Izrada interaktivne igre u grafičkom pokretaču Unity

Korov, Mario

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Mario Korov

Izrada interaktivne igre u grafičkom pokretaču Unity

Završni rad

Osijek, 2020.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Mario Korov

Izrada interaktivne igre u grafičkom pokretaču Unity

Završni rad

Mentor: doc. dr. sc. Domagoj Ševerdija

Osijek, 2020.

Title: Making an interactive game using Unity game engine

Sažetak:

U ovom radu proučit ćemo kako korištenjem Unity pokretača i integriranog dodatka Visual Studia izraditi interaktivnu igru. Za početak ćemo odabrati vrstu projekta i upoznati se s izgledom i funkcionalnosti Unity sučelja, a nakon toga ćemo postaviti neke osnovne elemente scene.

Središnji dio ovog rada bit će stvaranje interaktivnog lika te njegova povezanost sa ostalim elementima u sceni. Kako budemo dodavali elemente koji će biti u interakciji sa likom, struktura koda će postajati kompleksnija.

Na kraju ćemo pokazati kako igru možemo jednostavno eksportirati za Windows sustave, ali i mogućnost eksportiranja na druge sustave i konzole. Nakon eksportiranja igra je gotova i spremna za pokretanje.

Kljucne riječi:

3D projekt, Unity, C#, Visual Studio

Abstract:

In this paper, we will study how to make an interactive game using the Unity game engine and integrated extension Visual Studio. At the beginning we will choose a type of project and get familiar with Unity layout, after that, we will set up some basic elements of the scene.

The center part of this paper will be the creation of an interactable character and his connection with other elements in the scene. As we add new elements that will be in interaction with the character, the code structure will become more complex.

In the end, we will see how to export the game for the Windows system and also the possibility of exporting it to some other system and consoles. Once the game is exported, it is ready to play.

Key words:

3D project, Unity, C#, Visual Studio

Sadržaj

1. Uvod 1
2. Stvaranje 3D projekta 2
3. Postavljanje scene 4
4. Stvaranje interaktivnog lika 5
5. Stvaranje prepreka i cilja 9
6. Korisničko sučelje 10
7. GameManager 12
8. Animacija i zvuk 14
9. Izvoz igre 17
10. Zaključak 20

1 Uvod

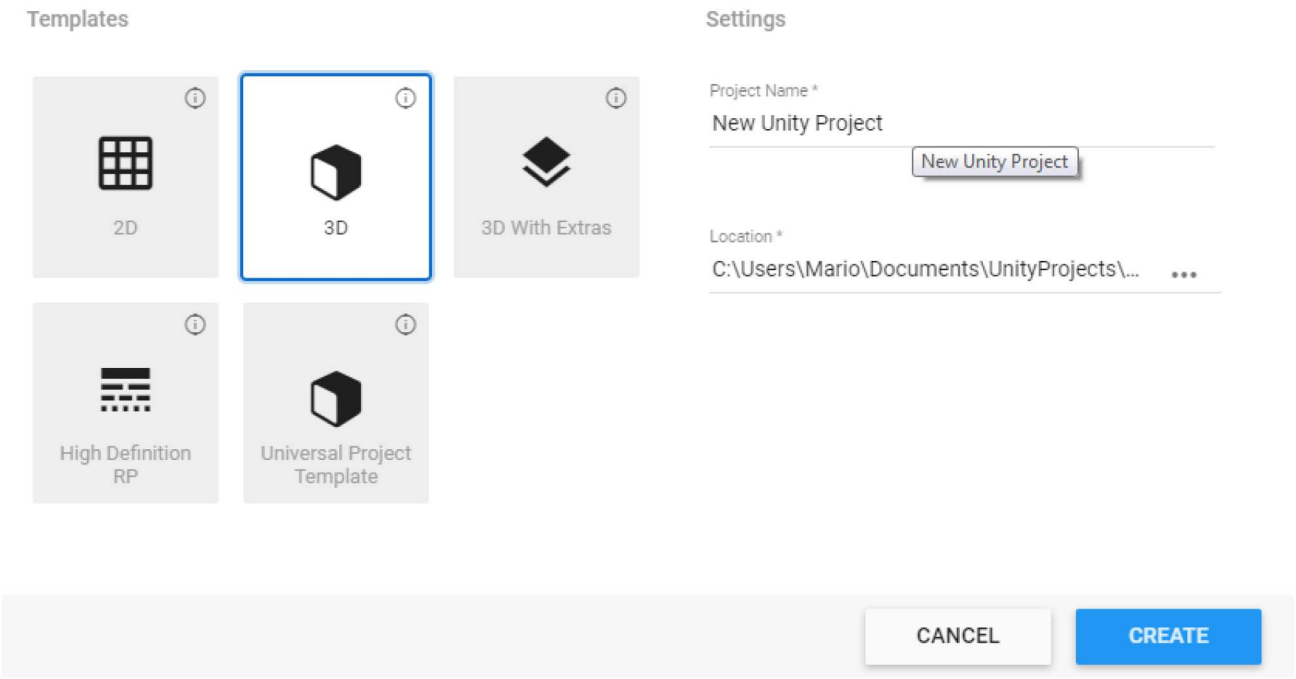
Unity je profesionalni alat za izradu i pokretanje igara, a prva verzija je izdana 2005. godine samo za Mac operacijski sustav. Danas je Unity jedan od najboljih alata za izradu igara koji podržava gotovo sve najpoznatije platforme i konzole (Windows, Mac, iOs, Android, Xbox, Playstation, WebGL itd.)[1]. Postoji besplatna i plaćena verzija, a razlika je u podršci koju plaćena verzija dobija zajedno sa nekim dodatnim materijalima i mogućnostima. Za potrebe ovog rada koristit ćemo besplatnu Unity Widnows verziju i integrirani dodatak Visual Studio. Programski jezik koji ćemo koristiti je C#.

