Razvoj 2D igre s dinamičkom pozadinom

Luka, Ljubičić

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Ljubičić

RAZVOJ 2D IGRE S DINAMIČKOM POZADINOM

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Ljubičić

Matični broj: 44034/15-R

Studij: Informacijski sustavi

RAZVOJ 2D IGRE S DINAMIČKOM POZADINOM

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. dr. sc. Radošević Danijel

Varaždin, lipanj 2018.

Luka Ljubičić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

---

Sažetak

Cilj završnog rada je prikazati razvoj 2D igre s dinamičkom pozadinom u Unity okruženju. Prikazati ćemo kako implementirati dinamičku alokaciju objekata i fizikalni model objekata u prostoru. Također u radu će biti prikazana implementacija igre koja će biti predloženi koncept dinamičkog upravljanja pozadinom.

Ključne riječi: Unity; dinamičko upravljanje pozadinom; fizikalni model objekta; c#; tekstualna datoteka; android igra; Windows igra;

Sadržaj

1. Uvod

U ovom radu prikazana je izrada 2D platformne igre za računalo i za mobilne korisnike pod nazivom „Squishy". Glavni lik te igre je dinosaurus koji skače po dinamički generiranim platformama, a uz to ga prati dinamički generirana pozadina. Igra također dealocira generirane objekte kako bi pružala što bolji doživljaj korisniku i imala što bolje performanse. Igra je podržana na Windows i Android operacijskim sustavima. Igra je napravljena u Unity-u, a sve skripte su pisane u C# programskom jeziku. Skripte nam omogućuju funkcionalnost igre, dinamičko ostvarivanje alokacije i dealokacije objekata.

2. Metode i tehnike rada

Kao što je već spomenuto u uvodnom dijelu koristi ćemo *Unity game engine* i *C#* programski jezik. Također u radu će biti korišteni resursi preuzeti sa Interneta kao što su zvukovi, fontovi i slike. U nastavku rada za početak objasniti ćemo okruženje u kojem radimo kako bi nam kasnije stvari bile lakše za shvatit.

3. Unity

Unity je razvojno okruženje koje nam služi za izradu 2D i 3D igara. Unity je okruženje koje je korisnički naklonjeno (*eng.* user frendly), olakšava nam cjelokupnu izradu igre, sve što radimo vidimo u scenama koje nam olakšavaju rad.

3.1. Osnovni dijelovi Unitya

Unity koristi metode kao što su povuci i pusti (*eng.* drag and drop), ima već predefinirane objekte za korisničko sučelje (*eng.* user interface), audio objekte, skripte za kompatibilnost između više platformi (*eng.* cross-platform) i ostalo kao što su događaji (*eng.* events) i slično.

Scena je osnovni prozor u kojem radimo promjene i kad pokrenemo igru ona ima nalik sceni koju uređujemo. Svaka igra može imati više scena tako da ako imamo igru koja ima više nivoa (*eng.* levels) onda koristimo za svaki nivo po jednu scenu kako bi projekt bio jednostavniji i lako

se možemo snaći ukoliko dođe do greške. Također u ovom radu iako imamo jedan nivo imamo dvije scene, jedna scena je zaslužna za izbornik (*eng.* menu) dok je druga scena igra.

Svaka scena se sastoji od jednog ili više objekata koje možemo vidjet u panelu hijerarhija (*eng.* „hierarchy") i oni su nam osnovna komponenta za izradu igre. Svaki objekt ima poziciju, a također može imati komponente kao što su: skripte, događaji, slike, zvuk, animacije itd.
Skripte koje dodajemo objektima služe za upravljanje igrom, daju funkcionalnost igri i daju nam mogućnost za upravljanje objektima u toku igre. Skripte se mogu pisati u programskim jezicima C#, JavaScript. [1]

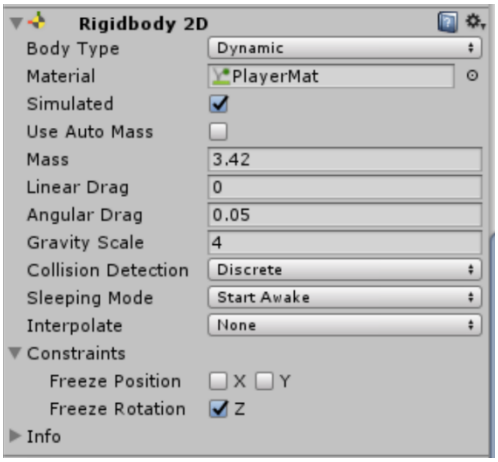
3.2. Komponente Unitya

Kao što je već spomenuto objekti imaju komponente. Navesti ćemo osnovne komponente koje se će biti korištene u ovome radu.

Komponente:

1. Transform
2. Rigidbody 2D
3. Sprite Renderer
4. Animator
5. Script
6. Box, Circle Colider
7. Audio Source

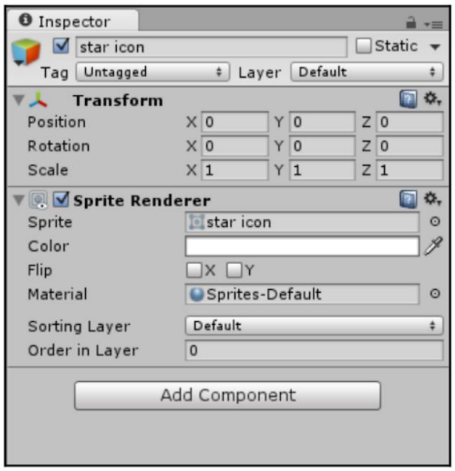
1. Komponenta transform – ima ju svaki objekt i služi nam kako bi odredili rotaciju, veličinu i poziciju po osi apscise i ordinate jer se svaki objekt nalazi negdje u sceni, bio on vidljiv ili ne.
2. Komponenta rigidbody 2D – dodajemo ga objektu kojem je potrebna fizika (u našem slučaju glavnom liku u igri). Bez dodavanja ikakvog koda posjeduje svojstva (*eng.* properties) kao što su masa, gravitacija, i ograničenja „zamrzavanja" neke osi objekta kako se on ne bi rotirao.



Slika 1. *Rigidbody 2D komponenta*

3. Komponenta sprite renderer - služi nam kako bi nekom objektu dodali sliku i ima svojstva kao što su poredak sloja i slično.

