Razvoj platformerske videoigre "Platformania" korištenjem Unity razvojnog okruženja

Kos, Paolo

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Paolo Kos

Platformania

Završni rad

Mentor: izv. prof. dr. sc. Marina Ivašić-Kos

Rijeka, rujan 2020.

Sadržaj

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 17. veljače 2020.

Zadatak za završni rad

Pristupnik: Paolo Kos

Naziv završnog rada: **Razvoj platformerske videoigre "Platformania" korištenjem Unity razvojnog okruženja**

Naziv završnog rada na eng. jeziku: **Development of platformer video game "Platformania" using Unity Engine**

Sadržaj zadatka: Proučiti i opisati različite vrste video igara s naglaskom na 2D platformere. Proučiti odgovarajuće alate za dizajn i razvoj video igara te opisati uporabu Unity razvojnog okruženja za izradu video igre.

Izraditi i opisati 2D platformer video igru te razvoj priče u igri. Opisati dizajni igre, animaciju likova, kretnje likova, proces generiranja mapa kao i sadržaja na mapi. Objasniti implementaciju korištenja oružja, života likova te računanje rezultata i praćenje napretka u igri kao i implementaciju zvukova u igri.

Ukoliko su za dizajn igre, animaciju likova, izradu glazbe i zvukova ili nešto drugo korištene specijalizirane aplikacije opisat postupak integracije s Unity-jem.

Mentor

Izv. prof. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 17.2.2020.

Paolo Kos

(potpis pristupnika)

Sažetak

U ovom radu opisana je izrada 2D platformerske video igre „Platformania“. Igra, osim samog igrača, sadrži protivnike i prepreke koji otežavaju dolazak do cilja. Igrač sakuplja novčiće te tako dobiva bodove. Također može nanijeti štetu protivniku, kao i protivnik igraču. Cilj igre je preći sve prepreke te doći do finalnog portala sa čim većim brojem novčića. Igra se sastoji od više različitih scena, kao što su main menu, gameplay scena, finalna scena itd. te je svaka od tih scena detaljno opisana u ovome radu.

Ključne riječi: Unity, platformer, igrač, protivnik, prepreke, skripta, C#, animacija

1. Uvod

Igre su sastavni dio života svih generacija, počevši od igara u prirodi, preko društvenih igara pa sve do video igara koje su nastale po uzoru na ove prethodne. Prva video igra nastaje 1972. godine pod nazivom „Pong“ [1]. Bila je to arkadna video igra za dva igrača koju je razvila tvrtka Atari, a nastala po uzoru na ping pong. Ova igra je pokrenula razvoj industrije računalnih igara u čitavom svijetu, a razvoj traje i danas. Gaming industrija danas broji preko 200 milijuna igrača, tzv. „gamera“, a zarada se procjenjuje na više od 100 milijardi dolara [2]. Video igre se najviše igraju na području Azije, točnije Kine, a slijede ju SAD, Njemačka i Velika Britanija.

Platformerske igre, kao podžanr akcijskih igara, jedne su od najrasprostranjenijih žanrova video igara. Procjenjuje se da je, kada je bio na vrhuncu, jedna trećina video igara bilo platformerskog žanra, sve do 2006. godine kada počinju prevladavati ostali žanrovi [3]. 2010. godine platformerske video igre ponovno doživljavaju rast u popularnosti, ali u obliku mobilnih „endless runnera“. Odlikuje ih jednostavnost levela i likova kao i mehanizma. Najčešći oblici kretnje jesu hodanje, trčanje, skakanje i penjanje što nije zahtjevno za programere, ali je vrlo učinkovito. Cilj igara je, kao što samo ime žanra kaže, preko platformi, doći do cilja, pritom izbjegavajući razne prepreke i neprijatelje. Nerijetko, platformerske video igre imaju ugrađene razne zagonetke koje igrače potiču na razmišljanje, kao i edukativni sadržaj najčešće namijenjen mlađim uzrastima.

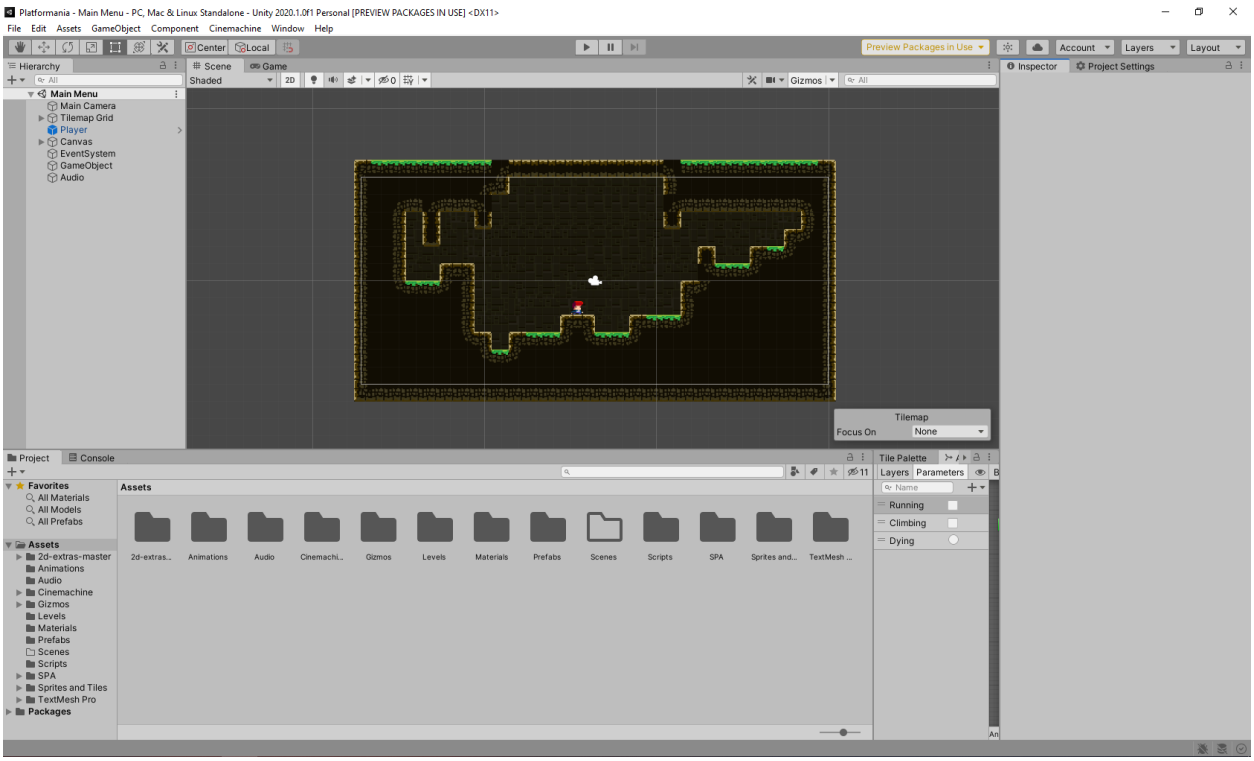
U ovom završnom radu prikazan je detaljan proces izrade 2D platformerske video igre „Platformania“ koristeći Unity razvojno sučelje. Igra se sastoji od više mapa i različitih neprijatelja, a cilj je doći do kraja igre sa najvećim mogućim brojem bodova. Igra je namijenjena za jednog igrača, a svi elementi igre, kao što su zvukovi i grafika preuzeti su s različitih izvora. Ideja za samu izradu ove video igre nastala je iz mega popularne igre „Super Mario“. Kao veliki ljubitelj video igara oduvijek sam želio izraditi vlastitu video igru, te proširiti znanje pomoću korištenja raznih alata za izradu video igara kao što je Unity.

2. Unity

Unity je višeplatformski softver za razvoj 2D i 3D video igara i simulacija [4]. Izradila i objavila ga je kompanija Unity Technologies 2005. godine na Worldwide Developers koferenciji tvrtke Apple Inc za platformu OS X. Danas Unity ima podršku za 27 platformi, a neke od njih su Windows, macOS, Linux, Android i sl. Također se, osim u gaming industriji, koristi i u raznim drugim industrijama, kao što su filmska, automobilska itd. Koristi programski jezik C# te objektno orijentirano programiranje za razvoj koda. Iako besplatan, pomoću svojeg Assets storea, Unity ima mogućnost kupovine raznih modela za video igru, kako korisnik ne bi morao samostalno izrađivati modele. Unity verzija koju sam koristio za izradu igre je 2020.1.2f1.

U nastavku su prikazane osnovne komponente Unity sučelja.

2.1 Unity sučelje

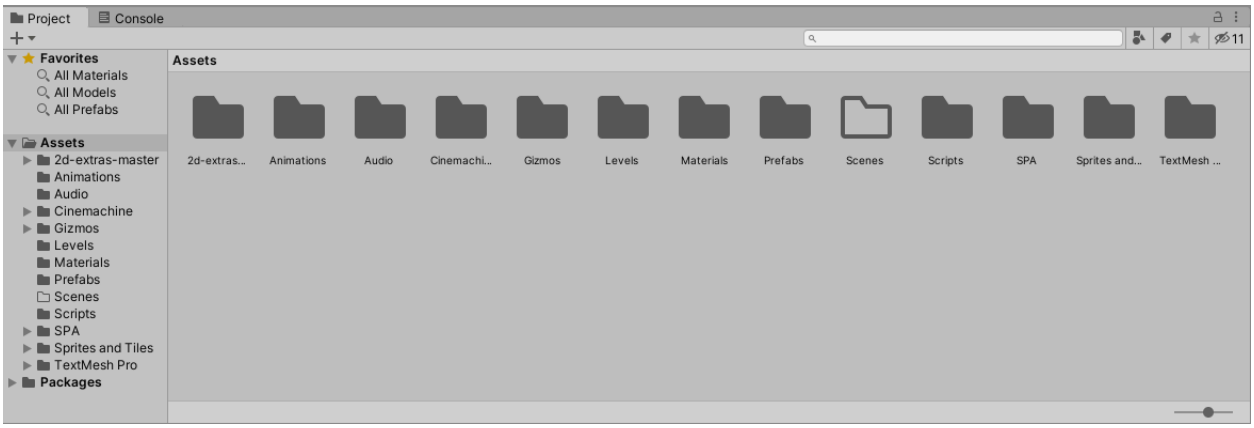


Slika 1: Unity Sučelje

Prilikom pokretanja programa, nalazimo se u glavnom prozoru odnosno sučelju prikazanom na slici 1. Prozor se sastoji od više dijelova, a to su:

- Project
- Console
- Hierarchy
- Scene
- Game
- Inspector
- Project settings

2.1.1 Project



Slika 2: Project prozor

Na slici 2 je prikazan prozor *Project* [5] koji sadrži folder *Assets* u koji se pohranjuju sve datoteke važne za izradu video igre. Folder *Assets* sadrži skripte, assetse, audio datoteke te ostale datoteke koje korisnik želi koristiti u projektu. Važno je sve datoteke rasporediti u zasebne, sortirane mape kako bi navigacija kroz iste bila brža i kako bi uvijek znali u kojoj se mapi nalaze potrebne datoteke. Assetsi korišteni u izradi video igre „Platformania“ nazivaju se „Super Platformer Assets“ te sadrže sve osnovne komponente za izradu 2D platformerske video igre (platforme, igrača, protivnike i sl.).