Kako bi pokrenuli Unity, koristit ćemo dodatak Unity Hub koji omogućava lakši pregled svih projekata i stvaranje projekta u više verzija Unity pokretača. Za izradu ovog projekta, točnije interaktivne igre, koristit ćemo najnoviju Unity 2019 verziju koja ima LTS (*Long Term Support*).

Slika 1.1: Unity logo

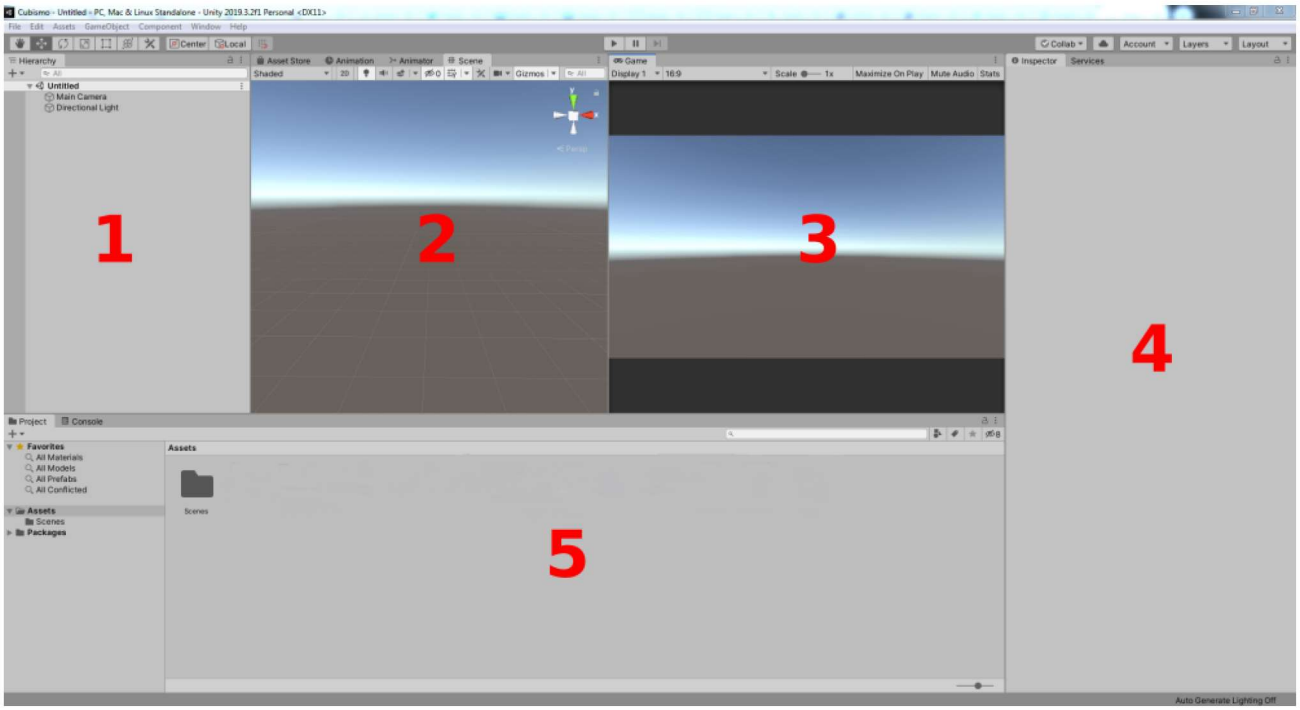
2 Stvaranje 3D projekta

Kako bi stvorili projekt potrebno je otvoriti Unity Hub koji će prikazati prozor sa listom prijašnjih projekata ukoliko ih imamo te tipku za stvaranje novih. Pritiskom tipke *New* otvara se novi prozor (Slika 2.1).



Slika 2.1: Odabir vrste projekta.

Iz navedene slike vidimo da možemo odabrati nekoliko vrsta projekata, a dvije glavne kategorije su 2D i 3D projekt. Izabrat ćemo 3D projekt i nazvat ćemo ga Cubismo. Nakon što smo odabrali vrstu projekta, otvorit će se Unity Editor (Slika 2.2) sa svojim osnovnim dijelovima:



Slika 2.2: Layout Unity Editora

1. *Hierarchy window*- hijerarhijski prikaz svih objekata (GameObject-a) koji se nalaze u sceni [1]

2. *Scene view*- omogućava navigaciju i editiranje scene, svi objekti koji se nalaze u sceni se odabiru, manipuliraju i mijenjaju kroz Scene view koji može biti iz 2D ili 3D perspektive, ovisno o vrsti projekta na kojem radimo [1]

3. *Game view*- simulira renderiranu verziju igre kroz kamere koje se nalaze u sceni, pritiskom tipke Play pokreće se simulacija igre koja predstavlja završni proizvod [1]

4. *Inspector window*- prikazuje detaljne informacije izabranog GameObject-a, omogućava pregled i mijenjanje komponenata i njihovih svojstava [1]

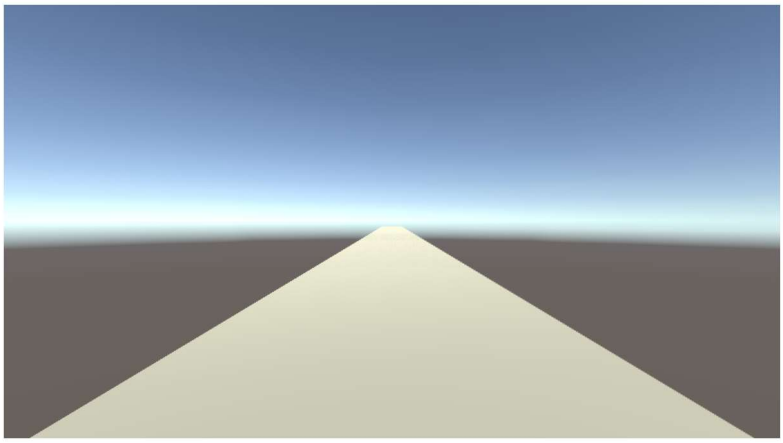
5. *Project window*- prikazuje sve datoteke i materijale koje smo kreirali ili unijeli u projekt, služi kao organizacijski prozor te je moguće kreirati nove mape u koje će se materijali po potrebi premjestiti i organizirati [1]

3 Postavljanje scene

Prije stvaranja interaktivnog lika potrebno je postaviti scenu, točnije statični objekt po kojem će se lik kretati te namjestiti postavke kamere i svjetla. Svaka scena već u sebi sadrži izvor svjetla i glavnu kameru kroz koju igrač promatra scenu. Kako bi stvorili statični objekt koristit ćemo ugrađeni Unity objekt generator koji ima mogućnost stvaranja 2D likova, ali i jednostavnih 3D geometrijskih tijela.