Sprite je jednostavna slika koja obično opisuje jedan objekt (na primjer znak) ili (pozadinu). Nekoliko Sprite-ova zajedno čini animaciju. Sprite Renderer je 2D renderer za prikaz sprite-a na zaslonu. [2]



Slika 2. *Sprite renderer objekt [2]*

Slika prikazuje svojstva sprite renderera, možemo postaviti sliku, dodijeliti boju, dodati materijal i time napraviti da naš objekt izgleda kako god mi želimo.

4. Komponenta animator - služi nam za animacije koje smo prethodno kreirali u Animator panelu. Animator panel nam služi kako bi napravili veze između animacija, odnosno omogućuje prelazak iz jednog stanja animacije u drugo stanje. Više o tome biti će rečeno prilikom implementacije animacija.

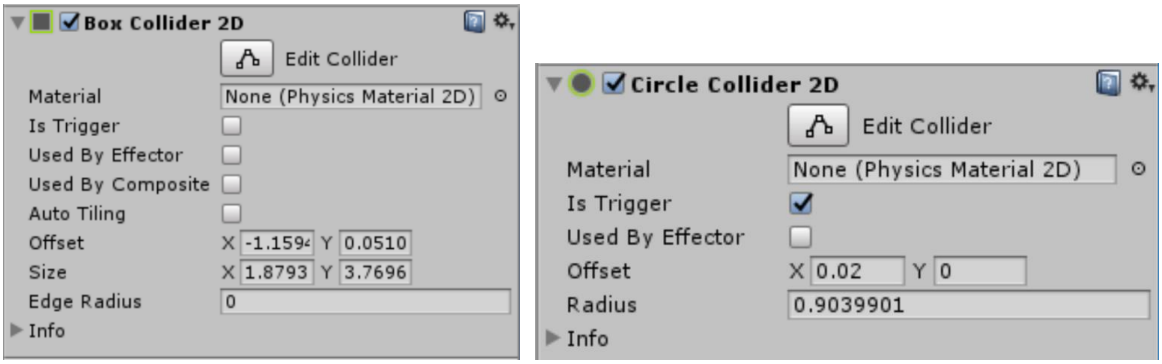
5. Komponenta script – Ako želimo dodati neku funkcionalnost dodajemo skriptu koja je ručno napisana od strane korisnika ili već implementirana u Unity-u. Valja naglasiti da u skripti postoje dvije predefinirane metode koje možemo i ne moramo koristiti.

Metoda Start() nam uglavnom služi za inicijalizaciju vrijednosti u skripti i pokreće se kada je objekt inicijaliziran.

Metoda Update() se izvršava kao beskonačna petlja s time da se izvršava onoliko puta koliko nam računalo može prikazati slika u sekundi.

Također imamo FixedUpdate() koji je sličan kao update ali ne ovisi o brzini računala. Ostale već predefinirane metode skripti objasnit ćemo na primjerima koda.

6. Komponenta box, circle colider – ako smo napravili objekt i dodali mu sprite renderer to ne znači da ako taj objekt sad ima takav oblik on u dodiru s drugim objektom predstavlja fizički objekt. Tom objektu moramo dodati neki colider kako bi on bio ograničenje drugom objektu odnosno dva objekta mogu bit u dodiru samo ako imaju neki collider. Dakle to je komponenta koja nam služi za postavljanje fizičkih ograničenja po nekoj osi.



Slika 3. *Box, Circle Collider 2D komponente*

Collider ima razna svojstva, a najzanimljivije svojstvo mu je „Is Trigger" jer nam omogućava da prepozna da li je došlo do dodira sa ostalim objektom koji ima komponentu collider, prilikom tog dodira objekt sa postavljenim svojstvom na (*true*) ponaša se kao obična slika odnosno nema fizičkog ograničenja.

7. Audio Source komponenta – služi nam za dodavanje zvukova u igri, dodajemo ju nekom GameObjectu kojeg kasnije možemo kontrolirati pomoću skripte i iz njega uzimamo komponentu tipa AudioSource te ju kontroliramo jednostavnim metodama Play, Pause i Stop. [3]

3.3. Prednosti i nedostaci Unitya

Unity kao i svaki drugi program ima prednosti i nedostataka, navesti ćemo neke prednosti i nedostatke.

Prednosti

- Unity je najbolja aplikacija za razvoj igara
- učinkovit za prikazivanje 2D scena, podržava i 3D
- moguće je izdati verziju na više platformi
- nije kompliciran

Nedostaci

- dokumentacija je zastarjela
- dosta dodataka renderiranja i slično košta puno
- ažuriranje programa može dovesti do grešaka

Unity je dobar i jednostavan alat za izradu manjih i srednjih igara ako mislite proizvesti nešto veće trebali bi se odlučiti možda za Unreal Engine ili Lumberyard.

3.4. Dodaci

Dodaci odnosno eng. Assets glavna je mapa u svakom projektu te sadrži sve datoteke i mape koje ćemo koristiti u projektu. Projekt radi jednostavnosti i preglednosti možemo strukturirati u nekoliko osnovnih mapa. Mape koje su osnovne za izradu ove igrice su: Animations, Pictures, Resources, Scenes, Scripts, Standard Assets.

Animations mapa sadrži sve animacije glavnog lika koje smo kreirali na temelju niza slika. Pictures mapa sadrži sve slike koje su korištene pri izradi animacija. Resources mapa sadrži sve ostale resurse koji su korišteni kao što su pozadina, zvukovi, slike koje su korištene pri izradi objekata, font i sl. Scenes mapa sadrži sve scene koje su korištene projektu, na primjer scena za glavni izbornik, scena za nivo i scena glavnog lika. Standard Assets je mapa koju Unity pruža kako bi nam olakšao izradu projekta za različite platforme (koristili smo CrossPlatformInput). Scripts mapa sadrži sve skripte projekta. U projektu svi dodaci multimedije nisu samostalno kreirani već su preuzeti sa Interneta, a sva implementacija skripti, animacija i ostalo napravljena je samostalno.