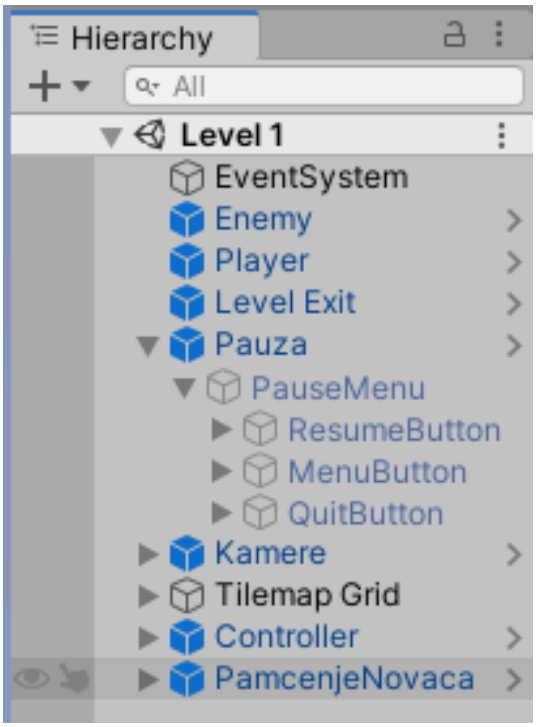
2.1.2 Console

Slika 3: Console prozor

Prozor *Console* (slika 3) [6] obavještava korisnika o problemima u projektu. Prikazuje razne pogreške, upozorenja i ostale poruke generirane u Unityu. Također, korisnik u konzoli može prikazati vlastite poruke koristeći Debug.Log funkciju što može biti korisno za otkrivanje problema u kodu ili za izradu interaktivne tekstualne video igre.

2.1.3 Hierarchy

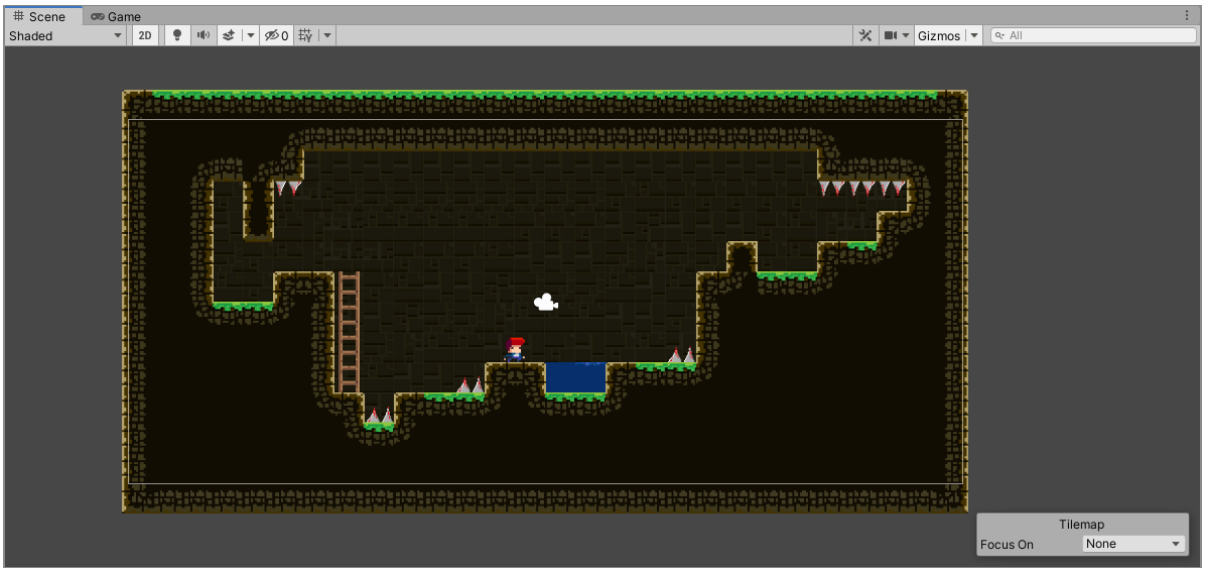
Na slici 4 možemo vidjeti *Hierarchy* [7] prozor koji prikazuje hijerarhiju objekata na trenutnoj sceni. Svaki objekt koji se dodaje ili obriše iz projekta, također se dodaje ili briše iz same hijerarhije. Objekti se pokreću redom od vrha prema dnu, što može biti korisno i jako bitno kod promjene scena u igri. Objekte možemo premještati po želji, a također ih možemo grupirati. Ukoliko grupiramo objekte dobijemo objekt „roditelj“ (eng. *parent*) i „dijete“ (eng. *child*). Korištenjem „parenting“ veze, korisnik može ugasiti samo „roditelja“ te tako automatski ugasiti i „djecu“ što može biti vrlo efikasno kada se koristi meni za pauzu. U ovom radu je upravo za pauzu korištena takva veza. Ukoliko korisnik pritisne tipku escape, pokrenut će roditelja (*PauseMenu)* te će automatski pokrenuti i gumbove kojima je *PauseMenu* roditelj, a to su *ResumeButton*, *MenuButton* i *QuitButton*. Ukoliko igrač ponovno klikne tipku Escape ugasit će roditelja, odnosno *PauseMenu GameObject*, a samim time će deaktivirati i ostale gumbove kojima je *PauseMenu* roditelj.



Slika 4: Hierarchy prozor

2.1.4 Scene

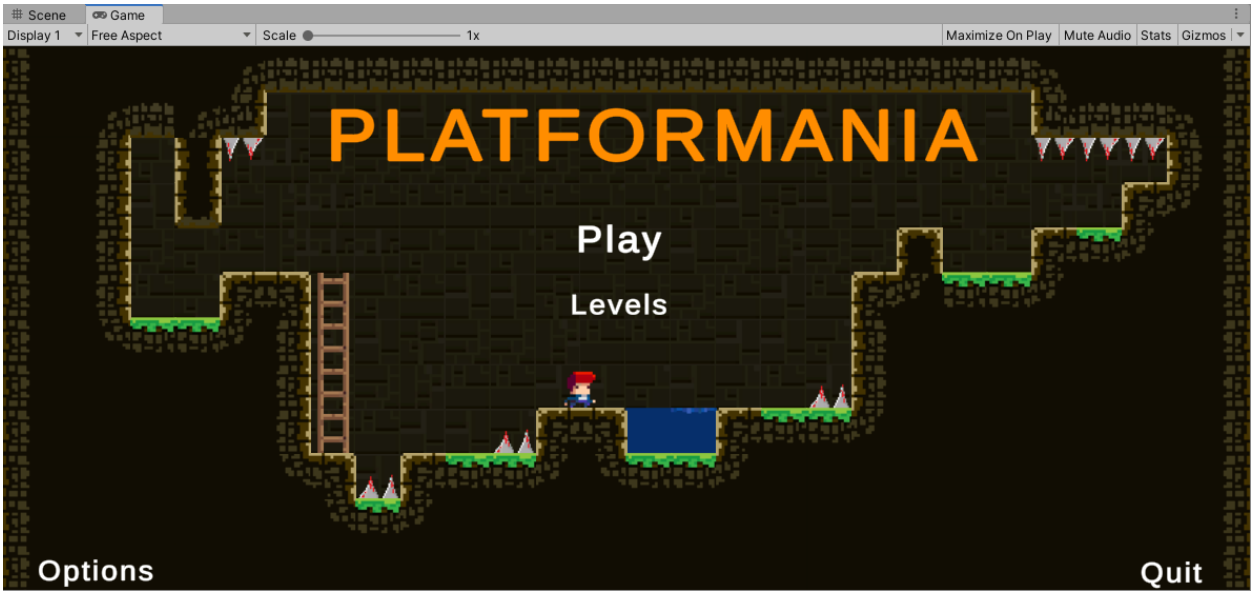
Na slici 5 vidimo *Scene* [8] prozor. On prikazuje interaktivni pogled na svijet koji korisnik kreira. Na sceni možemo selektirati, pozicionirati i mijenjati veličinu objekata, pomicati kameru i rasvjetu te dizajnirati izgled same mape. Unutar scene vidimo kako će igra izgledati kada ju pokrenemo te iz kojeg kuta ćemo gledati odnosno igrati igru.



Slika 5: Scene prozor

2.1.5 Game

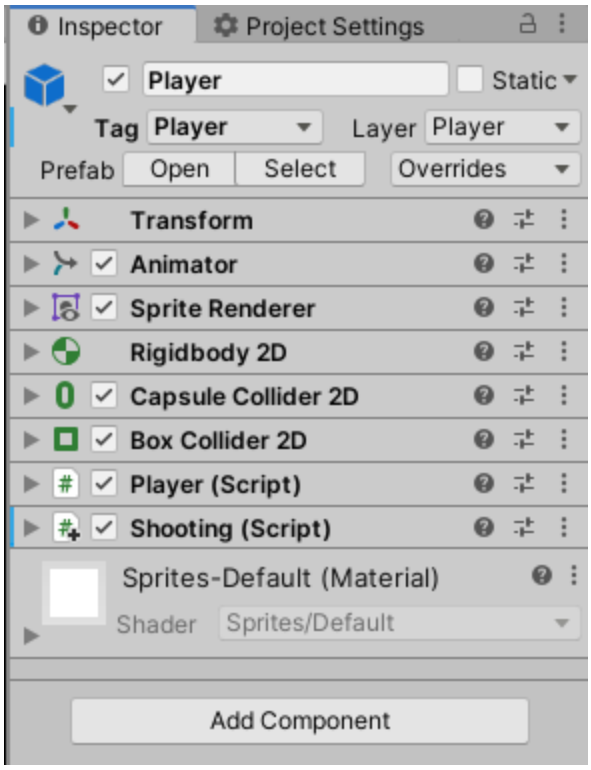
Game prozor [9] prikazan na slici 6 prikazuje finalni proizvod projekta, odnosno, prikazuje igru sa gledišta samog igrača kroz kameru. U game prozoru možemo pokrenuti bilo koju scenu u projektu te tako osigurati da igra vizualno izgleda dobro igraču. Kao i u Scene prozoru, možemo razmještati objekte, ali ovoga puta u realnom vremenu te tako vidimo kako će se odvijati određene situacije u igri. Game view služi kao testiranje produkta kao i pogled na igru u različitim rezolucijama.



Slika 6: Game prozor

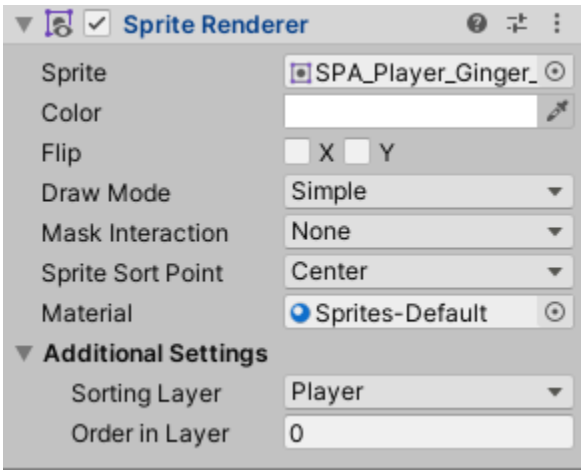
2.1.6 Inspector

Klikom na određeni objekt, korisnik će u prozoru *Inspector* (slika 7) [10] dobiti uvid na sve postavke odabranog objekta. Unutar *inspectora* korisnik može manualno mijenjati poziciju, rotaciju i veličinu objekata, dodavati animacije i skripte, kao i zvukove i ostale grafičke elemente. *Inspector* prikazuje sve komponente povezane sa objektom i njihove postavke, te omogućuje izmjenu istih. Također, korisnik pomoću *inspectora* može dodavati gravitaciju objektima te određivati njihovo ponašanje s obzirom na okolinu (kolizija i sl.).