Desnim klikom na prozor *Hierarchy* i odabirom *3D Object->Cube* stvorit ćemo kocku u sceni. Klikom na kocku u prozoru *Inspector* izlistat će se komponente koje ta kocka sadrži, a manipulirat ćemo svojstvom *Size* iz komponente *Transform* koja dopušta manipuliranje položaja, rotacije i veličine objekta.

Postavljanjem veličine (*Size*) na 15 za x-os i 300 za z-os proširili smo i produžili kocku. Promijenit ćemo ime dobivenog kvadra u "Platform" i time smo završili sa stvaranjem statičnog objekta po kojem će se interaktivni lik kretati. Premještanjem kamere na položaj iznad Platforme (Slika 3.1), promjenom pozadine scene u *Solid Color* i promjenom svjetlosti u bijelu boju (Slika 3.2) osnovno postavljanje scene je gotovo.



Slika 3.1: Položaj kamere

Slika 3.2: Promjena pozadine i svjetla

4 Stvaranje interaktivnog lika

U ovoj igri interaktivni lik će biti kocka s kojom će igrač izbjegavati zapreke kako bi došao do cilja. Ponovit ćemo postupak stvaranja kocke, ali bez promjene veličine. Kako bi igrač mogao kontrolirati kocku prvo je potrebno dodati komponentu *Rigidbody*.

U prozoru *Inspector* pritisnemo tipku *Add Component* i odaberemo komponentu *Rigidbody*. Sada je kocka pod utjecajem fizike za koju je zadužen Unity physics engine. Komponenta *Rigidbody* nam omogućuje primjenu različitih sila s kojim kocku pomičemo u željenom smjeru. Kako sada na kocku djeluje gravitacija, *Rigidbody* uz komponentu *BoxCollider* koja se nalazi na kocki i Platformi omogućuje sudare pa kocka neće propasti kroz Platformu. Istim postupkom za dodavanje komponenti dodajemo *C# Script*(u daljnjem tekstu "skripta") komponentu koju nazovemo *PlayerMove*. Skriptu otvaramo u integriranom dodatku Visual Studio 2019. Stvaranjem skripte PlayerMove smo zapravo stvorili klasu istog imena PlayerMove koja je podklasa klase Monobehaviour (Slika 4.1)

```
public class PlayerMove : MonoBehaviour
{
    0 references
    void Start()
    {

    }
    0 references
    void Update()
    {

    }

}
```

Slika 4.1: Klasa PlayerMove

Svaka skripta koja se kao komponenta nalazi na nekom GameObject-u iz scene mora biti podklasa klase Monobehaviour te ona nasljeđuje neke osnovne funkcije koje ćemo koristiti:

1. Start()- funkcija se izvršava samo jednom prilikom pokretanja scene ili aktivacijom objekta [3]

2. Update()- funkcija koja se izvršava svaki *frame*, unutar update funkcije provjeravamo unos naredbe igrača [3]

3. FixedUpdate()- poziva se u jednakim vremenskim intervalima neovisno o frame rate-u i zbog toga služi za izvršavanje svih naredbi povezanih sa kretanjem i ostalim fizikalnim silama koje djeluju na objekte preko komponente Rigidbody [3]

U klasi PlayerMove dodajemo instancu *Rigidbody* i nekoliko bool varijabli s početnom vrijednosti false. U funkciji Start() za instancu *Rigidbody* postavljamo komponentu koja se nalazi na objektu, zatim u funkciji Update() provjeravamo unos naredbi igrača te postavljamo bool varijable u određene vrijednosti ovisno drži li igrač odabranu tipku ili ne. Na kraju u ovisnosti o bool varijablama u funkciji FixedUpdate() postavljamo brzinu komponente *Rigidbody* pomoću vektora i float varijable speed koja je *public* kako bi je mogli kroz *Inspector* lakše mijenjati.