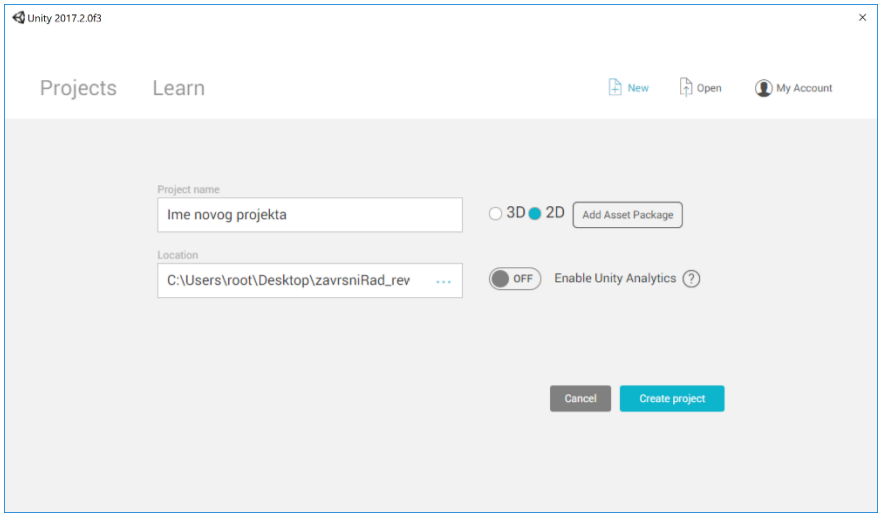
4. Implementacija

U nastavku prikazati ćemo implementaciju svake scene, njihovo povezivanje, kreiranje objekta i njihovih komponenti, suradnju skripti i komponenti, implementaciju ponavljajuće pozadine, dinamičko alociranje i dealociranje objekata, zapis rezultata igre i kreiranje postavki koje korisnik može podesiti.

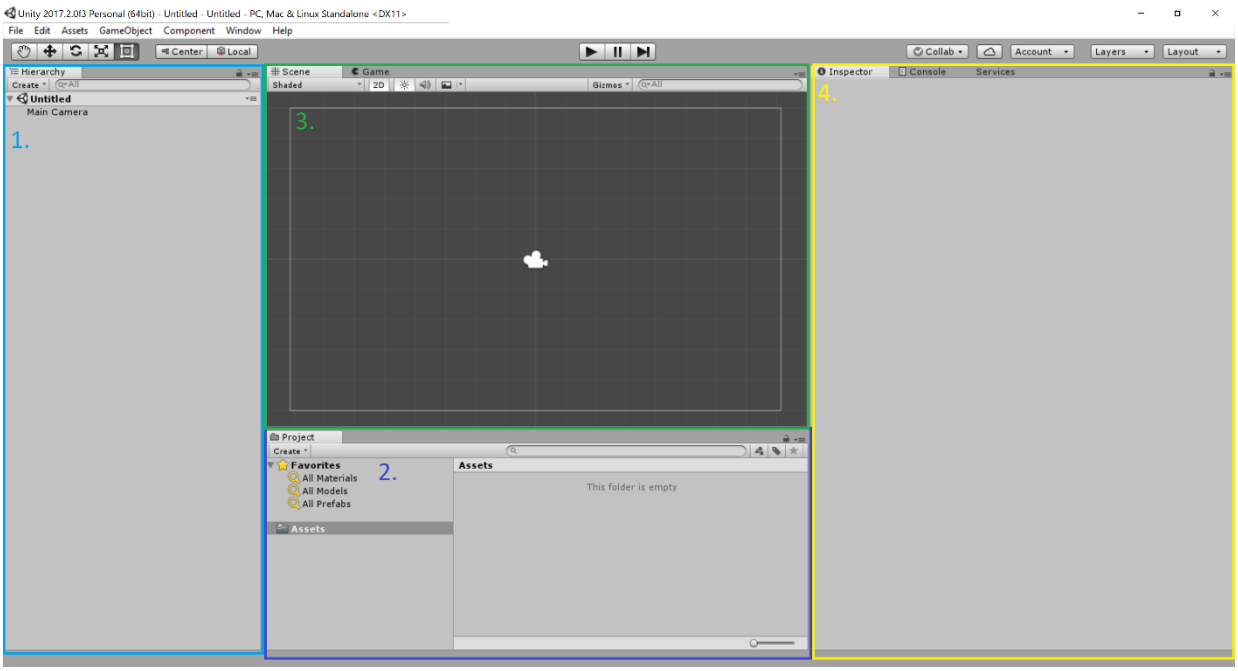
4.1. Scene i njihovo povezivanje

„Scene sadrže okruženja i izbornike igre. Zamislite svaku jedinstvenu datoteku scena kao jedinstvenu razinu. U svakoj sceni stavljate svoje okruženje, prepreke i dizajn, u osnovi projektirate i izgrađujete svoju igru u komadiće.” [3]

Za početak prikazat ćemo izradu jednostavnije scene, a to je scena glavni izbornik (*eng.* menu). Otvaramo Unity i kreiramo novi projekt, odaberemo naziv, tip projekta i putanju do direktorija gdje želimo da nam se projekt spremi.