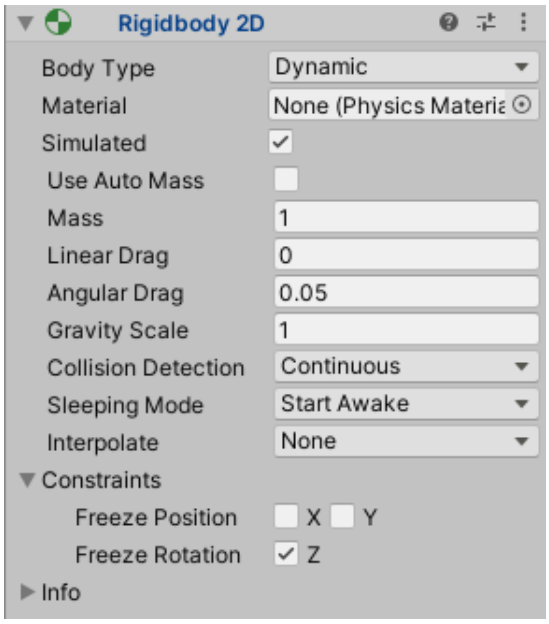
Slika 7: Prozor Inspector

Npr. klikom na objekt Player u Inspectoru (slika 7) dobijemo uvid u sve komponente vezane za taj objekt. Svaka od tih komponenti ima određenu funkcionalnost, npr. komponenta *Animator* se koristi kako bi glavni lik mogao imati animacije (za trčanje, skakanje i sl.). *Sprite Renderer* glavnog lika prikazuje na ekranu tako što komponenti dodamo sprite igrača u sekciju sprite (slika 8) [11].



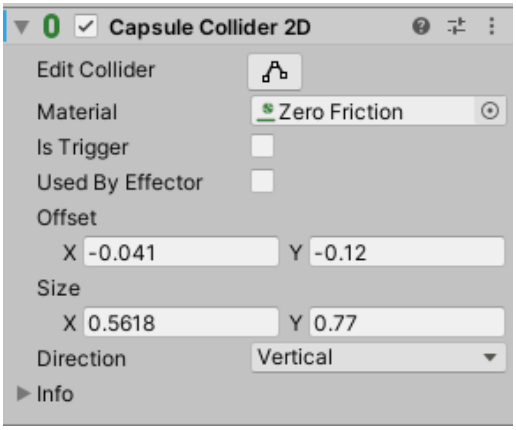
Slika 8: Sprite Renderer komponenta

Rigidbody 2D [12] prikazan na slici 9 daje objektu fizička svojstva, odnosno, omogućava objektu da reagira na dodire sa ostalim colliderima. Omogućuje kretnju po X i Y osima kao i rotaciju po Z osi.



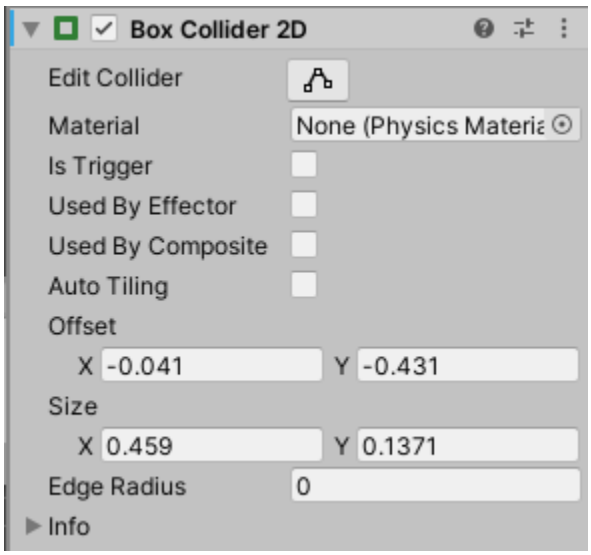
Slika 9: Rigidbody 2D komponenta

Capsule Collider 2D (slika 10) [13] je vrlo važna komponenta pri izradi igara koja objektima daje oblik i fizičku koliziju. U ovom je radu korištena kako bi igač imao koliziju sa okolinom. Npr. ukoliko igrač dodirne protivnika, zapravo njegov *Capsule Collider* dodirne collider protivnika te igrač primi štetu. Možemo mu mijenjati veličinu te manualno mu namještati poziciju klikom na gumb *edit collider*.



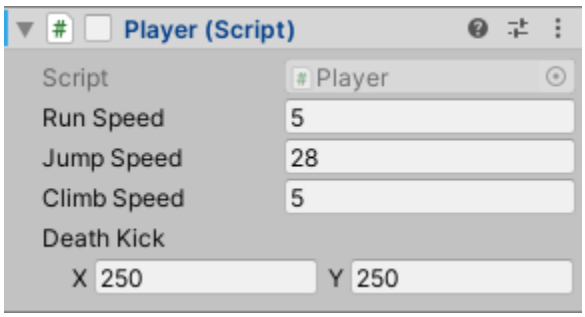
Slika 10: Capsule Collider 2D komponenta

Kao i *Capsule Collider*, *Box collider* (slika 11) [14] ima jednaka svojstva i funkciju. Najčešće se koristi kada želimo koliziju između lika i površine na kojoj se lik nalazi. Upravo je za to korišten i u ovom radu. Kasnije u skripti imamo kod koji će omogućiti skok ukoliko igrač, upravo svojim box colliderom dodiruje površinu.



Slika 11: Box Collider 2D komponenta

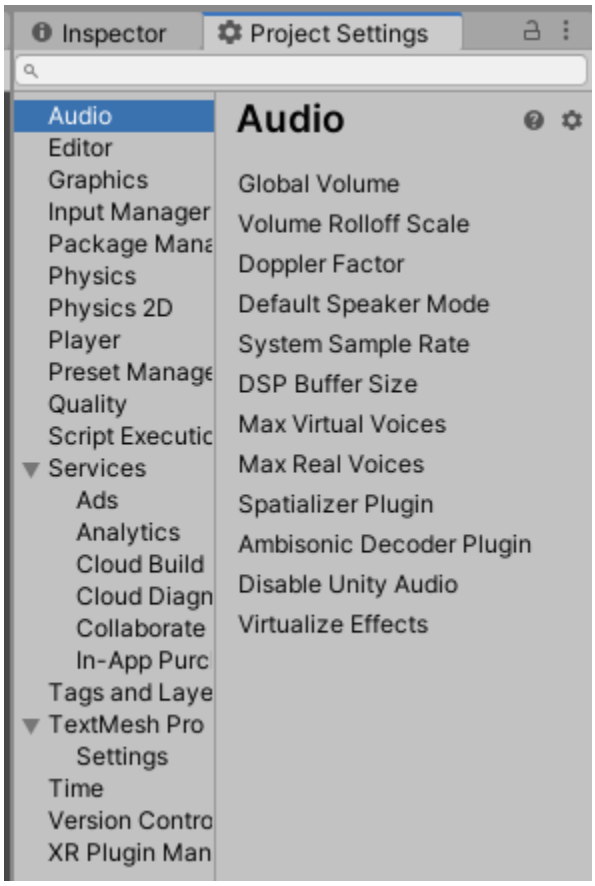
Posljednja komponenta koju sadrži objekt Player jest *Script* [15]. Skripte govore objektima kako se moraju ponašati sa okolinom. Kao što je prikazano na slici 12, skripta player koristi komponente Run Speed, Jump Speed, Climp Speed i Death Kick koje možemo manualno mijenjati unutar samog inspectora te tako, bez promjene samo koda unutar skirpte, vidjeti promjene unutar igre.



Slika 12: Script komponenta

2.1.7 Project settings

Project settings prozor (slika 13) [16] omogućuje manipulaciju postavkama čitavog projekta. Omogućuje izmjenu jačine zvukova u projektu, izmjenu načina upravljanja likovima u igri, kao i fizikom između dva objekta unutar igre (npr. koliziju između 2 objekta).



Slika 13: Project settings prozor

3. Razvoj igre

U nastavku je opisan razvoj igre Platformania koja je namijenjena za jednog igrača. Igra je zamišljena kao platformer unutar velike pećine te je cilj doći do portala kako bi izašao iz pećine. Glavni lik kao oružje ima luk i strijelu te može nanijeti štetu protivnicima, ali i protivnici njemu ukoliko ga dodirnu.

3.1 Ideja

Inpiracija za izradu ove igre bila je platformerska video igra „Super Mario“. Po uzoru na Super Mario i u ovoj igri je jednak cilj: -preko prepreka doći do kraja igre. Osim osnovnih elemenata platformerskih igara kao što su prelazak preko prepreka i izbjegavanje smrti, Platformania sadrži i ostale elemente koji nisu uobičajeni za ovaj tip igara. Osim samog prelaska preko prepreka, igrač može skupljati novčiće, ali i pucati na protivnike sa neograničenim brojem strijela. Igrač ima ukupno 5 života te svakom smrti gubi po život. Kada izgubi svih 5 života, ponovno se stvara na prvom levelu te mora započeti igru ispočetka. Ukoliko igrač na levelu pronađe srce, dobiva dodatni život. Svaki level je autentičan i posebno dizajniran, a na kraju svakoga se nalazi portal do kojeg igrač mora doći kako bi prešao na idući level. Putem, igrač sakuplja novčiće, te se na kraju ispisuje koliko je igrač novčića sakupio te koliko mu je preostalo života. Ukoliko igrač želi, može pokrenuti samo određeni level tj. ne mora igru započeti od prvog levela.

3.2 Scene

U nastavku su opisane scene korištene u izradi video igre te sve funkcionalnosti koje scene sadrže.

3.2.1 Main Menu scena

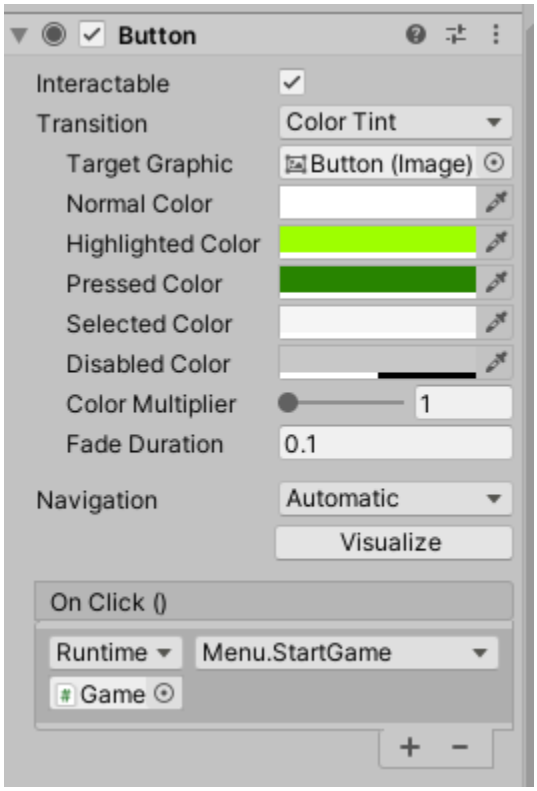
Pokretanjem igre prikazuje se glavni izbornik prikazan na slici 14. Ovdje igrač može započeti novu igru, odabrati level koji želi igrati, ući u postavke igre ili ugasiti igru. Osnovne funkcionalnosti scene realizirane su pomoću interaktivnih gumbova koji pokreću iduće scene.