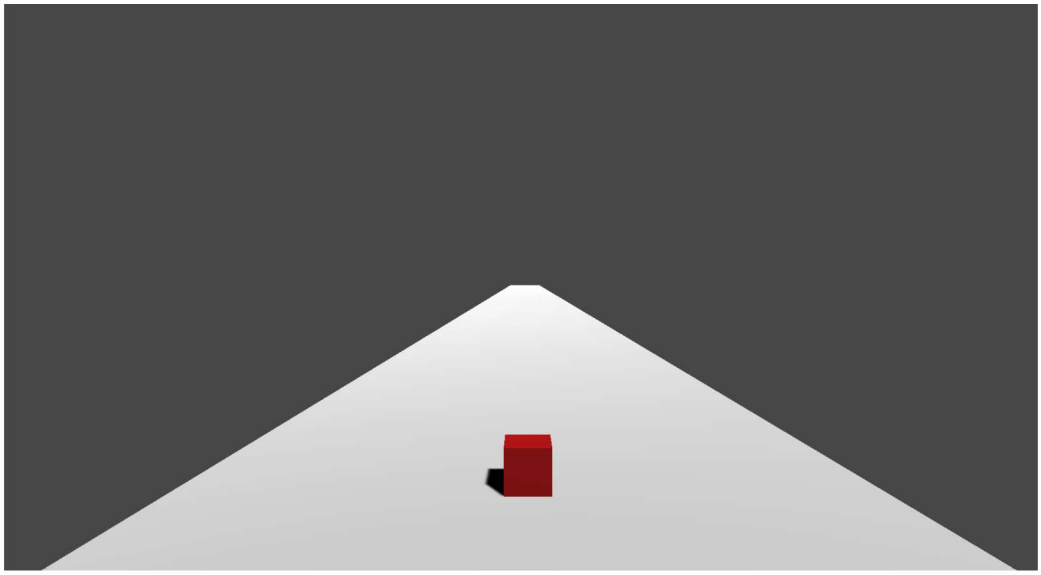
```
public class PlayerMove : MonoBehaviour
{

   Rigidbody rb;
   public float speed = 10;
   public float forwardSpeed = 15;
   bool startGame = false;
   bool left = false;
   bool right = false;
   void Start()
   {
      rb = GetComponent<Rigidbody>();
   }
   void Update()
   {
      if (Input.anyKey)
         startGame = true;
      if (startGame == true)
         rb.velocity = new Vector3(0, rb.velocity.y, forwardSpeed);
      if (Input.GetKey(KeyCode.A))
         left = true;
      else if (Input.GetKey(KeyCode.D))
         right = true;
      if (Input.GetKeyUp(KeyCode.A))
         left = false;
      if (Input.GetKeyUp(KeyCode.D))
         right = false;
   }
   private void FixedUpdate()
   {
      if (left == true)
         rb.velocity = new Vector3(-speed, rb.velocity.y, forwardSpeed);
      else if (right == true)
         rb.velocity = new Vector3(speed, rb.velocity.y, forwardSpeed);
      }
}
```

U prozoru *Inspector* postavljamo varijablu speed na željeni broj. Prilikom kretanja kocka će se nekontrolirano rotirati pa je je potrebno u komponenti *Rigidbody* onemogućiti rotaciju. Kako bi simulirali igru potrebno je pritisnuti tipku play i dok se kocka kreće prema naprijed igrač će moći pomicati kocku lijevo ili desno. Dodat ćemo skriptu CameraFollow glavnoj kameri u sceni kako bi mogla pratiti kocku.

```
public class CameraFollow : MonoBehaviour
{
   public Transform playerPositon;
   public Vector3 offset;
   void Update()
      {
         transform.position = playerPositon.position + offset;
      }
}
```

U *Inspectoru* na mjesto *Transform* instance playerPosition izabrat ćemo *Transform* komponentu koja je na kocki kako bi ju kamera pratila, a vektor Offset ćemo postaviti na vrijednosti (0,3,-10) kako bi kamera bila iza i iznad kocke. Kako bi lakše razlikovali kocku od Platforme, desnim klikom na prozor *Project Create->Material* stvorili smo materijal pod nazivom Color-Red kojeg ćemo dodati kao materijal komponente *Mesh Renderer* na kocki. Promijenit ćemo boju materijala u *Inspectoru* u crveno i tada će kocka poprimiti crvenu boju. Prije stvaranja prepreka promijenimo *tag* kocke u Player kako bi kroz kod mogli lakše identificirati kocku.



Slika 4.2: Interaktivna kocka crvene boje

5 Stvaranje prepreka i cilja

Kako bi igraču otežali put do kraja platforme (cilja) dodat ćemo prepreke. Stvorit ćemo kocku s nazivom Prepreka i dodati materijal crne boje, a zatim i skriptu Obstacle.

```
public class Obstacle : MonoBehaviour
{
  private void OnCollisionEnter(Collision collision)
  {
    if (collision.collider.CompareTag("Player"))
    {
      collision.collider.GetComponent<PlayerMove>().enabled = false ;
    }
  }
}
```

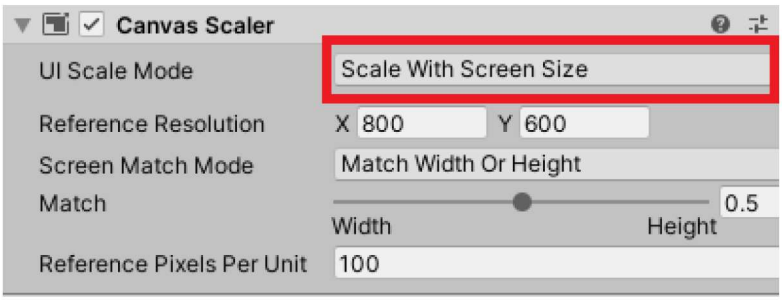
Funkcija *OnCollisionEnter* će prepoznati svaki objekt u trenutku sudara sa Preprekom, a dodatnim uvjetom osiguravamo da će se kod izvršiti samo ako je to objekt sa tagom Player, što je u ovom slučaju kocka kojom upravlja igrač. Nakon sudara, isključuje se skripta PlayerMove i igrač više nema kontrolu. Kasnije ćemo dodati određenu animaciju i tekst na ekranu kako bi igraču dali povratnu informaciju da je igra završena. Tipkama Ctrl + D dupliciramo prepreke i raspoređujemo duž platforme.

Idući korak je stvaranje ciljne linije kroz koju igrač treba proći. Stvorit ćemo kocku FinishLine sa materijalom zelene boje i proširit ćemo je na širinu Platforme. Spustit ćemo FinishLine tek toliko da se vidi iznad platforme i u komponenti *BoxCollider* odabrati *Is Trigger*. Sada igrač s kockom uz detekciju sudara može proći kroz FinishLine. Kada bi opcija *Is Trigger* ostala isključena, kocka bi se sudarila sa FinishLine i ne bi mogla proći. Dodajemo skriptu Finish kako bi isključili kontrole igrača nakon prolaska kocke kroz cilj, a kasnije ćemo dodati i tekst na ekranu koji će obavijestiti igrača o pobjedi.

```
public class Finish : MonoBehaviour
{
  private void OnTriggerExit(Collider other)
  {
    other.GetComponent<PlayerMove>().enabled = false;
  }
}
```