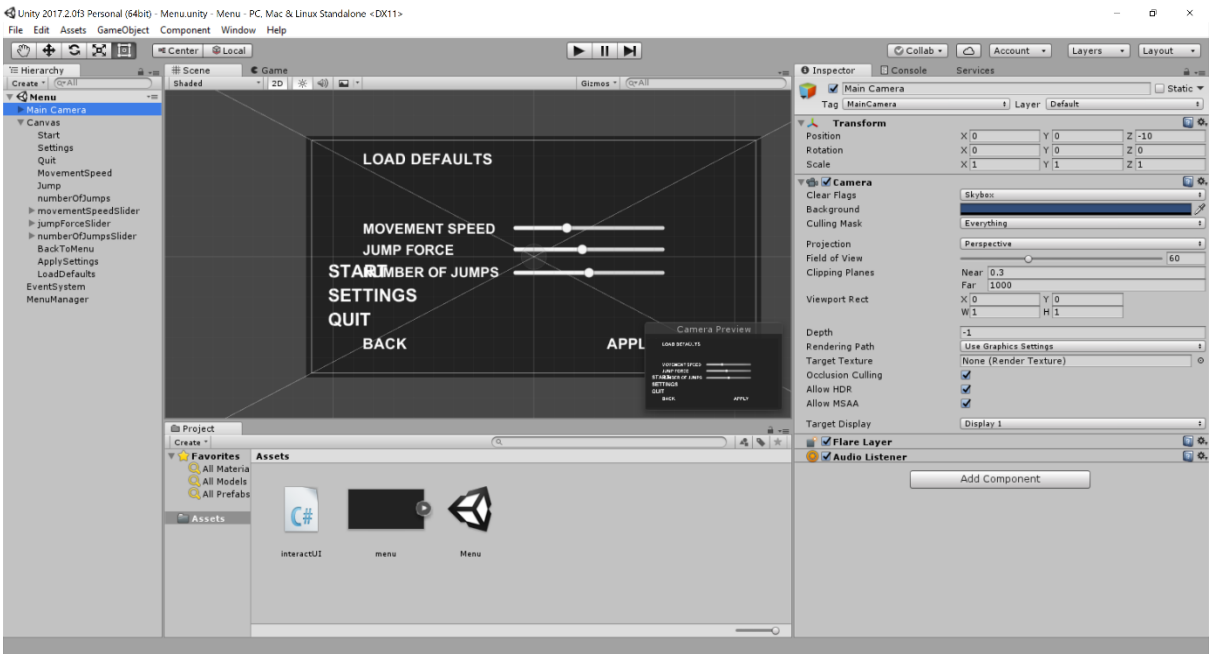
Slika 4. *Unity kreiranje novog projekta*



Slika 5. *Početni zaslon projekta*

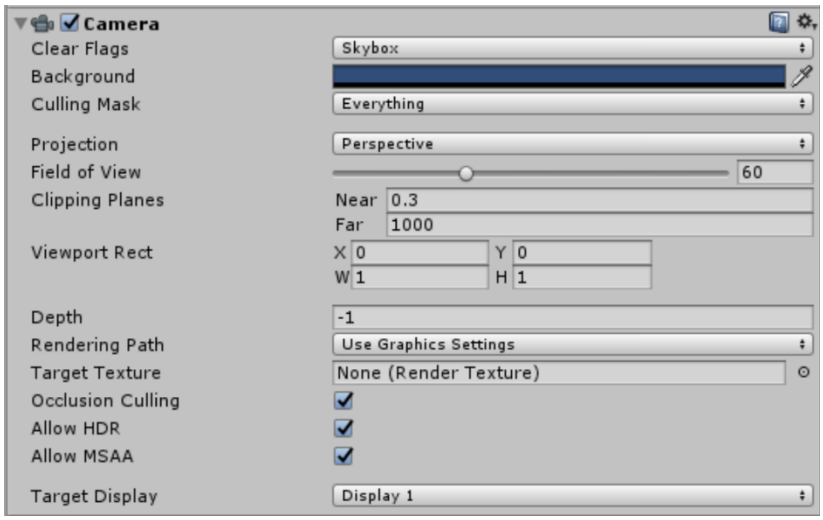
Na slici je prikazan početni zaslon kreiranog projekta. Broj 1 Hierarchy panel prikazuje nam sve objekte koje smo kreirali, Unity je kreirao objekt Main Camera koji nam prikazuje scenu i igru. Broj 2 prikazuje panel Project u kojem imamo mapu Assets u kojoj se nalaze sve datoteke koje su nam potrebne. Broj 3 prikazuje nam scenu onako kako sad izgleda i tu radimo sve vizualne promjene. Broj 4 nam prikazuje „Inspector“ panel kojeg ćemo koristiti prilikom rada sa objektima, prikazuje nam komponente objekta i njihova svojstva.

U mapu „Assets“ dodajemo potrebne datoteke, postoji nekoliko načina za dodavanje na primjer pritiskom desne tipke miša i navigiranjem i odabirom datoteke. Mi ćemo koristit jednostavniji način i povući i ispustiti datoteke u mapu.



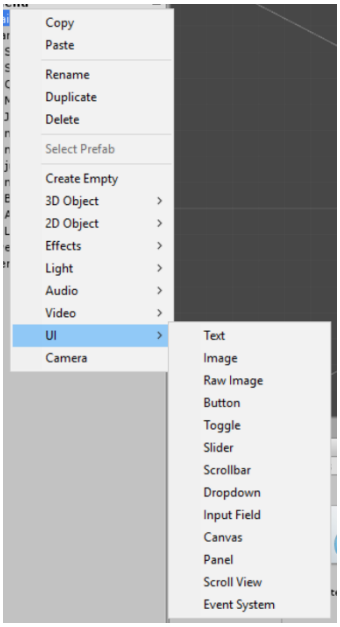
Slika 6. *Završni prikaz scene menu*

U scenu smo dodali sliku naziva „menu”, sliku povučemo i ispustimo u scenu. Namjestimo veličinu slike tako da je ona nešto veća ili jednaka okviru kamere. Također radi ljepšeg prikaza možemo namjestiti svojstvo kamere Projection na Perspective.



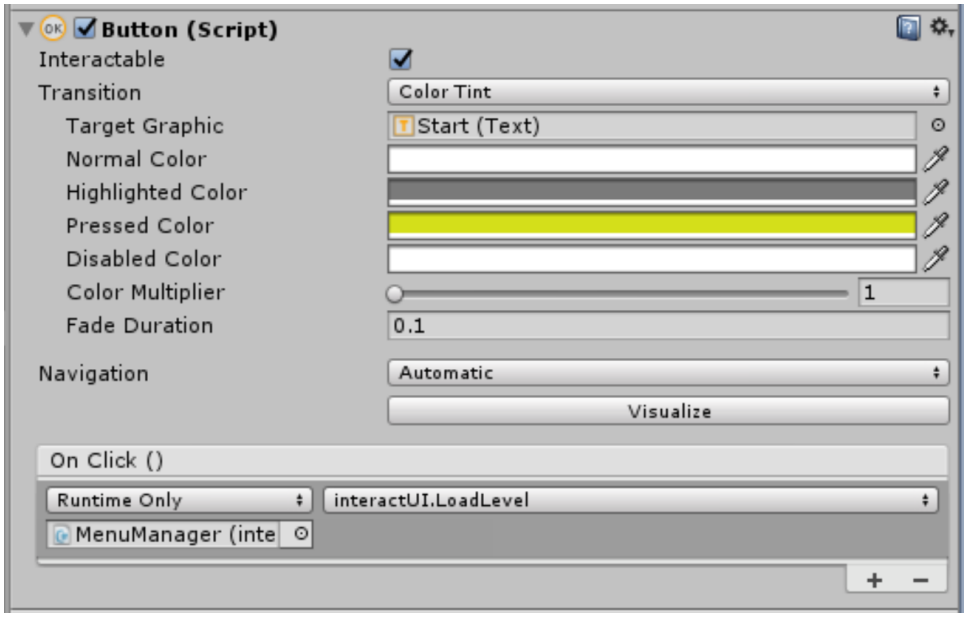
Slika 7. *Svojstva kamere*

Na slici koja prikazuje završni prikaz scene menu, u panelu „Hierarchy“ možemo vidjeti objekt Canvas UI (*eng.* user interface) objekti.



