završni izbornik nudi samo osnovne opcije, povratak u glavni izbornik i izlazak iz igre. Na završnom izborniku (koji je ujedno i završna scena) piše poruka kojom se čestita igraču na igranju i pobjedi, a iznad toga piše da je igra završena (engl. Game Over). Također je vidljiv brojač smrti koji pokazuje konačan broj pokušaja, odnosno eliminiranja glavnog igrača.



Slika 20. Završna scena

Izvor: autor

Scena About kojoj se pristupa preko glavnog izbornika sadrži informacije o igri i popis kontrola za upravljanje glavnim igračem, isto kao i kod pause izbornika u donjem desnom kutu. U glavni izbornik možemo se vratiti tipkom Main Menu.



Slika 21. O igri

Izvor: autor

12. RAZINE

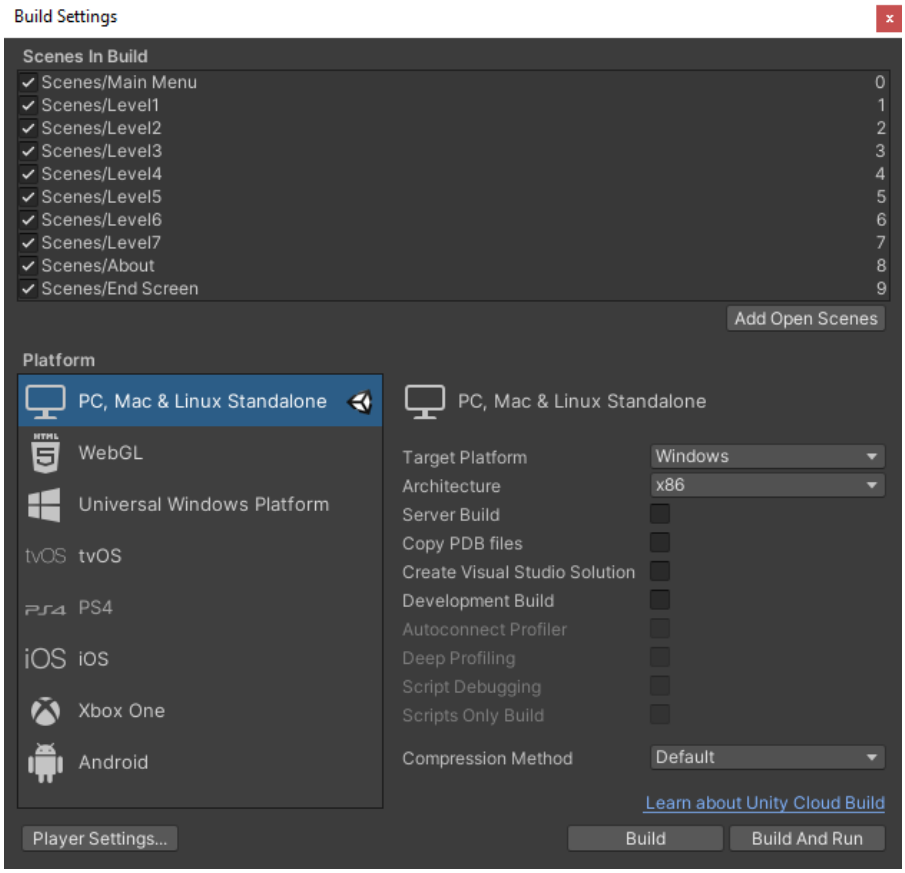
Kako bi igrač uspješno napredovao do sljedeće razine, mora se dodirnuti sa zastavicom koja se nalazi na kraju svake razine. Zastavica je objekt s box colliderom koji glavni lik mora dotaknuti svojim kako bi se učitala sljedeća scena. Učitavanje scena omogućeno je dodavanjem skripte.

```
public class LevelControl : MonoBehaviour
{
    public string level;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            //ucitava level
            SceneManager.LoadScene(level);
        }
    }
}
```

Kod 37. Level control

Izvor: autor

U polju Level u inspectoru se nalazi naziv scene koja će se učitati prilikom pokretanja skripte pa su tako na svakoj razini upisani određeni nazivi sljedećih razina: Level2, Level3 i tako dalje. U build settings prozoru moraju obavezno biti označene sve scene koje se koriste jer se inače neće uzimati u obzir. U build settings prozoru se nalazi popis svih scena i platformi za koje je moguće kreirati projekt.