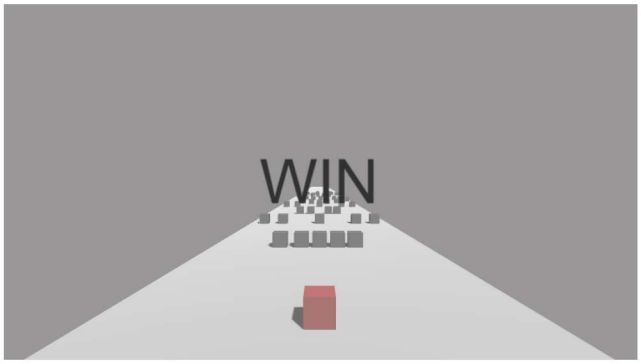
6 Korisničko sučelje

Stvorit ćemo vrlo jednostavno korisničko sučelje koje će se sastojati od polu transparentne slike i teksta. Desnim klikom na *Hierarchy* i odabirom *UI->Panel* automatski smo stvorili panel i *Canvas* na kojem se taj panel nalazi. Svi elementi korisničkog sučelja su *child* objekti *Canvasa*. Promijenit ćemo postavke *Canvas Scalera* kako bi se elementi na ekranu smanjivali ili povećavali ovisno o veličini ekrana. (Slika 6.1)

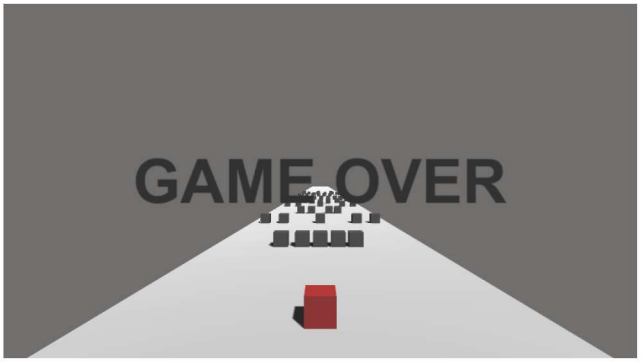


Slika 6.1: Postavke Canvas Scalera

Panelu ćemo dodati UI element Text. Napravit ćemo dva takva panela sa različitim tekstom (Slika 6.2, Slika 6.3)

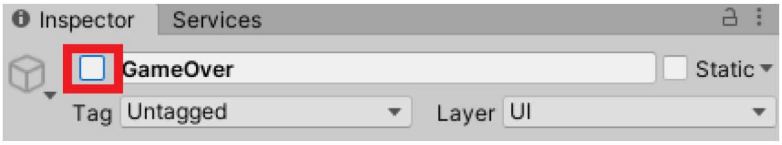


Slika 6.2: Win panel

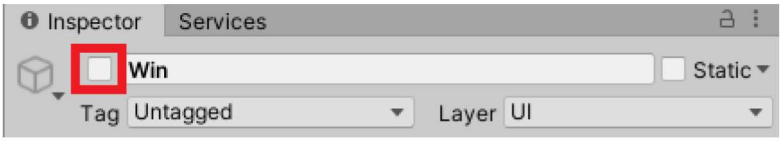


Slika 6.3: Game Over panel

Ostaje nam još deaktivirati panele (Slika 6.4, Slika 6.5) kako ne bi bili prikazani tijekom cijele igre, a aktivirat ćemo ih u određenom trenutku pozivom funkcija koje ćemo definirati u idućem poglavlju.



Slika 6.4: Deaktivacija GameOver panela



Slika 6.5: Deaktivacija Win panela

7 GameManager

Kako bi pojednostavili strukturu koda koristit ćemo jednostavan GameManager čije će se funkcije moći pozivati u drugim skriptama (klasama). GameManager će sadržavati funkcije koje će ponoviti igru (level) u slučaju neuspjeha igrača, aktivirati panele na ekranu ovisno o uspjehu igrača i omogućiti izlaz iz igre pritiskom tipke *Escape*.

Desnim klikom na *Hierarchy* i odabirom *Create Empty*, stvorili smo "prazan" objekt koji nema komponente *Mesh Renderer i Mesh Filter* pa samim time ni vizualnu reprezentaciju u sceni, ali sadrži komponentu *Transform* (objekt se nalazi u sceni, ali ga ne možemo vidjeti). Promijenit ćemo ime u GameManager i dodati skriptu (klasu) istog imena.