Slika 8. *Dodavanje UI objekta*

Desnim klikom miša u panel „Hierarchy“ dodajemo UI objekte i napravimo scenu onakvom kakvom želimo. Dodajemo UI objekte tipa button i u „Inspector“ panelu mijenjamo njihov naziv i komponente. Svaki UI objekt tipa button također ima komponentu Button sa raznim svojstvima. Glavna komponenta buttona ima i događaj (*eng.* event, On Click). Taj događaj nam omogućava radnju prilikom klika na njega. Dodajemo mu skriptu koja će pokrenuti danu radnju odnosno metodu prilikom interakcije.



Slika 9. *Svojstva komponente button*

Ispustimo skriptu u događaj On Click i odaberemo metodu. Kada smo dodali skriptu vidimo njen naziv, i postavimo metodu koja će se izvršiti u interakciji u ovom slučaju metoda se zove LoadLevel.

Sada ćemo prikazati skriptu za menu, ova skripta će biti cijela opisana i neke ćemo još tako prikazat ostale će biti samo opisane.

```
public class interactUI : MonoBehaviour
{
    private GameObject txtStart;
    private GameObject txtQuit;
    private GameObject txtSettings;

    private GameObject txtMovementSpeed;
    private GameObject txtJump;
    private GameObject txtNumberOfJumps;

    private GameObject movementSpeedSlider;
    private GameObject jumpForceSlider;
    private GameObject numberOfJumpsSlider;
    private GameObject loadDefaults;

    private Slider movementSpeed;
    private Slider jumpForce;
    private Slider numberOfJumps;

    private GameObject backToMenu;
    private GameObject applySettings;

    void Start()
    {
        txtStart = GameObject.Find("Start");
        txtSettings = GameObject.Find("Settings");
        txtQuit = GameObject.Find("Quit");

        txtMovementSpeed = GameObject.Find("MovementSpeed");
        txtJump = GameObject.Find("Jump");
        txtNumberOfJumps = GameObject.Find("numberOfJumps");

        movementSpeedSlider = GameObject.Find("movementSpeedSlider");
        jumpForceSlider = GameObject.Find("jumpForceSlider");
        numberOfJumpsSlider = GameObject.Find("numberOfJumpsSlider");

        backToMenu = GameObject.Find("BackToMenu");
        applySettings = GameObject.Find("ApplySettings");
        loadDefaults = GameObject.Find("LoadDefaults");

        movementSpeed = movementSpeedSlider.GetComponent<Slider>();
        jumpForce = jumpForceSlider.GetComponent<Slider>();
        numberOfJumps = numberOfJumpsSlider.GetComponent<Slider>();

        movementSpeed.value = PlayerPrefs.GetFloat("movementSpeed", 10);
        jumpForce.value = PlayerPrefs.GetFloat("jumpForce", 18.28f);
        numberOfJumps.value = PlayerPrefs.GetInt("numberOfJumps", 2);
        GetSettings();
        ApplySettings();
        HideSettings();
    }
```

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

public void LoadSettings()
{
    HideMenuUI();
    SettingsTab();
}

public void ApplySettings()
{
    PlayerPrefs.SetFloat("movementSpeed", movementSpeed.value);
    PlayerPrefs.SetFloat("jumpForce", jumpForce.value);
    PlayerPrefs.SetInt("numberOfJumps", (int) numberOfJumps.value);
}

public void SettingsTab()
{
    movementSpeed.value = PlayerPrefs.GetFloat("movementSpeed");
    jumpForce.value = PlayerPrefs.GetFloat("jumpForce");
    numberOfJumps.value = PlayerPrefs.GetInt("numberOfJumps");

    txtMovementSpeed.active = true;
    txtJump.active = true;
    txtNumberOfJumps.active = true;
    movementSpeedSlider.active = true;
    jumpForceSlider.active = true;
    numberOfJumpsSlider.active = true;
    backToMenu.active = true;
    applySettings.active = true;
    loadDefaults.active = true;
}

public void HideSettings()
{
    txtMovementSpeed.active = false;
    txtJump.active = false;
    txtNumberOfJumps.active = false;
    movementSpeedSlider.active = false;
    jumpForceSlider.active = false;
    numberOfJumpsSlider.active = false;
    backToMenu.active = false;
    applySettings.active = false;
    loadDefaults.active = false;
}

public void LoadDefaults()
{
    PlayerPrefs.SetFloat("movementSpeed", 10);
    PlayerPrefs.SetFloat("jumpForce", 18.28f);
    PlayerPrefs.SetInt("numberOfJumps", 2);

    movementSpeed.value = PlayerPrefs.GetFloat("movementSpeed");
    jumpForce.value = PlayerPrefs.GetFloat("jumpForce");
    numberOfJumps.value = PlayerPrefs.GetInt("numberOfJumps");
    HideSettings();
    ShowMenuUI();
}
```

```
    public void HideMenuUI()
    {
        txtStart.active = false;
        txtSettings.active = false;
        txtQuit.active = false;
    }

    public void ShowMenuUI()
    {
        HideSettings();
        txtStart.active = true;
        txtSettings.active = true;
        txtQuit.active = true;
    }