Slika 22. Build settings

Izvor: autor

13. POZADINSKA GLAZBA

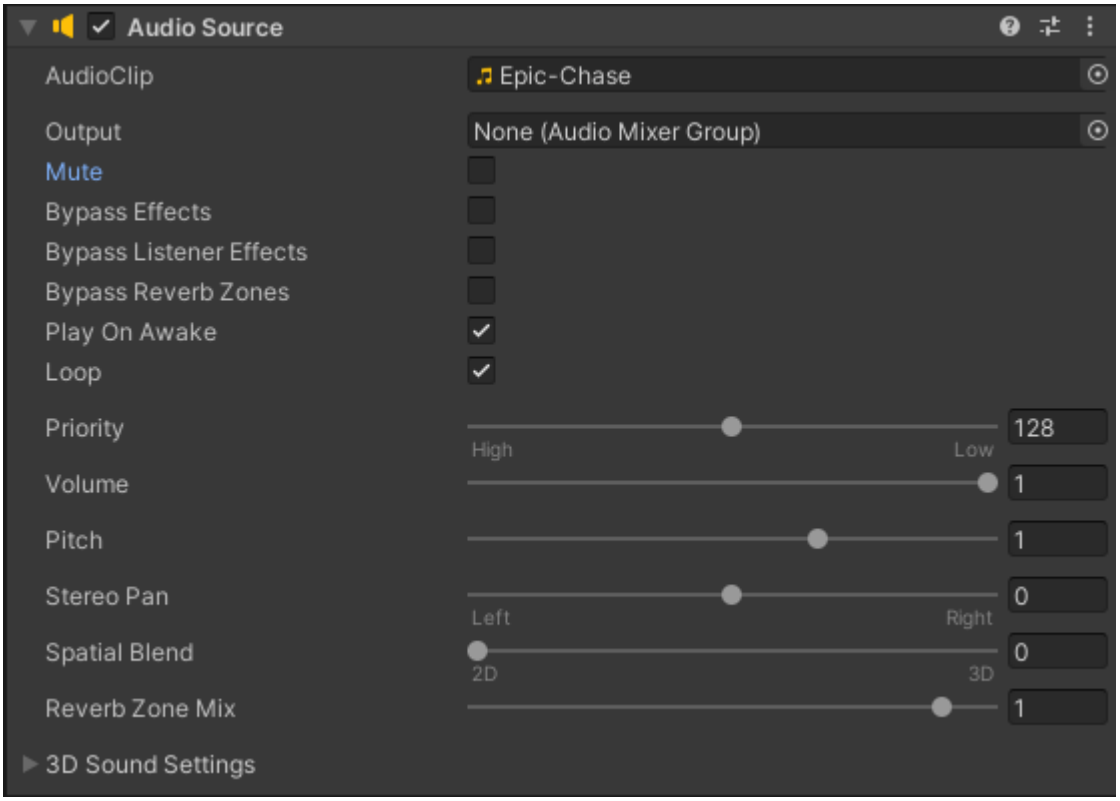
Pozadinska glazba daje video-igri realističniji doživljaj igranja. Moguće je kreirati vlastitu pozadinsku glazbu i zvučne efekte te ih dodavati kao assete u projekt, no u ovom radu koristi se pozadinska glazba naziva Epic-Chase koja je preuzeta s web-stranice za besplatno preuzimanje zvučnog sadržaja Chosic [13]. Emitiranje glazbe omogućava komponenta Audio Source. U Audio Source komponenti nalazi se polje Audio Clip u koje je dodana zvučna datoteka. Dodana je skripta DontDestroy koja služi kako se pozadinska glazba ne bi ugasila prijelazom na novu razinu ili ponavljala kada igrač umre. Razlog tom problemu je to što se objekt Music nalazi samo u sceni u koju je prvo bio dodan. Objektu Music dodana je oznaka Music tako da ga skripta može pronaći.

```
public class DontDestroy : MonoBehaviour
{
    private void Awake()
    {
        GameObject[]                    musicObj                    =
GameObject.FindGameObjectsWithTag("Music");

        if(musicObj.Length > 1)
        {
            Destroy(this.gameObject);
        }
        DontDestroyOnLoad(this.gameObject);
    }
}
```