Slika 14: Main Menu scena

Kako bi gumbovi funkcionirali, korištena je skripta Menu koja sadrži klasu Menu u kojoj se nalaze funkcije za promjenu scena kao što su StartGame() koja učitava prvi level igre, ExitGame() kojom se izlazi iz igre i Options() kojom se otvara prozor za definiranje postavki. Scene se dohvaćaju pomoću imena scena koje želimo pokrenuti kao što je „Level 1“ u funkciji SceneManager.LoadScene(„Level 1“) za učitavanje prve scene.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{
    public void StartGame()
    {
        SceneManager.LoadScene("Level 1");
    }

    public void ExitGame()
    {
        Application.Quit();
    }

    public void Options()
    {
        SceneManager.LoadScene("Options");
    }
```

Osim skripte, kako bi aktivirali scenu, potrebno je unutar Inspectora dodati On Click() akciju (slika 15) koja sadrži objekt „Game Object“ i u klasi Menu iz „Menu“ skripte pokreće funkciju StartGame() koja će se pokrenuti kada kliknemo na gumb. Funkciju koja će se pokrenuti kada kliknemo na gumb biramo u padajućem izborniku.



Slika 15: On Click()

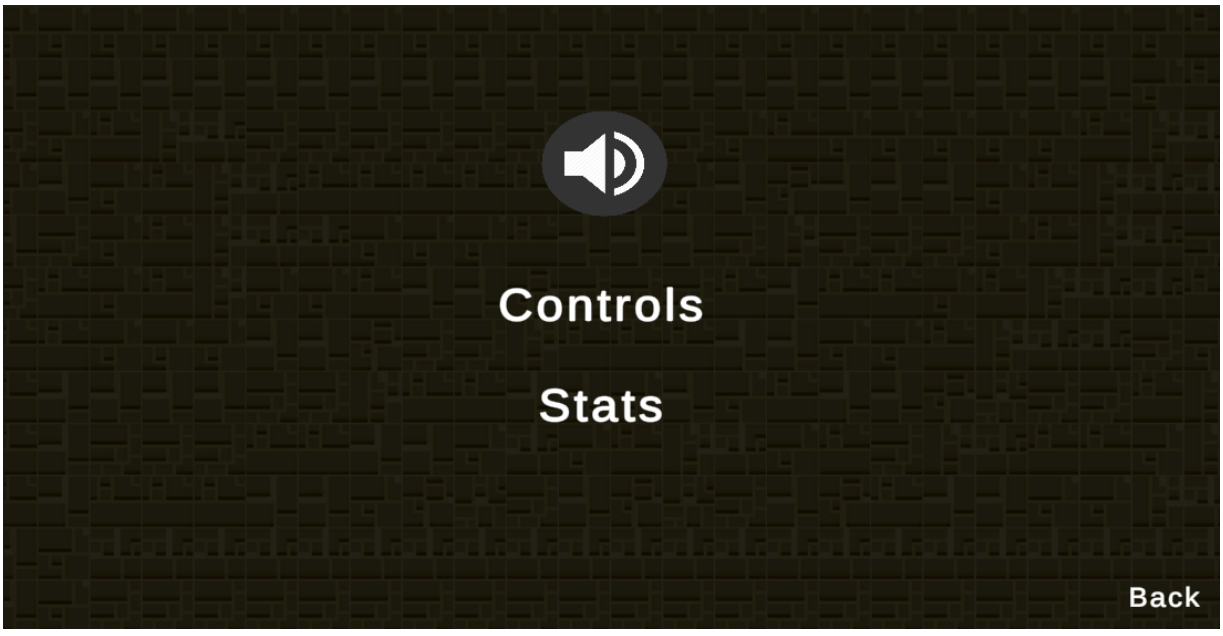
3.2.2 Level Selection scena

Na slici 16 vidimo Level Selection scenu implementiranu na jednaki način kao Main Menu scena. Pomoću gumbova pokreću se određene scene, npr. klikom na gumb sa oznakom Level 2, pokrenut će se funkcija unutar skripte Menu koji će promijeniti scenu na scenu s imenom „Level 2“.

Slika 16: Level Selection scena

3.2.3 Options scena

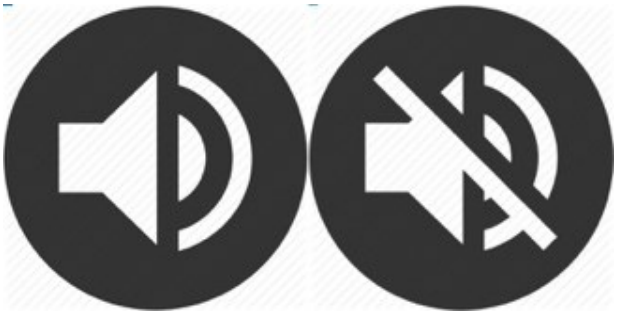
Options scena prikazana je na slici 17. Na njoj se nalaze 3 gumba za prelazak na iduće scene („Controls“,“Stats“ i „Back“), ali i gumb za gašenje i pokretanje glazbe u igri.



Slika 17: Options scena

Za upravljanje gumba za glazbu zaslužna je Skripta Sound koja sadrži klasu Sound unutar koje se nalaze funkcije za pokretanje i gašenje glazbe. Ukoliko igrač želi ugasiti glazbu, klikom na gumb, pokrenut će se funkcija UpdateIkonaIVolumen koja će AudioListener.volume postaviti na 0, odnosno, ugasiti će se zvukovi igre, te će se slika gumba promjeniti u odgovarajuću sliku gumba za ugašenu glazbu (slika 18). Ukoliko igrač želi ponovno pokrenuti glazbu, ponovnim klikom na gumb, AudioListener.volume će se postaviti na 1 te će slika gumba biti odgovarajuća za pokrenutu glazbu (slika 18).