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

Kao što smo spomenuli Unity nam kreira metode Start i Update. Update nam ovdje nije potreban jer ne trebamo kod koji se uporno izvodi. Na vrhu skripte vidimo atribute (*eng.* fields). Svaki objekt koji se nalazi u „Hierarchy“ panelu je tipa GameObject klase. Vidljivost atributa je private što znači da ostale skripte ne vide ove atribute. U metodi Start inicijaliziramo te atribute tako da možemo radit s njima. Također vidimo statičku klasu PlayerPrefabs koja nam služi za dohvaćanje ili postavljanje neke vrijednosti u programu i u ovom slučaju nam služi za dohvaćanje odnosno postavljanje vrijednosti u igri, kao što je brzina micanja glavnog lika i slično. Kako ne bi za postavke radili novu skriptu dohvaćamo UI objekte i postavljamo njihovu vidljivost na *false*. Ako korisnik klikne na Settings pozivaju se metode HideMenuUI i SettingsTab. HideMenuUI skriva trenutni izbornik, a SettingsTab postavlja vidljivost UI elemenata na true.
Metoda CloseGame zatvara aplikaciju, dok metoda LoadLevel pomoću SceneManager klase i njezine metode LoadScene koja prima parametar naziva scene tipa *string* pokreće drugu scenu koju prije izvoza postavljamo u *build settings*-ima.

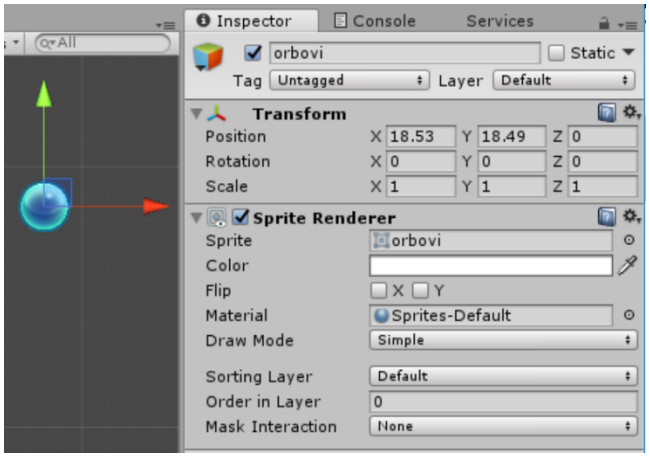
4.2. Objekti i komponente

GameObject je najvažniji koncept u Unity-u. Svaki objekt u igri je GameObject na primjer glavni lik, svjetla, kamera i slično. GameObject ne može ništa učiniti sam po sebi, moramo mu postaviti svojstva prije nego što postane glavni lik, okruženje ili poseban efekt..

GameObject-u možemo dodati komponente ovisno o onome što želimo da on postane, bio to glavni lik, stablo, zvuk, svjetlo i slično. [3]

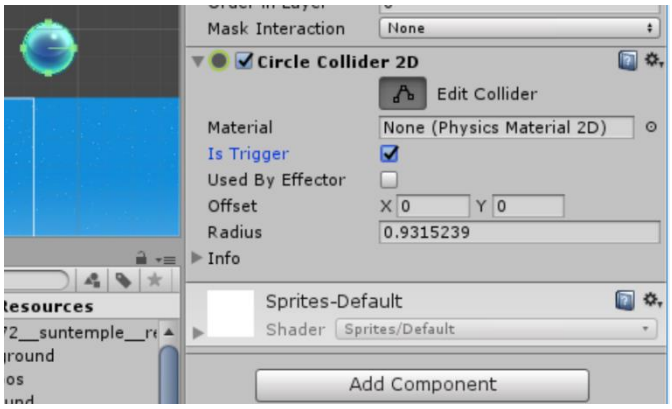
4.2.1. Povezivanje objekta, komponenti i C# skripti

Sada ćemo prikazati kako možemo kreirati objekt, dodati mu komponentu i skriptu. Za početak dodajmo sliku u našu Assets mapu, nakon toga povučemo i ispustimo sliku u scenu.



Slika 10. *Stvaranje objekta*

Ako smo na ovaj način dodali objekt, vidimo da mu se dodala i komponenta Sprite Renderer i za sliku (Sprite) je postavljena slika koju smo povukli. Ova „kugla" u igri će predstavljat stvar za skupljanje. Pošto moramo znati je li glavni lik došao u doticaj s loptom moramo dodati još jednu komponentu, a to je Circle collider. Da bi dodali komponentu kliknemo na Add Component i izaberemo komponentu koju želimo dodati.



Slika 11. *Dodavanje komponente circle collider*

Kao što vidimo također „Is Trigger" svojstvo smo postavili na true. Dok objekt ima tu komponentu i to svojstvo postavljeno na *true*, ta komponenta se ponaša drukčije, više ne predstavlja ograničenje

drugog objekta da prođe kroz njega, nego nam služi da provjerimo je li drugi objekt koji ima komponentu collider u doticaju s tom komponentom. Tu provjeru možemo izvršiti putem koda. Desnom tipkom miša kliknemo u projektnom panelu u mapi Assets te dodajemo C# script. Nazvati ćemo ju „OrbCollected" i povučemo tu skriptu u Inspector panel, sada kugla ima tu skriptu dodanu.

```
public class OrbCollected : MonoBehaviour {
    public AudioSource pickUpSound;

    void OnTriggerEnter2D(Collider2D other){
        if (other.gameObject.tag == "Player" && gameObject.tag.Equals("orb")) {
            Color orbColor = gameObject.GetComponent<Renderer>().material.color;
            if (orbColor == Color.magenta)
            {
                ScoreChange.scoreValue += 2;
            }
            else if (gameObject.GetComponent<SpriteRenderer>().sprite.name.Equals("triangle"))
            {
                ScoreChange.scoreValue += 20;
            }
            else if(orbColor == Color.white)
            {
                ScoreChange.scoreValue += 5;
            }

            gameObject.SetActive (false);
        }
        pickUpSound.Play ();
    }
}
```