Kod 38. Pozadinska glazba

Izvor: autor

Audio Source komponenta nudi mogućnost raznih podešavanja zvučnih efekata.

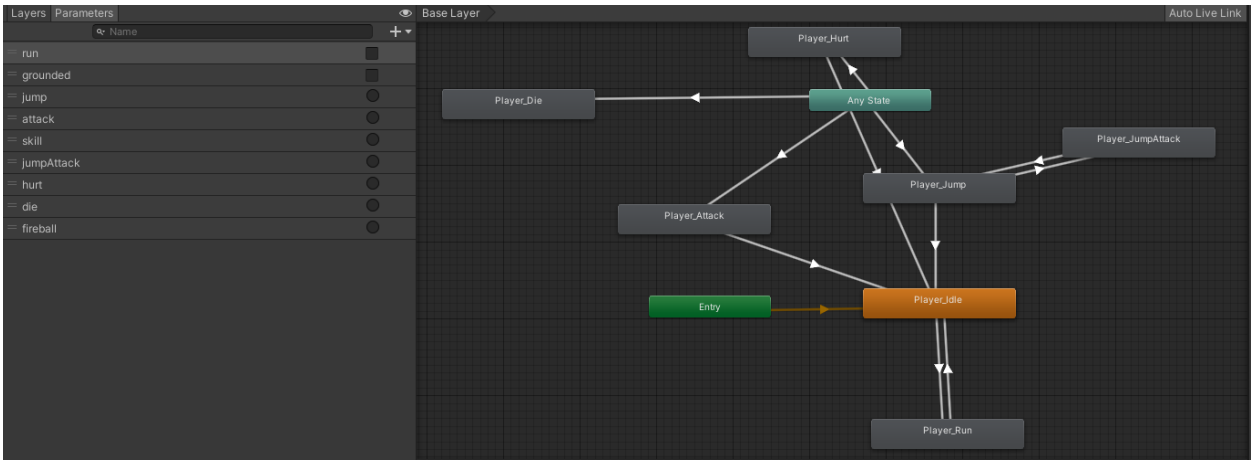


Slika 23. Audio Source

Izvor: autor

14. ANIMACIJE

Animacije su bitan element svake video-igre kako bi se vizualno dočaralo ponašanje objekata u igri. U ovom radu koristi se više uzastopnih slika koje zajedno čine neku animaciju. Svaki živi objekt u igri ima svoju animaciju za svaki pokret, pa čak i stajanje na mjestu. Bez animacija bi igraču teško bilo predočiti što se zapravo događa i zašto određeni objekt radi određenu radnju.



Slika 24. Animator prozor glavnog igrača

Izvor: autor

Prozor Animator pomaže kod spajanja raznih animacija da bi Unity znao kada treba koju animaciju upotrijebiti. Spajanje se vrši tranzicijama (eng. transitions). Kao primjer uzet je glavni igrač s njegovim animator controllerom. Kao što je vidljivo na slici iznad, postoji mnogo animacija, a to su: umiranje (Player_Die), ozljeda (Player_Hurt), napad (Player_Attack), skok (Player_Jump), trčanje (Player_Run) i napad u zraku (Player_JumpAttack). Animacija Player_Idle je početna animacija te se ona izvodi kada igrač stoji na mjestu. Entry označava početak animacija te je spojen direktno s animacijom stajanja. Any State omogućuje da se u bilo kojem trenu izvede neka animacija, na primjer ozljeda i umiranje. U prozoru Parameters na lijevoj strani vide se parametri koji su dodani u tranzicije. Njih označuju bijele strelice između animacija. Kada se zadovolji određeni parametar, ta tranzicija se uključi i izvede se animacija; kada je neka animacija gotova, vraća se u idle stanje i čeka se daljnja naredba. Klikom na željenu tranziciju otvaraju se njezine postavke te se mogu dodavati uvjeti odnosno parametri da se ta tranzicija izvede. Parametri se aktiviraju u programskom kodu kada su potrebni.

```
   //animacija napada
if (!isGrounded())
{
        anim.SetTrigger("jumpAttack");
}
else
{
        anim.SetTrigger("attack");
   }
```