```
void UpdateIkonaIVolumen()
    {
        if (PlayerPrefs.GetInt("Muted",0) == 0)
        {
            AudioListener.volume = 1;
            musicToggleButton.GetComponent<Image>().sprite = musicOnSprite;
        }
        else
        {
            AudioListener.volume = 0;
            musicToggleButton.GetComponent<Image>().sprite = musicOffSprite;
        }
    }
```



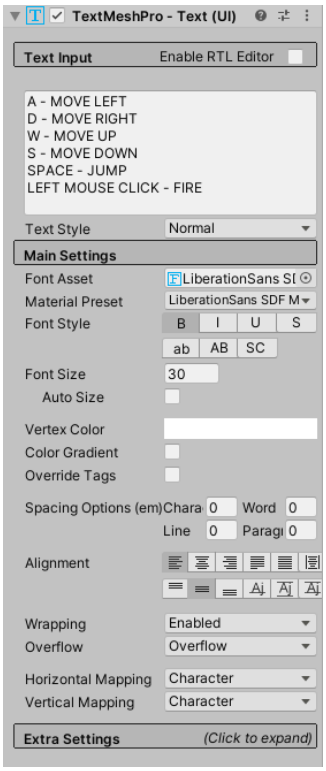
Slika 18: Slika gumba

3.2.4 Controls scena

Slika 19 prikazuje Controls scenu koja sadrži gumb za povratak na scenu „Options“ koji je implementiran na isti način kao i ostali gumbovi. Također, sadrži tekst sa uputama kako igrač može upravljati sa svojim likom.

Slika 19: Controls scena

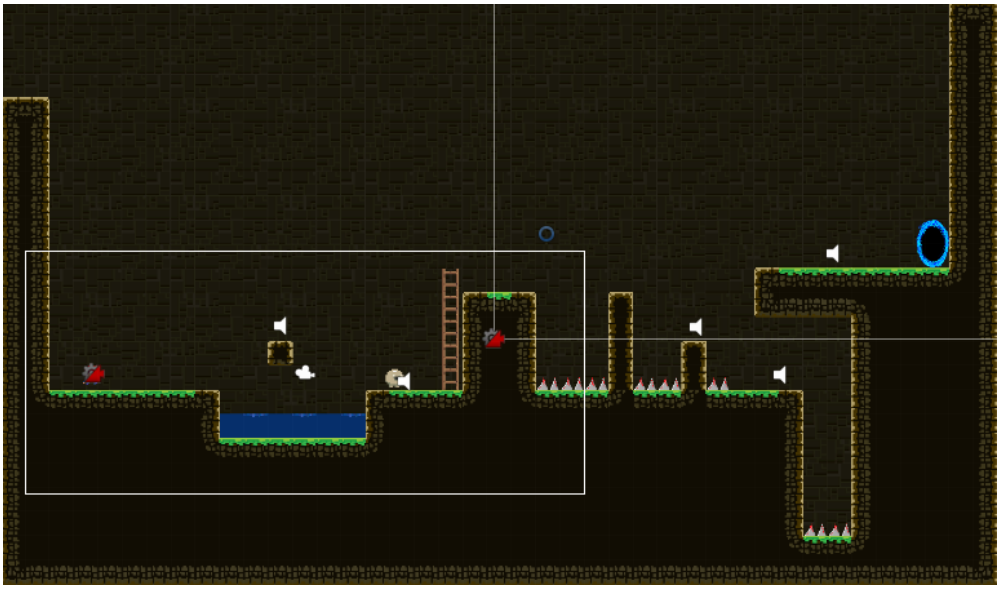
Kako bi tekst bio vizualno kvalitetniji, koristi se TextMeshPro (slika 20) koji koristi napredne tehnike renderiranja teksta koje su naprednije od uobičajenog teksta ugrađenog u Unity. Osim vizualne kvalitete, TextMeshPro nudi jednako dobre performanse kao običan tekst, ali i mnogo više opcija za uređivanje teksta ovisno o vlastitim potrebama.



Slika 20: TextMeshPro

3.2.5 Level scene

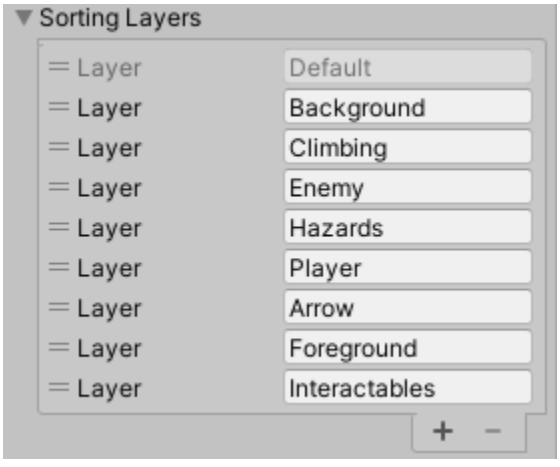
Igra ukupno sadrži 5 levela. Svaki level je zasebna scena, odnosno, prelaskom sa Levela 1 na Level 2, zapravo prelazimo sa scene „Level 1“ na scenu „Level 2“ itd. Osim samog levela, možemo vidjeti broj života igrača, te količinu sakupljenih novčića u igri. Na slici 21 možemo vidjeti izgled prvog levela u igri.



Slika 21: Level 1 scena

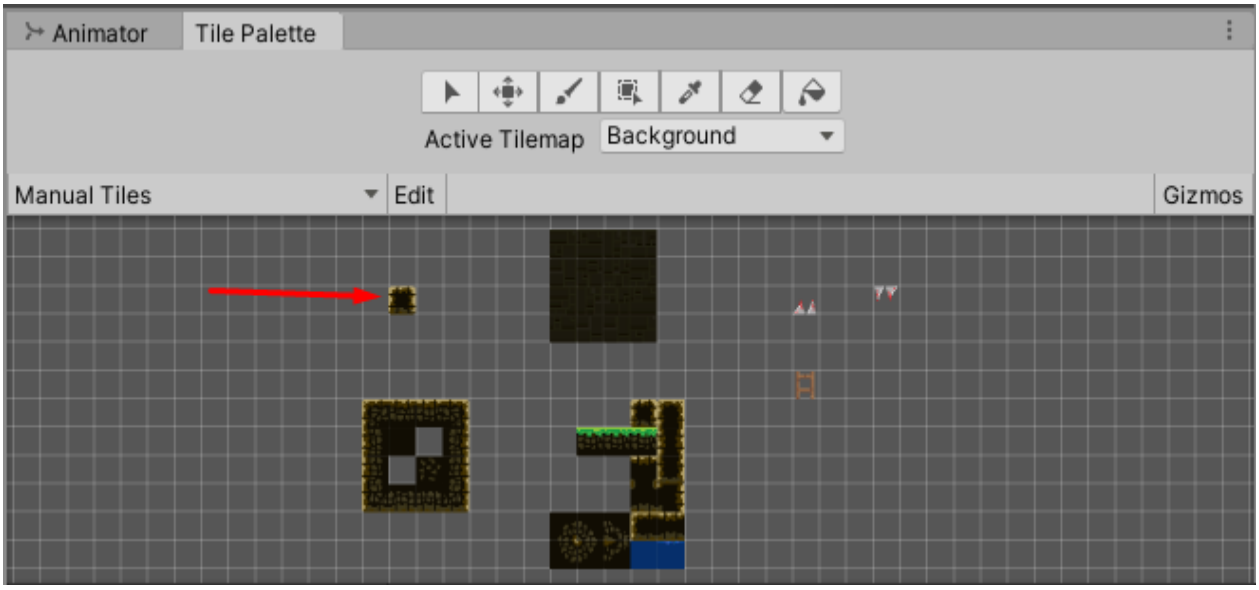
3.2.5.1 Dizajn levela

Prije nego što krenemo sa samim dizajniranjem, važno je odrediti Sorting Layers (slika 22) koji određuju kojim će se redom određeni spriteovi renderirati. Layeri označavaju grupu objekata koji imaju neku zajedničku karakteristiku. Npr layer za pozadinu neće imati interakciju sa igračem, te ako još neki objekt stavimo u skupinu layera sa pozadinom, taj objekt također neće imati interakciju sa igračem nego će se nalaziti u pozadini [17]. Ako odaberemo Foreground, on će se automatski prikazati ispred Background zato što smo tako odredili unutar Sorting Layers taba.



Slika 22: Sorting Layers

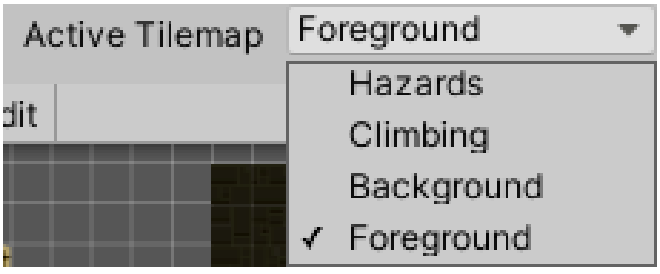
Nakon što namjestimo Sorting Layers, možemo započeti sa dizajniranjem. Svaki od levela dizajniran je pomoću *Tile Palette* (slika 23), a kako bi olakšali proces dizajniranja, stvorimo *Rule Tile* (slika 24). Pomoću *Rule Tile-a* možemo „namjestiti“ kojim će se redom postavljati određeni *spriteovi*. Npr. ukoliko imamo *sprite* koji označava površinu kojom se igrač kreće (npr. trava). Unutar *Rule tilea* odredimo da se on automatski postavi ukoliko iznad njega nema ni jednog drugog *spritea*, a ima ispod i sa strane (slika 24 - desno). Potom ćemo unutar *Tile Palette* odabrati jedan *sprite* i na njega povezati kreirani *Rule Tile*. Tako ćemo dobiti samo jedan *sprite* kojim ćemo na brzi način moći dizajnirati velike levele, odnosno automatski postavljati sve *sprite-ove* (slika 23, označeno crvenom strijelicom).



Slika 23: Tile Palette sa označenim spriteom za Rule Tile

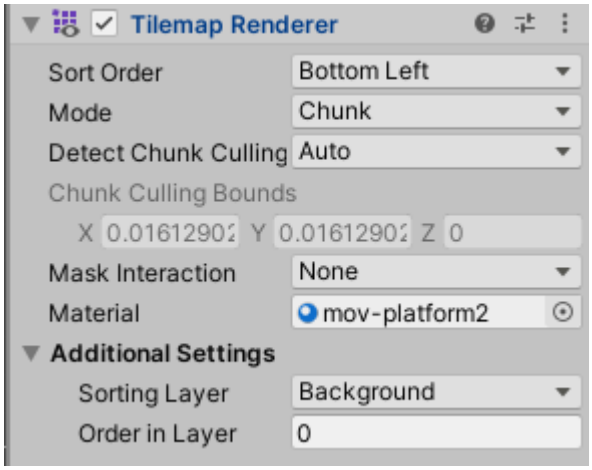
Slika 24: Rule Tile

Unutar Tile Palette, klikom na *Active Tilemap* (slika 25), dobijemo padajući izbornik unutar kojeg možemo odabrati koji ćemo dio levela dizajnirati. *Hazards* Tilemap služi za dodavanje prepreka, kao što su u ovoj igri bodlje ili voda. *Climbing* Tilemap dodaje ljestve na koje se igrač može penjati, a *Foreground* dodaje dio levela po kojem igrač može hodati i sa kojim može imati interakciju. *Climbing* i *Foreground* mora imati komponentu *Rigidbody 2D* koju dodajemo unutar *inspectora*.



Slika 25: Active Tilemap

Kako bi objekt znao koji se Tilemap koristi za njegovu svrhu, moramo u Inspectoru dodati Tilemap Renderer prikazan na slici 26. On će upravljati kako će se pojedini Tile postavljen na Tilemapu renderirati.



Slika 26: Tilemap Renderer za pozadinu

3.2.6 Pause Menu scena

Ukoliko igrač pritisne „Escape“ dugme na tipkovnici, otvara se meni za pauzu. Unutar menia, nalaze se tri gumba. Gumb *Resume* će nastaviti igru tamo gdje je igrač stao kada je pritisnuo pauzu. Gumb *Menu*, vraća igrača na glavni Meni, a gumb *Quit* gasi igru (slika 27).

Slika 27: Meni za pauzu

Meni za pauzu koristi skriptu *PauseMenu* koja sadrži istoimenu klasu. Unutar klase, nalaze se funkcije za svaki od tri gumba unutar menia. Najprije u funkciji Update() definiramo što će se dogoditi klikom na „Escape“ tipku na tipkovnici.

```
public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if(GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }
```

Ukoliko igrač pritisne Escape pokreće se funkcija Pause(). Ona će pokrenuti sam meni, namjestiti vrijeme na 0, odnosno zaustaviti ga te varijablu „GameIsPaused“ postaviti na true.

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

Funkcija Resume() će maknuti meni, pokrenuti vrijeme, odnosno postaviti ga na 1, te će varijablu GameIsPaused vratiti na false.

```
public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }
```

Funkcija MainMenu() koja pokreće vrijeme te pokreće funkciju RestartajSesiju() iz skripte GameController, odnosno vraća igrača na početni meni.

```
public void MainMenu()
{
 Time.timeScale = 1f;
 FindObjectOfType<GameController>().RestartajSesiju();
}
```

Funkcija QuitGame() pokreće ugrađenu Unity funkciju Quit() koja će ugasiti igru.

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

3.3 Igrač

Igrač, odnosno Player, je glavni objekt u igri. Sastoji se od animiranih spriteova prikazanih na slici 28. Igrač može pucati, penjati se, skakati, trčati te mirovati. Sadrži dynamic RigidBody2D, Capsule i Box collidere te skriptu Player koja mu omogućuje kretnje i skriptu Shooting koja mu omogućuje pucanje.

Slika 28: Spriteovi igrača

Na početku skripte Player postavimo brzinu kretnje, skoka i penjanja igrača te tzv. deathKick koji će igrača „odbaciti“ prilikom smrti kako bi dobili bolji efekt smrtnosti. isAlive je kontrolna varijabla tipa bool koju smo postavili na true, te tako kasnije u skripti možemo provjeravati je li igrač živ. Potom postavimo varijable za Rigidbody, Animator, Capsule i Box collider te gravitaciju.

```
[SerializeField] float runSpeed = 5f;
[SerializeField] float jumpSpeed = 3f;
[SerializeField] float climbSpeed = 5f;
[SerializeField] Vector2 deathKick = new Vector2(25f, 25f);

bool isAlive = true;

Rigidbody2D rigidBody;
Animator animator;
CapsuleCollider2D bodyCollider;
BoxCollider2D feet;
float gravityScaleAtStart;
```

Nakon što smo postavili varijable, u funkciji Start() dohvatimo odgovarajuće komponente pridružene nekom Game objektu unutar Unity-a koristeći generičku funkciju Component GetComponent<Type type>(). Generička funkcija GetComponent<Type type>() će napraviti referencu na instancu klase (u ovom slučaju Player) i vratiti komponentu s kojom želimo raditi i koju smo odredili unutar znakova manje i veće, npr. GetComponent<Rigidbody2D>() vraća RigidBody2D povezan za objekt Player itd [18].

```
void Start()
    {
        rigidBody = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        bodyCollider = GetComponent<CapsuleCollider2D>();
        feet = GetComponent<BoxCollider2D>();
        gravityScaleAtStart = rigidBody.gravityScale;
    }
```

3.3.1 Kretnja

Osnovna kretnja, trčanje i okret, izvedene su pomoću funkcija Run() i Flip(). Unutar skripte Run() koja služi za samu kretnju Input klasa dohvaća unose igrača sa tipkovnice ili miša. Input.GetAxis(„Horizontal“) koristimo zato što za kretnju, igrač koristi X-os, odnosno horizontalnu os. Ovisno o unosu, lijevom ili desnom, funkcija poprima vrijednost od -1 do 1 [19]. Potom u vektor spremamo brzinu kojom će se igrač kretati tako što na X osi pomnožimo horizontalnu kretnju sa brzinom kretnje, a na Y osi kažemo da brzina bude defaultna:

```
Vector2 playerVelocity = new Vector2(moveHorizontal * runSpeed,rigidBody.velocity.y);
```

RigidBody.velocity predstavlja defaultnu brzinu kretnje, te ju mi postavimo na brzinu koju smo definirali u playerVelocity. playerHasHorizontalSpeed je varijabla tipa bool. Ona uzima apsoultnu vrijednost kretnje igrača po X osi te ako je ona veća od Mathf.Epsilon (predstavlja najmanju veličinu koju float može imati, a da ona nije 0) bool će vratiti true. Potom iz animatora pozivamo animaciju za trčanje koja će se vršiti u oba smjera na osi X, ovisno o unosu igrača:

```
bool playerHasHorizontalSpeed = Mathf.Abs(rigidBody.velocity.x) > Mathf.Epsilon;
animator.SetBool("Running", playerHasHorizontalSpeed);
```

Funkcija Flip() služi kako bi igrač mogao promijeniti smjer kretnje. Kao i kod funkcije Run() postavimo playerHasHorizontalSpeed. Ako je bool pozitivan, odnosno true, koristimo klasu transform koja gleda poziciju objekta u prostoru te joj pomoću Mathf.Sign pridodajemo vektor koji će postati pozitivan (+1) ili negativan (-1) ovisno o kretnji po X osi, a na Y osi će ostati default. Ukoliko je negativan, igrač će gledati lijevo, a ukoliko je pozitivan, igrač će gledati desno:

```
transform.localScale = new Vector2(Mathf.Sign(rigidBody.velocity.x), 1f);
```

```
private void Run()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        Vector2 playerVelocity = new Vector2(moveHorizontal * runSpeed,rigidBody.velocity.y);
        rigidBody.velocity = playerVelocity;

        bool playerHasHorizontalSpeed = Mathf.Abs(rigidBody.velocity.x) > Mathf.Epsilon;
        animator.SetBool("Running", playerHasHorizontalSpeed);
    }

    private void Flip()
    {
        bool playerHasHorizontalSpeed = Mathf.Abs(rigidBody.velocity.x) > Mathf.Epsilon;
        if (playerHasHorizontalSpeed)
        {
            transform.localScale = new Vector2(Mathf.Sign(rigidBody.velocity.x), 1f);
        }
    }
```

Skakanje i penjanje omogućeno je pomoću funkcija Jump() i Climb(). U funkciji Jump() najprije provjeravamo da li noge igrača dodiruju tlo:

```
if (!feet.IsTouchingLayers(LayerMask.GetMask("Ground")))
```

Ukoliko ne dodiruju, igrač ne može skakati. Tako sprječavamo beskonačan broj skokova u visinu kao i skakanje po zidovima. Zatim provjeravamo da li je igrač pritisnuo tipku za skok, odnosno Space dugme na tipkovnici te ukoliko jest igraču dodajemo brzinu skoka definiranu na početku koda te ju zbrajamo sa brzinom kretnje igrača:

```
if (Input.GetButtonDown("Jump")){
```

```
Vector2 dodajBrzinuSkoka = new Vector2(0f, jumpSpeed);
rigidBody.velocity += dodajBrzinuSkoka;

}
```

Funkcija Climp() provjerava da li noge igrača dodiruju ljestve koje smo nazvali „Climbing“, ukoliko ne dodiruju, prekida se animacija za penjanje te mu dodajemo početnu gravitaciju:

```
if (!feet.IsTouchingLayers(LayerMask.GetMask("Climbing"))){

animator.SetBool("Climbing", false);
rigidBody.gravityScale = gravityScaleAtStart;
return;
}
```

Ukoliko se igrač nalazi na ljestvama, pozivamo Input.GetAxis(„Vertical“) koji prati kretnje igrača po Y osi, dajemo igraču brzinu penjanja slično kao kod trčanja te gravitaciju postavljamo na 0 kako igrač ne bi klizio sa ljestvi nego ostao na njima. Ukoliko se igrač kreće po Y osi, pokrenuti će se animacija za penjanje:

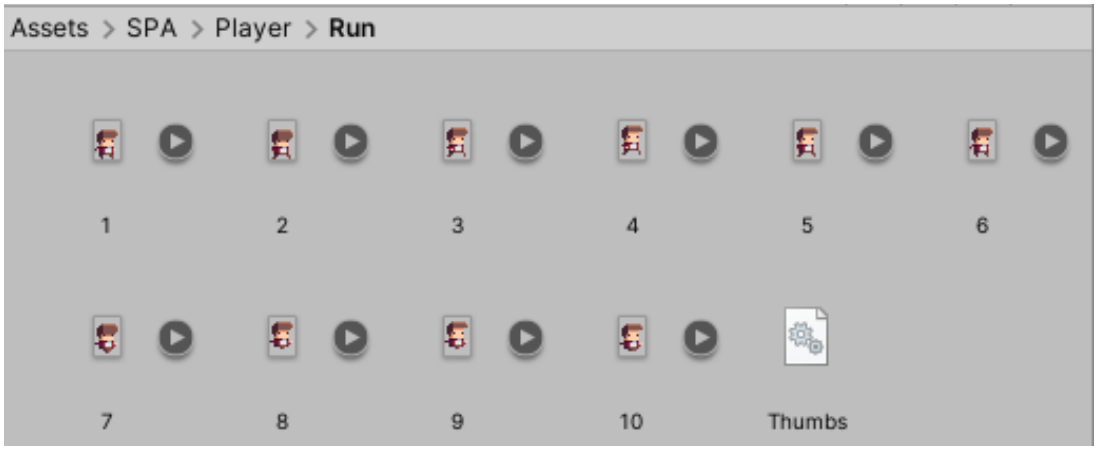
```
bool playerHasVerticalSpeed = Mathf.Abs(rigidBody.velocity.y) > Mathf.Epsilon;
animator.SetBool("Climbing", playerHasVerticalSpeed);
```

```
private void Jump()
    {
        if (!feet.IsTouchingLayers(LayerMask.GetMask("Ground")))
        {
            return;
        }


        if (Input.GetButtonDown("Jump"))
        {
            Vector2 dodajBrzinuSkoka = new Vector2(0f, jumpSpeed);
            rigidBody.velocity += dodajBrzinuSkoka;
        }
    }

    private void Climb()
    {
        if (!feet.IsTouchingLayers(LayerMask.GetMask("Climbing")))
        {
            animator.SetBool("Climbing", false);
            rigidBody.gravityScale = gravityScaleAtStart;
            return;
        }

        float moveVertical = Input.GetAxis("Vertical");
        Vector2 climbVelocity = new Vector2(rigidBody.velocity.x, moveVertical * climbSpeed);
```

```
    rigidBody.velocity = climbVelocity;
    rigidBody.gravityScale = 0f;

    bool playerHasVerticalSpeed = Mathf.Abs(rigidBody.velocity.y) > Mathf.Epsilon;
    animator.SetBool("Climbing", playerHasVerticalSpeed);
}
```

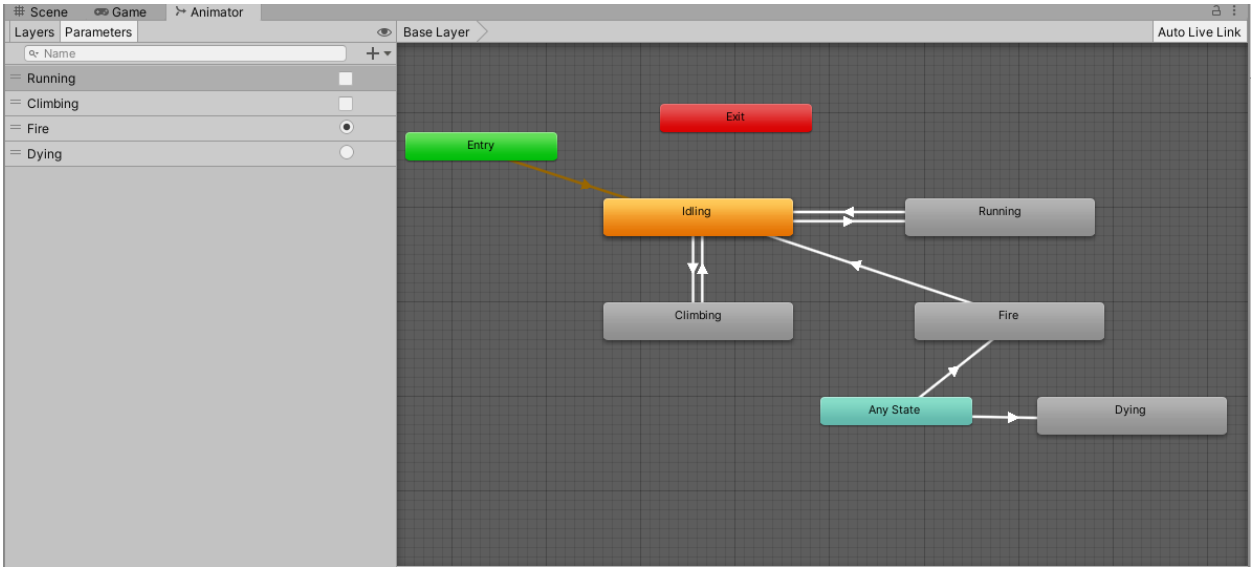
3.3.2 Animacije

Animacije su vrlo bitna stavka igara. One likovima dodaju osobine te bolji doživljaj igranja samim korisnicima [20]. Kako bi implementirali animacije, moramo ih kreirati, a potom ih povezati u Animatoru. Kreiramo ih tako što odaberemo sve spriteove koje želimo koristiti u animaciji (slika 29), pritisnemo desni klik te odaberemo Create -> Animation. Unity nam automatski poveže odabrane spriteve u jednu cjelinu i poreda ih kako bi dobili animaciju.

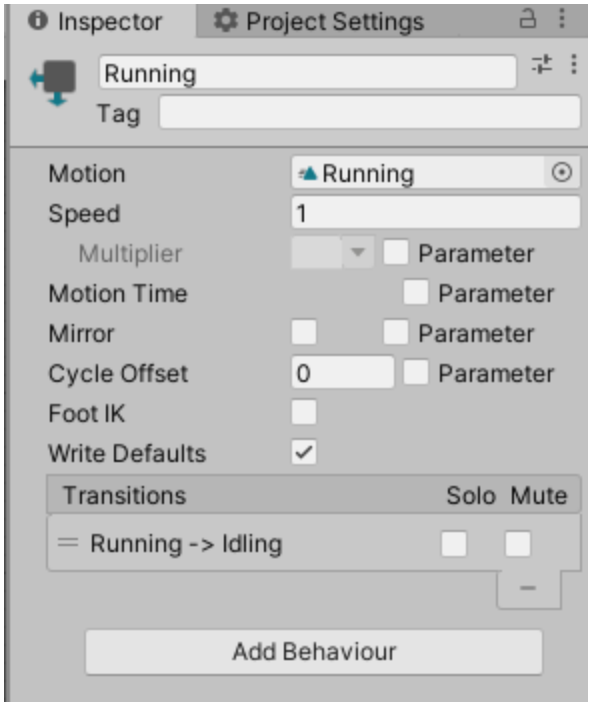


Slika 29: Spriteovi za animaciju trčanja

Nakon što smo kreirali sve potrebne animacije ubacujemo ih u Animator prikazan na slici 30. Desnim klikom na default animaciju Entry odaberemo stavku Make Transition, te napravimo tranziciju na kreiranu animaciju Idling. Unutar Idling animacije selektiramo stavku loop te će tako igrač, kada stoji mirno, cijelo vrijeme pokretati animaciju Idling. Potom smo iz Idling napravili tranzicije na animacije Running i Climbing. Kako bi Animator znao kada mora pokrenuti animaciju, moramo dodati parametre za određene animacije. Unutar Ispectora pod Conditions odaberemo kreirani parametar Running i postavimo ga na true. Unity će tako prepoznati da mora pokrenuti animaciju za trčanje ukoliko je parametar istina (true), a ukoliko je false, pokrenuti će se animacija Idling (slika 31). Postupak ponavljamo za sve preostale animacije. Dying animacija ima parametar Trigger zato što se ona poziva samo u određenim situacijama te ima tranziciju iz Any State zato što igrač može umrijeti u bilo kojem stanju pa će i bilo koja animacija preći u animaciju za umiranje ukoliko igrač umre.



Slika 30: Animator



Slika 31: Inspector za animacije

3.3.3 Pucanje i život

Glavna skripta zadužena za pucanje naziva se Shooting. Najprije strijeli dodajemo brzinu, udaljenost od lika (offset) te cooldown kako lik ne bi mogao previše brzo pucati te postavimo bool puca na true. U funkciji Update() provjeravamo da li je korisnik kliknuo lijevu tipku na mišu te da li puca, ukoliko je to istina stvaramo objekt, odnosno strijelu, dajemo joj offset,

```
GameObject objekt = (GameObject) Instantiate(arrow, new Vector2 (transform.position.x + offset.x*transform
.localScale.x, transform.position.y + offset.y), Quaternion.identity);
```

odnosno namještamo joj udaljenost od lika te joj namještamo visinu na kojoj će ona letjeti te joj dajemo rotaciju.

Zatim gledamo da li je igrač okrenut u lijevu stranu, ukoliko jest, okrećemo sprite strijele u lijevu stranu:

```
if(gameObject.transform.localScale.x == -1)
{
        objekt.transform.localScale = new Vector3(-5,5,0);
}
```

Ukoliko to ne bismo učinili, strijela bi u lijevu stranu letjela naopako. Potom objektu, odnosno strijeli dodajemo brzinu leta i kretnju po X osi, pokrećemo funkciju Pucanje() koja će pucanju dati cooldown te pokrećemo animaciju pucanja.