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameManager : MonoBehaviour {
    public GameObject winScreen;
    public GameObject gameOverScreen;
    private AudioSource source;
    public void Win()
    {
        winScreen.SetActive(true);
    }
    public void EndScreen()
    {
        gameOverScreen.SetActive(true);
        Invoke("Restart", 1.2f);
    }
    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            Application.Quit();
        }
    }
}
```

U skripti GameManager omogućili smo *namespace UnityEngine.SceneManagement* kako bi dobili pristup klasama i funkcijama s kojima možemo ponoviti igru u slučaju neuspjeha igrača. Funkcija EndScreen() aktivira panel gameOverScreen, a funkcija Invoke poziva funkciju Restart() nakon 1.2 sekundi. Funkcija Restart() će ponoviti igru i vratiti igrača na početnu poziciju. Funkcija Win() će aktivirati panel winScreen u slučaju prelaska ciljne linije. Važno je spomenuti da funkcije moraju biti *public* kako bi ih mogli koristiti u drugim klasama. U *Inspectoru* GameManagera za instance winScreen i gameOverScreen uzet ćemo panele s tekstom koje smo ranije stvorili.

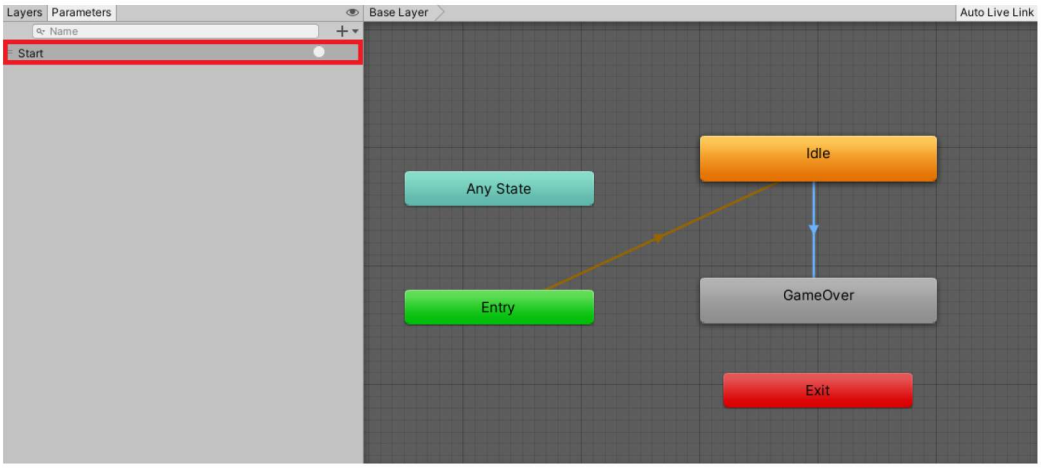
Navedene funkcije ćemo pozivati u klasama Finish pri prijelazu ciljne linije, Obstacle u slučaju sudara igrača i Prepreke te u klasi PlayerMove ako igrač padne s Platforme, za detalje pogledati u [4].

8 Animacija i zvuk

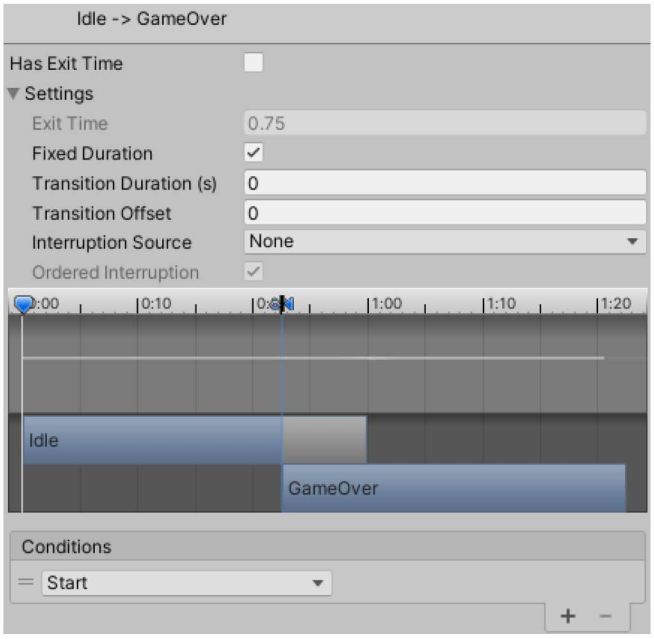
Za animaciju interaktivnog lika koristit ćemo jednu od glavnih grana računalne animacije poznatu kao Keyframing. Keyfreaming je postupak u kojem mi kao animatori odabiremo ključne podatke u određenom vremenu, a računalo samo popuni potrebne podatke između zadanih ključnih podataka. Promjenom ključnih podataka zapravo animirano lik. [2] Kako bi animirali interaktivni lik odabiremo ga u prozoru *Hierarchy* te otvaramo novi prozor *Animation* i pritiskom tipke *Create* dodali smo komponentu *Animator* zajedno sa *Animator Controllerom* i jednim *Animation Clipom* kojeg ćemo nazvati Idle. *Animation Clip* Idle nećemo animirati jer će on predstavljati stanje interaktivnog lika kakvo je na početku igre. Odabirom *Create New Clip* u prozoru *Animation* stvaramo novi *Animation Clip* pod nazivom GameOver kojeg želimo pokrenuti u trenutku sudara sa Preprekom. Za početak ćemo unutar GamaOvera odabrati *key frameove* vezane za veličinu interaktivnog lika (Slika 8.1).