Kod 39. Primjer animacije za napad

Izvor: autor

Kada igrač nije prizemljen i izvršava se napad, parametar jumpAttack se uključuje i izvršava se animacija Player_JumpAttack. Kada je igrač prizemljen i izvršava se kod za napad, aktivira se parametar attack i izvršava se animacija Player_Attack.

Kod neprijatelja je dodan Add event koji se dodaje u određeni dio animacije gdje se označava kada se nešto dogodilo. U ovom primjeru se na 0:11 sekundi događa napad na glavnog igrača, točnije funkcija Damage Player.

15. ZAKLJUČAK

Gaming industrija nastavlja s razvojem velikom brzinom iz dana u dan. Video-igre nisu nešto bez čega ljudi ne bi mogli živjeti, ali nudi veliko zadovoljstvo i opuštanje od svakodnevnih briga i problema. Sam proces izrade video-igara je prilično dug i zahtjevan, ali uz puno volje može se napraviti vrlo zadovoljavajući rezultat.

Svaka osoba koja želi postati developer odnosno programer video-igara, mora proći uglavnom kroz sve segmente koji su opisani u ovom završnom radu. Uz pomoć Unitya i Asset Storea na jednostavan se način svaka osoba može iskušati u ovom području, nebitno preferira li 2D ili 3D svjetove. Obično na jednom projektu radi istovremeno više programera, dizajnera i ostalog osoblja jer je moguća podjela na različite dijelove projekta; nekome više odgovara programiranje dok nekome više odgovara dizajniranje likova ili nešto drugo. Ovakav način izrade projekta je vrlo zanimljiv jer nudi mogućnost više grana izrade, a ne samo programiranja ili nečega drugog.

Način na koji je u ovom završnom radu izrađen projekt može proširiti znanje ne samo na području gaming industrije, već programerskog svijeta općenito.

16. POPIS LITERATURE

[1] Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There. https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/ (29.1.2021.)

[2] Adam Sinicki, What is Unity? Everything you need to know. https://www.androidauthority.com/what-is-unity-1131558/ (29.1.2021.)

[3] Introduction to Unity 3D https://www.studytonight.com/3d-game-engineering-with-unity/introductionto-unity (29.1.2021.)

[4] Unity Hub. https://docs.unity3d.com/Manual/GettingStartedUnityHub.html (14.2.2021.)

[5] Quick guide to the Unity Asset Store. https://unity3d.com/quick-guide-to-unity-asset-store (19.2.2021.)

[6] Download Visual Studio Tools. https://visualstudio.microsoft.com/downloads/ (4.6.2021.)

[7] Visual Studio. https://www.incredibuild.com/integrations/visual-studio (4.6.2021.)

[8] Michael Klappenbach, What is a Platform Game?. https://www.lifewire.com/what-is-a-platform-game-812371 (17.7.2021.)

[9] 2D computer graphics. https://graphics.fandom.com/wiki/2D_computer_graphics (18.7.2021.)

[10] Unity - Creating Sprites. https://www.tutorialspoint.com/unity/unity_creating_sprites.htm (18.7.2021.)

[11] C# Inheritance. https://www.w3schools.com/cs/cs_inheritance.php (2.8.2021.)

[12] Thaleah Fat by Tiny Worlds. https://tinyworlds.itch.io/free-pixel-font-thaleah (15.8.2021.)

[13] Chosic https://www.chosic.com/ (6.11.2021.)

17. POPIS PROGRAMSKIH KODOVA

18. POPIS SLIKA