```
GetComponent<Animator>().SetTrigger ("Fire");
```

```
public GameObject arrow;
   public Vector2 velocity;
   public Vector2 offset = new Vector2(0.4f,-0.2f);
   public float cooldown = 0.5f;
   bool puca = true;

   void Update()
   {
      if(Input.GetMouseButtonDown(0) && puca)
      {
         GameObject objekt = (GameObject) Instantiate(arrow, new Vector2 (transform.position.x + offset.x*tr
ansform.localScale.x, transform.position.y + offset.y), Quaternion.identity);

         if(gameObject.transform.localScale.x == -1)
         {
            objekt.transform.localScale = new Vector3(-5,5,0);
         }

         objekt.GetComponent<Rigidbody2D>().velocity = new Vector2 (velocity.x * transform.localScale.x, v
elocity.y);
         StartCoroutine(Pucanje());
         GetComponent<Animator>().SetTrigger ("Fire");
      }
   }

   IEnumerator Pucanje()
```

```
    {
        puca = false;
        yield return new WaitForSeconds(cooldown);
        puca = true;
    }
```

Skripta Strijela upravlja štetom koju strijela čini protivnicima te ju uništava ukoliko dodirne površinu. Proizvoljno odaberemo količinu štete koju će strijela raditi (u ovom slučaju 50) te to pohranimo u varijablu steta. Potom u funkciji OnTriggerEnter2D koja provjerava je li došlo do kolizije između strijele i nekog objekta [21] dohvatimo hp iz skripte Zivot, protivnika te dohvatimo komponentu UnistiStrijelu. Ukoliko strijela dotakne protivnika, napraviti će mu štetu te će uništiti samu strijelu:

```
if(protivnik && hp){

        hp.NapraviStetu(steta);
        Destroy(gameObject);

}
```

Komponenta UnistiStrijelu prazna je skripta koju smo dodali objektima kao što je površina i prepreke, te ukoliko strijela dotakne neki od tih objekata ona će se uništiti.

```
[SerializeField] float steta = 50;
CapsuleCollider2D bodyCollider;

private void OnTriggerEnter2D(Collider2D col)
{
 var hp = col.GetComponent<Zivot>();
 var protivnik = col.GetComponent<EnemyMovement>();
 var unistiStrijelu = col.GetComponent<UnistiStrijelu>();

if(protivnik && hp)
{
 hp.NapraviStetu(steta);
 Destroy(gameObject);
}

if(unistiStrijelu)
{
 Destroy(gameObject);
}
}
```

Igrač ukupno ima 5 života. Svakim dodirom prepreke ili neprijatelja gubi 1 život, te nakon izgubljenih 5 života, vraća se na početni meni igre. Funkcija Die() unutar skripte Player() provjerava da li igrač dodiruje protivnika ili prepreku:

```
if (bodyCollider.IsTouchingLayers(LayerMask.GetMask("Enemy", "Hazards"))){},
```

ukoliko dodiruje, pokreće se animacija za umiranje, igraču se dodaje deathKick te potom pokreče funkciju Smrt() iz skripte Game Controller. Te zatim postavlja isAlive na false.

```
private void Die()
    {
        if (bodyCollider.IsTouchingLayers(LayerMask.GetMask("Enemy", "Hazards")))
        {
            animator.SetTrigger("Dying");
            GetComponent<Rigidbody2D>().velocity = deathKick;
            if(isAlive)
            {
                FindObjectOfType<GameController>().Smrt();
            }
            isAlive = false;
        }
    }
```

U skripti GameController, unutar funkcije Start() definiramo ukupan broj života igrača.

```
brojZivota = 5;
```

Potom funkcija Smrt() provjerava je li broj života igrača veći od 1, ukoliko jest, pokreće se funkcija za oduzimanje života. Ukoliko je broj života manji od 0. Pokreće se funkcija za ponovno pokretanje sesije.

```
public void Smrt()
    {
        if (brojZivota > 1)
        {
            OduzmiZivot();
        }else
        {
            RestartajSesiju();
        }
    }
```

Funkcija OduzmiZivot() smanjuje broj života za jedan, dohvaća indeks trenutne scene te ju ponovno pokreće, ali ovoga puta tekst koji označava broj života postaje za jedan broj manji.

```
private void OduzmiZivot()
    {
        brojZivota--;
        var currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(currentSceneIndex);
        textZaZivot.text = brojZivota.ToString();
    }
```

Funkcija RestartajSesiju() će pokrenuti scenu glavnog menia te će uništiti gameObject, odnosno obrisati igrača sa scene.

```
private void RestartajSesiju()
    {
        SceneManager.LoadScene("Main Menu");
        Destroy(gameObject);
    }
```

3.3.4 Novčići

Osim pucanja na neprijatelje, igrač, prelazeći mape, može sakupljati novčiće (slika 32).

Slika 32: Novčići

Svaki novčić vrijedi 10 bodova te ukoliko ih sakupi i umre, ne može ih više sakupiti, ali se rezultat pamti sve dok igrač ne izgubi svih 5 života. Maksimalan broj novčića koje igrač može sakupiti je 40 što donosi ukupno 400 bodova. Sakupljanje novčića implementirano je u skripti SakupljanjeNovcica. Osim vrijednosti svakog novčića, moramo definirati i zvuk za sakupljanje novčića (AudioClip). Potom unutar funkcije OnTriggerEnter2D definiramo varijablu igrač koja će dohvatiti objekt Player, odnosno samog igrača. OnTriggerEnter2D je ugrađena Unity funkcija koja provjerava da li su se dva objekta međusobno dodirnula. Ukoliko igrač dodirne novčić pokrenuti će se funkcija PovecajNovac iz skripte GameController, pokreće se zvuk sakupljanja novčića i uništava se objekt, odnosno novčić.

```
[SerializeField] int kolicinaNovaca = 10;
[SerializeField] AudioClip zvukSakupljanja;
private bool collected = false;


private void OnTriggerEnter2D(Collider2D col)
{
```

```
var igrac = col.GetComponent<Player>();
if(igrac && collected == false)

{
 FindObjectOfType<GameController>().PovecajNovac(kolicinaNovaca);
 AudioSource.PlayClipAtPoint(zvukSakupljanja, Camera.main.transform.position);
 Destroy(gameObject);
 }
}
```

Kao i za život, unutar GameController skripte definiramo početni broj novčića koji je jednak 0

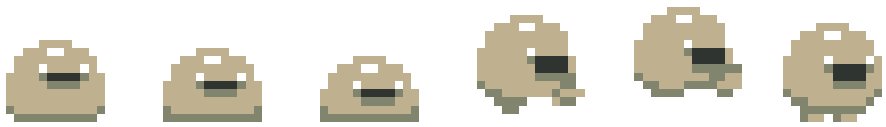
```
kolicinaNovaca = 0;
```

Potom funkcija PovecajNovac() prima vrijednost novca te trenutnu količinu novca povećava za 10 i to ispisuje na ekran

```
public void PovecajNovac(int novac)
   {
      kolicinaNovaca += novac;
      textZaNovac.text = kolicinaNovaca.ToString();
   }
```

3.4 Neprijatelj i prepreke

Kako bi otežali gameplay, dodajemo neprijatelje i prepreke koje igrač mora ili ubiti ili prekočiti. Neprijatelji su prikazani na slici 33 te je njih moguće ubiti i preskočiti.



Slika 33: Neprijatelj

Njihova kretnja je omogućena u skripti EnemyMovement. Unutar skripte definiramo brzinu kretnje neprijatelja, te u Start() funkciji dohvatimo sam objekt:

```
myRigidBody = GetComponent<Rigidbody2D>():
```

Unutar Update() funkcije provjeravamo da li je lik okrenut prema desnoj strani pozivom funkcije IsFacingRight() koja provjerava da li je vrijednost na X osi veća od 0. Ukoliko neprijatelj gleda u desnu stranu, dodajemo mu pozitivnu brzinu kretnje na X osi:

```
myRigidBody.velocity = new Vector2(moveSpeed, 0f;
```

a ukoliko ne gleda desno, odnosno okrenut je na lijevu stranu, dodajemo mu negativnu vrijednost brzine na X osi te mu tako dajemo brzinu u lijevu stranu:

```
myRigidBody.velocity = new Vector2(-moveSpeed, 0f);
```

Posljednje što skripta provjerava jest da li protivnik ima koliziju sa okolinom pomoću ugrađene funkcije OnTriggerExit2D, te ukoliko ima, protivnik mijenja smjer slično kao što igrač mijenja smjer pritiskom na tipku:

```
private void OnTriggerExit2D(Collider2D collision)
{
        transform.localScale = new Vector2((Mathf.Sign(myRigidBody.velocity.x)), 1f);
}
```

```
[SerializeField] float moveSpeed = 1f;

    Rigidbody2D myRigidBody;

    void Start()
    {
        myRigidBody = GetComponent<Rigidbody2D>();
    }


    void Update()
    {
        if (IsFacingRight())
        {
            myRigidBody.velocity = new Vector2(moveSpeed, 0f);
        }
        else
        {
            myRigidBody.velocity = new Vector2(-moveSpeed, 0f);
        }

    }

    bool IsFacingRight()
    {
        return transform.localScale.x > 0;
    }

    private void OnTriggerExit2D(Collider2D collision)
```

```
    {
        transform.localScale = new Vector2((Mathf.Sign(myRigidBody.velocity.x)), 1f);
    }
```

Protivnik ukupno ima 1 život, odnosno 100 hp-a. Ukoliko strijela pogodi protivnika, oduzeti će mu ukupno 50 hp-a odnosno, potrebne su dvije strijele kako bi se uništio protivnik. Skripta Zivot prati stetu učinjenu protivniku. Skripta daje 100 hp-a protivniku:

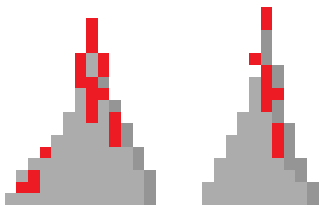
```
[SerializeField] float hp = 100f;
```

potom poziva funkciju NapraviStetu() koja od ukupnog hp-a oduzima načinjenu štetu, te ukoliko je hp manji ili jednak nuli uništava protivnika:

```
hp -= steta;
if(hp <= 0){
Destroy(gameObject);
}
```

```
[SerializeField] float hp = 100f;

    public void NapraviStetu(float steta)
    {
        hp -= steta;
        if(hp <= 0)
        {
            Destroy(gameObject);
        }
    }
```

Osim neprijatelja u igri imamo i dvije vrste prepreka. To su bodlje i voda (slike 34 i 35).



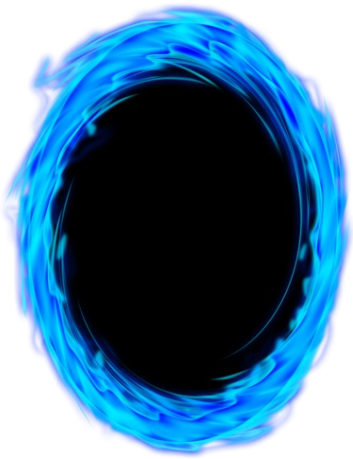
Slika 34: Bodlje

Slika 35: Voda

Njihove funkcionalnosti opisane su prethodno u skripti Player. Osim prazne skripte UništiStrijelu nema povezanih skripti. Elementi povezani za njih jesu Tilemap, TilemapRenderer i Tilemap Collider kako bi ih mogli prikazati na mapi i postavljati na mapu jednostavnije (opisano u poglavlju 3.2.5.1 Dizajn levela). Ukoliko igrač dodirne prepreku, gubi život te se vraća na početak levela.

3.4 Izlazak iz levela i završna scena

Dolaskom do portala prikazanom na slici 36. igrač prelazi na novi level. Skripta koja je zaslužna za funkcionalnost portala naziva se LevelExit.



Slika 36: Portal

U skripti smo definirali delay za prelazak na idući level i slow motion. Potom smo definirali IENumerator[] LoadNextLevel() koji će usporiti vrijeme za veličinu slow motion-a:

```
Time.timeScale = LevelExitSlowMoFactor;
```

dodati delay prije prelaska na idući level:

```
yield return new WaitForSecondsRealtime(LevelLoadDelay);
Time.timeScale = 1f;
```

te potom dohvatiti indeks sadašnje scene i uvećati ga za 1, odnosno preći na iduću scenu.

```
var currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
SceneManager.LoadScene(currentSceneIndex + 1) ;
```

```
[SerializeField] float LevelLoadDelay = 2f;
[SerializeField] float LevelExitSlowMoFactor = 0.2f;

void OnTriggerEnter2D(Collider2D other)
{
 StartCoroutine(LoadNextLevel());
}

IEnumerator LoadNextLevel()
{
 Time.timeScale = LevelExitSlowMoFactor;
 yield return new WaitForSecondsRealtime(LevelLoadDelay);
 Time.timeScale = 1f;
 var currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
 SceneManager.LoadScene(currentSceneIndex + 1);
}
```

Nakon što prođe sve levele, igrač dolazi na posljednju scenu prikazanu na slici 37, gdje se ispisuje čestitka i prikazuje se gumb za povratak u glavni meni. Također, u kutevima se ispisuju i ukupan broj sakupljenih novčića te preostali broj života.

Slika 37: Posljednja scena

3.5 Statistika

Kako bi igrač mogao pratiti svoju statistiku, u igru je ugrađena scena „Stats“ (slika 38). Scena sadrži dva gumba, gumb za povratak na prethodni meni te gumb za resetiranje statistike. Statistika koju igra pamti je:

- najbolje ostvareni rezultat (HighScore)

- broj sveukupno uništenih protivnika (Total Enemies Killed)

- broj sveukupno ispaljenih strijela (Total Arrows Shot)

- broj sveukupno izgubljenih života (Total Health Lost)

Slika 38: Stats scena

Kako bi igra vodila statistiku, morali smo napraviti skripte za upisivanje i pamćenje podataka tijekom igre. Skripta zadužena za praćenje najbolje ostvarenog rezultata naziva se „Leaderboard“. Unutar skripte definiramo text za highscore te ključ za pamćenje statistike. Unutar funkcije start nazali se funkcija PlayerPrefs.GetInt(string key, int default value) koja vraća vrijednost definiranu na ključu ili defaultnu vrijednost ukoliko vrijednost ključa nije definirana [22].

```
highScoreBroj.text = PlayerPrefs.GetInt(highScoreKey,0).ToString();
```

Potom smo definirali funkciju NamjestiNaNula() koja će ispisati broj nula nakon što kliknemo gumb za reset.

```
Public Text highScoreBroj;
string highScoreKey = „HighScore“;

void Start()
{
 highScoreBroj.text = PlayerPrefs.GetInt(highScoreKey,0).ToString();
}

public void NamjestiNaNula()
{
 highScoreBroj.text = „0“;
}
```

Unutar skripte GameController definiramo početnu vrijednost HighScorea:

```
static int highScoreBroj = 0;
```

Te u funkciji PovecajNovac() dodajemo provjeru je li trenutni rezultat veći od prethodno ostvarenog najboljeg rezultata, te ako jest, trenutni rezultat će postati najbolje ostvareni rezultat te spremimo taj rezultat i ispišemo ga na ekran:

```
if(kolicinaNovaca > highScoreBroj){
PlayerPrefs.SetInt(highScoreKey,kolicinaNovaca);
PlayerPrefs.Save();
}
```

```
public void PovecajNovac(int novac)
    {
        kolicinaNovaca += novac;
        textZaNovac.text = kolicinaNovaca.ToString();

        if(kolicinaNovaca > highScoreBroj)
        {
            PlayerPrefs.SetInt(highScoreKey,kolicinaNovaca);
            PlayerPrefs.Save();
        }

    }
```

Za vođenje statistike života igrača zaslužna je skripta ZivotCounter. Skripta sadrži definirani tekst za život te Update() funkciju koja sadži funkciju PlayerPrefs.GetInt(string key, int default value) čija je funkcionalnost jednaka kao u skripti za najbolje ostvareni rezultat

```
public Text zivot;

    void Update()
    {
        zivot.text = PlayerPrefs.GetInt("zivot").ToString();
    }
```

Kako bi ispisivali broj izgubljenih života, unutar skripte GameController smo definirali brojač nazvan „zivot“ te smo ga postavili na 0:

```
private int zivot = 0;
```

Potom smo unutar funkcije OduzmiZivot() povećali brojač za 1 svaki put kada je igrač izgubio život, te smo broj ispisali u statistiku:

```
zivot++;

PlayerPrefs.SetInt("zivot",zivot);
```

```
private void OduzmiZivot()
    {
        brojZivota--;
        var currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(currentSceneIndex);
        textZaZivot.text = brojZivota.ToString();
        zivot++;
        PlayerPrefs.SetInt("zivot",zivot);
    }
```

Statistika za protivnike i strijele implementirana je na jednak način kao statistika za život unutar skripti „EnemyCounter“ i „ArrowCounter“ te skripti „Zivot“ i „Shooting“.

Kako bi mogli resetirati dosadašnju statistiku, unutar skripte „GameController“ definirali smo funkciju ResetajStatse() koja će klikom na gumb pokrenuti funkciju PlayerPrefs.DeleteKey(string key) izbrisati ključ za svaki podatak odnosno, obrisati će dosad zapamćenu statistiku [23]. Tako će npr:

```
PlayerPrefs.DeleteKey("strijela");
```

obrisati statistiku za ukupan broj ispaljenih strijela.

```
public void ResetajStatse()
    {
        PlayerPrefs.DeleteKey("HighScore");
        PlayerPrefs.DeleteKey("neprijatelji");
        PlayerPrefs.DeleteKey("strijela");
        PlayerPrefs.DeleteKey("zivot");
}
```

4. Zaključak

U ovom radu detaljno je prikazan razvoj 2D platformerske video igre korištenjem Unity razvojog alata i C# programskog jezika. Objašnjeni su glavni elementi igre, odnosno igrač, protivnik, prepreke te dizajn samih levela. Svi assetsi su preuzeti s interneta (uz nekoliko promjena u bojama) kao i glazba i zvukovi u igri. Animacije su stvorene u samom Unityu.

Igra ima mogućnost proširenja. Moguće je dodati više protivnika, više levela kao i mogućnost promjene glavnog lika i sl.

U radu je najprije prikazano Unity sučelje te osnovne funkcionalnosti istog, a potom je prikazan pojedini objekt i programski kod vezan za svaki od tih objekata. Igra izgleda jednostavno, ali implementacija svih dijelova igre nije te se kroz čitavi rad može naučiti puno o izradi 2D video igara i programiranju.

5. Bibliografski navod

[1] [Mrežno] https://en.wikipedia.org/wiki/Pong [Pristupano 8. kolovoz 2020.]

[2] [Mrežno] https://express.24sata.hr/tehno/gaming-industrija-koja-najbrze-raste-i-zara-uje-14723 [Pristupano 4. kolovoz 2020.]

[3] [Mrežno] https://en.wikipedia.org/wiki/Platform_game [Pristupano 8. kolovoz 2020.]

[4] [Mrežno] https://docs.unity3d.com/Manual/index.html [Pristupano 8. kolovoz 2020.]

[5] [Mrežno] https://docs.unity3d.com/Manual/ProjectView.html [Pristupano 8. kolovoz 2020.]

[6] [Mrežno] https://docs.unity3d.com/Manual/Console.html [Pristupano 8 kolovoz 2020.]

[7] [Mrežno] https://docs.unity3d.com/Manual/Hierarchy.html [Pristupano 8. kolovoz 2020.]

[8] [Mrežno] https://docs.unity3d.com/Manual/UsingTheSceneView.html [Pristupano 8. kolovoz 2020.]

[9] [Mrežno] https://docs.unity3d.com/Manual/GameView.html [Pristupano 8. kolovoz 2020.]

[10] [Mrežno] https://docs.unity3d.com/Manual/UsingTheInspector.html [Pristupano 8. kolovoz 2020.]

[11] [Mrežno] https://docs.unity3d.com/Manual/class-SpriteRenderer.html [Pristupano 8. kolovoz 2020]

[12] [Mrežno] https://docs.unity3d.com/Manual/class-Rigidbody2D.html [Pristupano 8. kolovoz 2020]

[13] [Mrežno] https://docs.unity3d.com/Manual/class-CapsuleCollider2D.html [Pristupano 9. kolovoz 2020]

[14] [Mrežno] https://docs.unity3d.com/Manual/class-BoxCollider2D.html [Pristupano 9. kolovoz 2020]

[15] [Mrežno] https://docs.unity3d.com/Manual/ScriptingSection.html [Pristupano 9. kolovoz 2020]

[16] [Mrežno] https://docs.unity3d.com/Manual/comp-ManagerGroup.html [Pristupano 9. kolovoz 2020]

[17] [Mrežno] https://docs.unity3d.com/Manual/Layers.html [Pristupano 8. kolovoz 2020]

[18] [Mrežno] https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html [Pristupano 9. kolovoz 2020]

[19] [Mrežno] https://docs.unity3d.com/ScriptReference/Input.GetAxis.html [Pristupano 10. kolovoz 2020]

[20] [Mrežno] https://docs.unity3d.com/ScriptReference/Animation.html [Pristupano 10. kolovoz 2020]

[21][Mrežno] https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html [Pristupano 10. kolovoz 2020]

[22][Mrežno] https://docs.unity3d.com/ScriptReference/PlayerPrefs.GetInt.html [Pristupano 20. kolovoz 2020]

[23][Mrežno] https://docs.unity3d.com/ScriptReference/PlayerPrefs.DeleteKey.html [Pristupano 20. kolovoz 2020]

6. Popis slika

7. Prilozi

Uz ovaj rad priložena je igra te Unity projekt sa svim komponentama za izradu igre