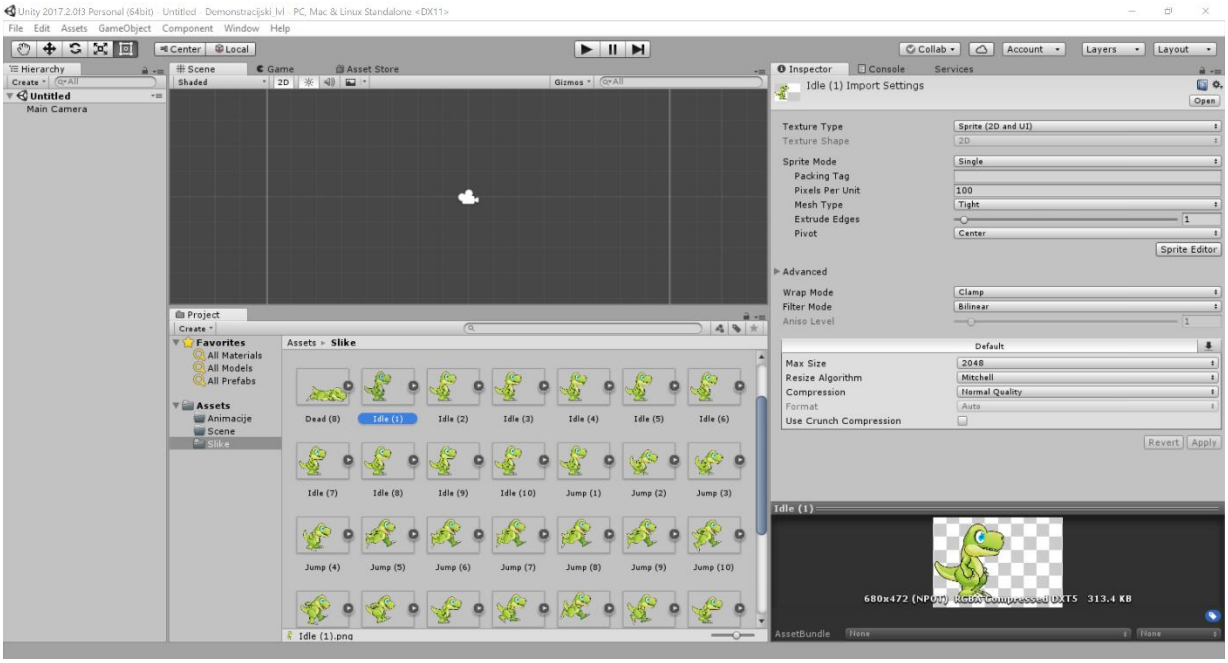
U skripti vidimo da imamo atribut AudioSource vidljivosti public radi jednostavnosti jer sad na tu skriptu možemo povući GameObject koji ima komponentu AudioSource te ju možemo upotrijebit. Vidimo metodu OnTriggerEnter2D i ima parametar tipa Collider2D. Ta metoda je zaslužna za provjeru dodira dva objekta, parametar kojeg prima mi smo ga nazvali „other" je svaki drugi objekt koji ima neki collider na sebi. Obavljamo provjeru da li je oznaka (*eng.* tag) drugog objekta „Player" i kasnije upravljamo rezultatom igre pomoću nove klase „ScoreChange" koja ima statički atribut scoreValue te ga uvećavamo svaki put kada je došlo do doticaja igrača i orba. Također budući da je skripta stavljena direktno na objekt orb pomoću linije *gameObject.SetActive(false);* orb nestane kako nebi korisnik mogao isti skupiti nekoliko puta. Dok *pickUpSound.Play();* samo pusti zvuk kada je došlo do tog doticaja.

4.3. Kreiranje igraćeg lika

Za kreiranje igraćeg lika potrebno je kreirati skriptu koja nam služi za kretanje lika, ograničenja (box colider i circle colider) i niz slika odnosno spriteova kako bi kreirali animaciju.

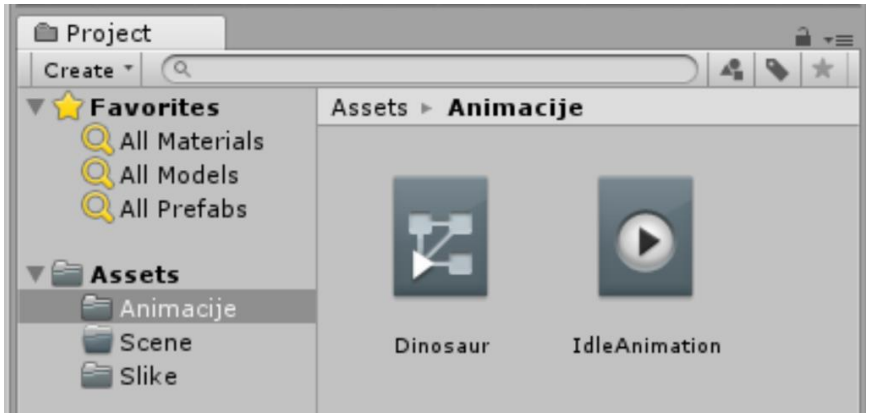
4.3.1. Kreiranje animacije

Potrebno je dodati niz slika u jednu mapu koje ćemo koristiti za kreiranje animacije (vidi slika xy).