Slika 8.1: Key frames

Prvi *key frame* postavlja veličinu lika na vrijednost 0.75 po svim osima, a drugi na 0.5 po svim osima. Tijekom animacije GameOver naš lik će se postupno smanjivati od normalne veličine 1 do 0.5 u vremenskom periodu od jedne sekunde. Iako smo odabrali samo dva *keyframea* animacija će izgledati "glatko", vrijednosti se neće odjednom promijeniti sa 1 na 0.75 i 0.5 već će računalo preko *Animatora* popuniti vrijednosti između zadanih brojeva. Ostaje nam još definirati prijelaz iz Idle u GameOver animaciju. Otvaramo prozor *Animator* i dodajemo *Trigger* parametar Start (Slika 8.2) zajedno sa prijelazom animacije koje ima postavke kao na slici (Slika 8.3).



Slika 8.2: Parametar Start i shema animacija



Slika 8.3: Postavke prijelaza animacije

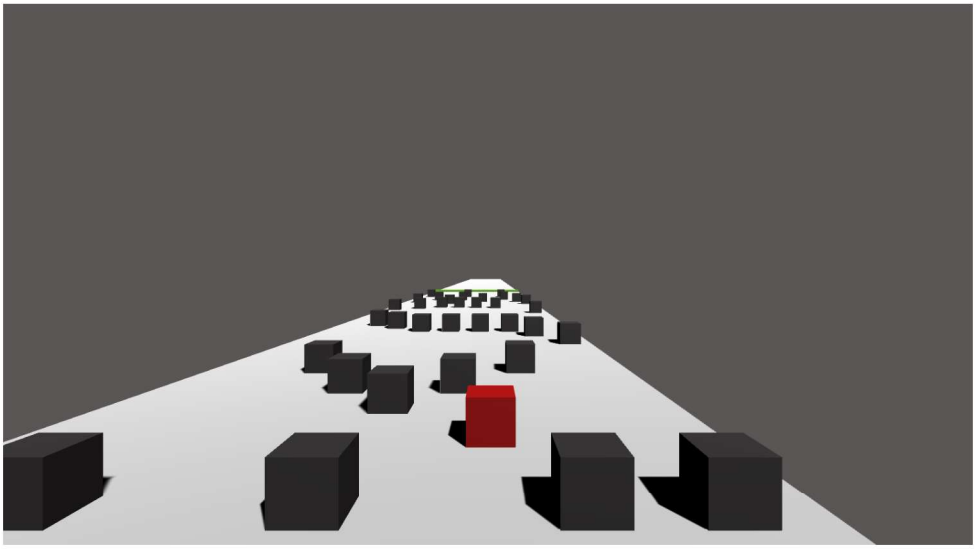
Animaciju GameOver možemo pronaći u prozoru *Project* i klikom na nju pristupamo dodatnim postavkama i isključujemo *Loop Time* opciju kako se animacija ne bi ponavljala nakon završetka jednog ciklusa. Kako bi pokrenuli animaciju GameOver aktivirat ćemo parametar Start u klasi Obstacle u trenutku sudara, detaljnije pogledati u [4].

Kako bi završili interaktivnog lika potrebno je dodati zvučne efekte. Za potrebe ove igre preuzet ćemo besplate zvukove sa Unity Asset Store-a. Također je potrebno dodati komponentu *Audio Source* interaktivnom liku. *Audio Source* će biti izvor našeg zvuka kojeg će primati komponenta *Audio Listener*. *Audio Listener* se po početnim postavkama nalazi na glavnoj kameri i služi kako bi kroz zvučnike reproducirao svaki zaprimljeni zvuk. [3]

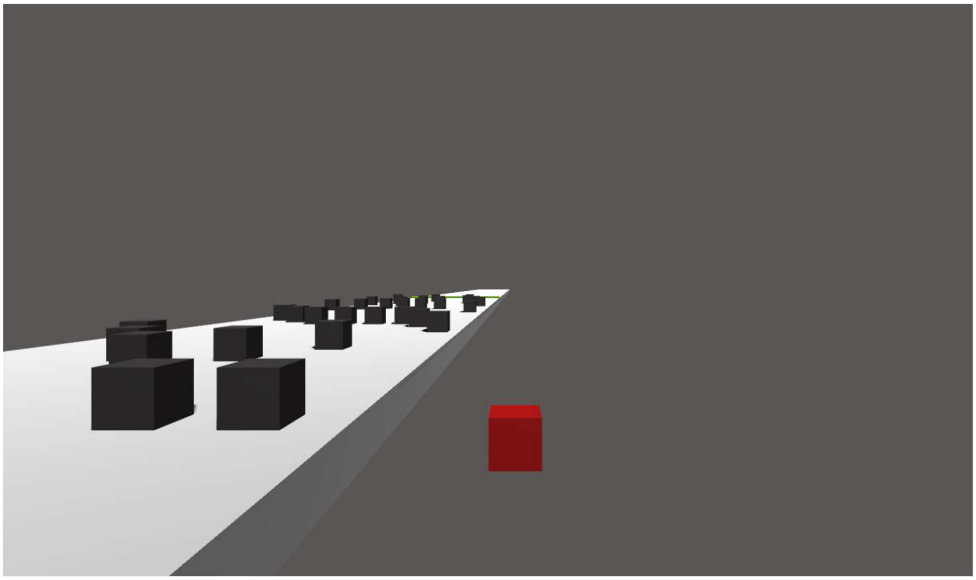
Dodat ćemo dvije *public* funkcije u klasi PlayerMove za pokretanje zvučnih efekata: PlayWin i PlayGameOver. Funkcija PlayWin će se pozvati u klasi Finish prelaskom ciljne linije, a funkcija PlayGameOver u klasi Obstacle prilikom sudara sa Preprekom, detaljnije pogledati u [4].

9 Izvoz igre

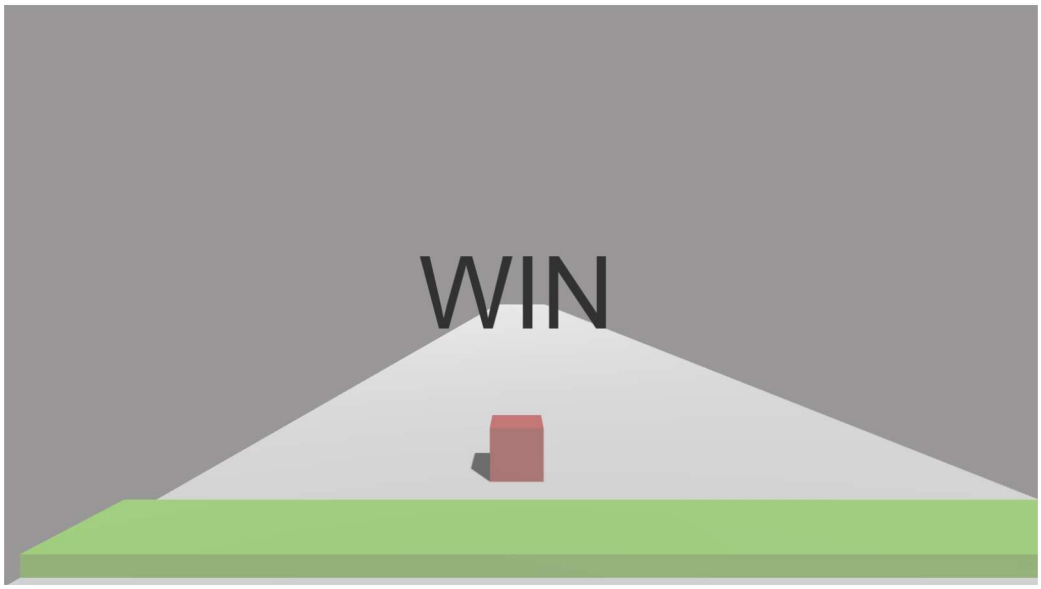
Nakon što smo završili sa postavljanjem scene i svih drugih elementa (interaktivni lik, prepreke, cilj, korisničko sučelje) koji su nam potrebni za gotovu igru, u Unity Editoru provjeravamo simulacijom postoje li neki *bugovi* unutar igre.



Slika 9.1: Izbjegavanje prepreka

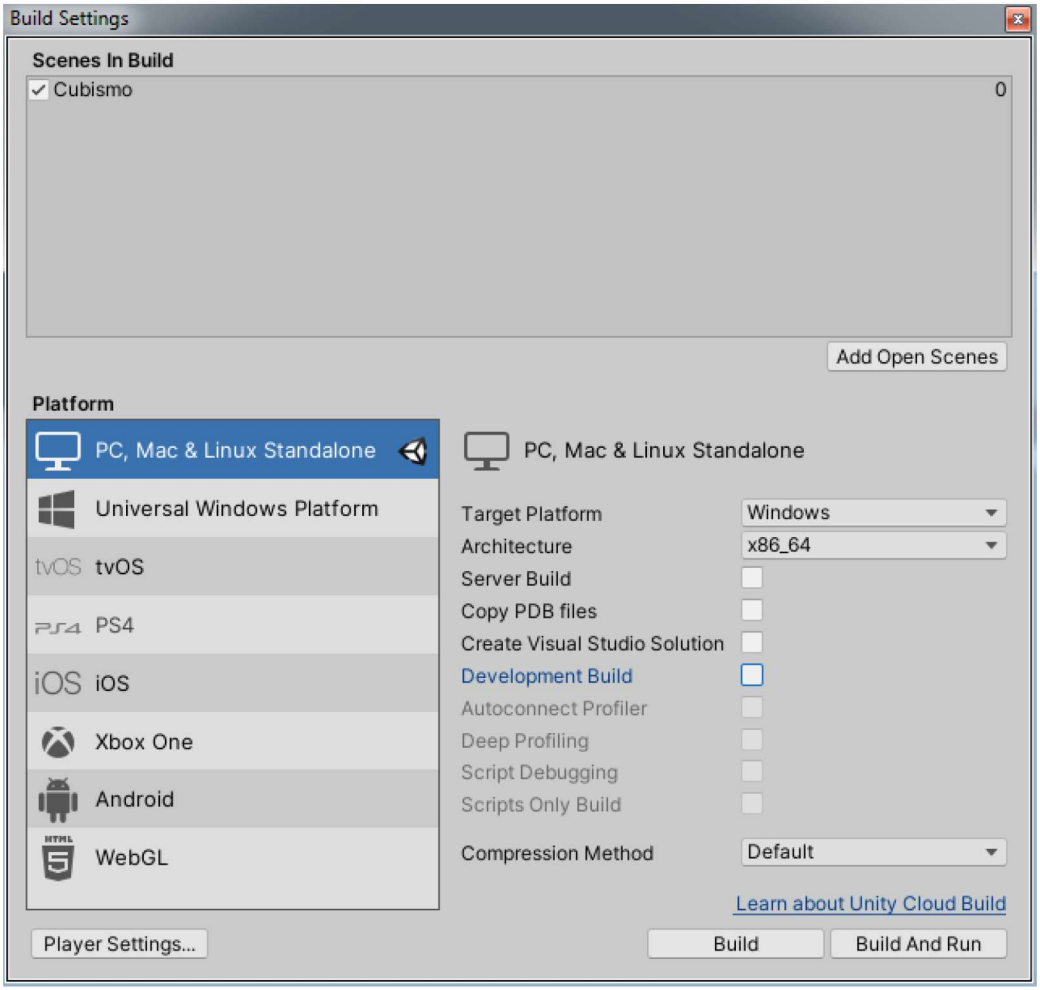


Slika 9.2: Padanje sa Platforme



Slika 9.3: Pobjeda nakon ponovljene igre

Nakon provjere odlazimo na *File->Build Settings* i otvara se novi prozor (Slika 9.4).



Slika 9.4: Postavke eksportiranja

Prvo trebamo dodati scene koje želimo koristiti, a u našem slučaju imamo samo jednu scenu i nju dodajemo klikom na *Add Open Scenes*. Na slici (Slika 9.4) vidimo da možemo izabrati platformu (Windows, Mac, Linux) te 32-bitnu ili 64-bitnu arhitekturu. Ova igra je namjenjena za Windows 64-bitnu arhitekturu, ali se jednostavnim odabirom može eksportirati na neku od drugih ponuđenih platformi ili konzola. Klikom na tipku *Build* Unity Editor će eksportirati igru u željeni folder, a klikom na *.exe* file pokrećemo igru.

10 Zaključak

Kako bi izradili igru koristeći Unity pokretač i Visual Studio potrebno nam je prethodno poznavanje C# jezika, a Unity službena dokumentacija koju smo koristili u ovom radu olakšala nam je svladavanje osnovnih vještina potrebnih za izradu 2D i 3D igara.

U ovom radu izradili smo jednostavnu 3D interaktivnu igru koja nam pokazuje određene prednosti Unity pokretača. Pokazalo se vrlo lako povezati kod i vizualne elemenate igre, te simulirati završni proizvod. Postoji ugrađena podrška za osvjetljenje, animaciju, zvuk i fiziku, to jest sve ono što jedan alat za izradu igara treba imati. Također, jedna od prednosti Unity pokretača je mogućnost jednostavnog eksportiranja igre na različite platforme, uključujući i konzole, Android, iOS operativne sustave i mnoge druge, a to značajno pojednostavljuje i skraćuje vrijeme razvoja igre. Takva fleksibilnost i opsežna službena dokumentacija čini Unity jednim od najboljih alata za izradu svih vrsta igara, kako za početnike tako i za iskusnije korisnike.

Literatura

[1] J. Hocking *Unity in action: Multiplatform game development in C#*

[2] P. Shirley, S.Marschner *Fundamentals of Computer Graphics*

[3] Official Unity documentation version 2019.4

[4] M.Korov, GitHub repository
`https://github.com/mathosmario/zavrsni`