Slika 12. *Slike potrebne za kreiranje animacija*

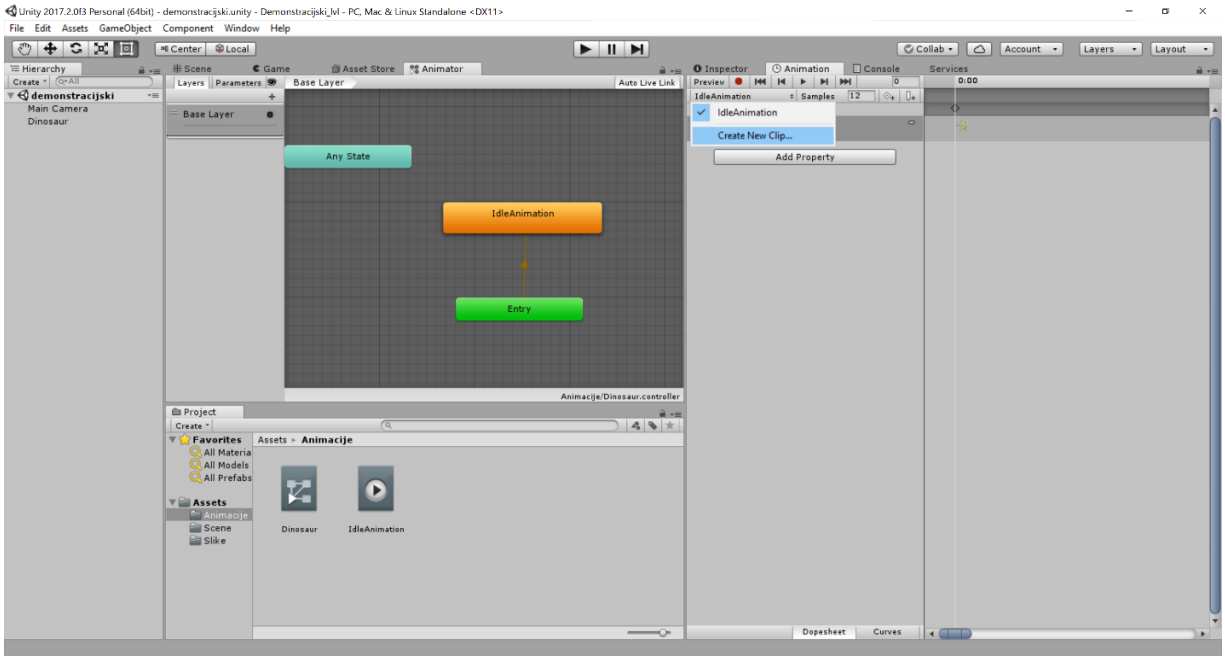
Na slici vidimo da se u mapi „Slike“ nalaze sve slike koje će nam biti potrebne prilikom kreiranja animacija. Prva animacija nam je „Idle“ koja nam prikazuje igraćeg lika u stajaćem položaju, potrebno je označiti sve „Idle“ slike i povući ih u dio scene. Unity će generirati animaciju za nas i pitat gdje ju želimo spremit, odaberemo mapu u ovom slučaju animacije Unity nam generira dvije datoteke, kontroler animacija (*slika 13* lijevo) i animaciju (*slika 13* desno).



Slika 13. *Kontroler animacija i animacija*

Pritiskom na gumb play vidimo da je animacija uspješno napravljena. Kontroler animacija nam omogućava promjenu stanja animacija.

Sada dodajemo drugu animaciju na način da označimo u glavnom panelu te otvaramo „Animation“ panel i pritisnemo „Create New Clip...“, dodijelimo ime i dodajemo mu naziv. [4]

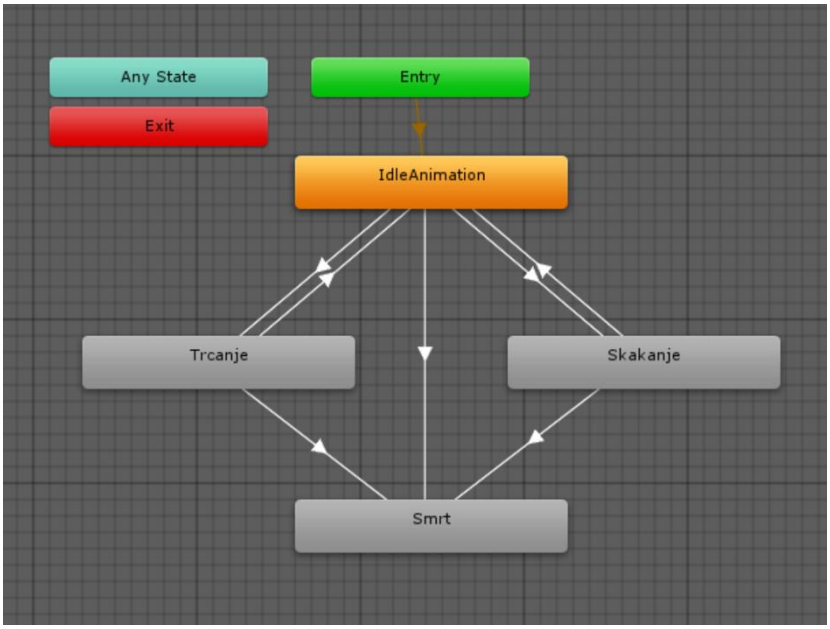


Slika 14. *Animator panel*

Na slici u panelu „Animator“ možemo vidjeti da imamo zeleni i narančasti kvadrat, zeleni služi za inicijalizaciju, a svaki koji je narančasti je standardna animacija. Ako smo dodali novu animaciju sada povlačimo nove slike za drugu animaciju u animation panelu. Ponavljamo ovaj korak onoliko puta koliko stanja želimo da naš igrači lik ima. Novokreirana animacija u „Animator“ panelu je sive boje, što znači da nije aktivna, a korisnikova akcija aktivira pojedinu animaciju. Kako bi mogli mijenjati stanja između animacija potrebno je napraviti skriptu koja prilikom pritiska tipke mijenja animaciju u onu koju želimo.

4.3.2. Kreiranje skripte za kretanje i promjenu animacije

Prije nego što krenemo sa pisanjem skripte moramo napraviti veze „Transitions“ koje nam služe za promjenu stanja animacija odnosno prikazuju iz kojeg stanja je moguće doći u drugo stanje, u mojem slučaju to izgleda ovako:



Slika 15. *Promjena stanja animacije*

U animator panelu u kartici „Parameters“ dodati ćemo jednu varijablu po imenu „State“ i postavimo ju na neku vrijednost. Sada svakoj vezi dodajemo uvjet, u nakon klika na pojedinu vezu u inspector panelu pod kategorijom „Conditions“ dodajemo tu našu novokreiranu varijablu i dajemo joj uvjet. U ovom slučaju uspoređivati ćemo da li je varijabla jednaka nekom broju. Ako promijenimo broj varijable mijenja nam se i animacija igraćeg lika. Vrijednost varijable „State“ za stanje mirovanja (IdleAnimation) odabrao sam broj nula, za trčanje jedan, za skakanje dva i za smrt tri.

U nastavku slijedi implementacija skripti koja će nam služiti za promjenu stanja animacije i kretanja igrača.

4.3.3. Skripte za igraćeg lika

Prvo na objekt igrača dodajemo oznaku (*eng.* tag) „*Player*“, zatim napravimo skriptu u mapi skripte (mapa u koju ćemo spremati sve skripte), nazvat ćemo ju „*PlayerManager*“. Skripta provjerava korisnikova pritiska određene tipke kako bi mogli promijeniti stanje animacije.

Na objekt igrača dodajemo skriptu na način da ga označimo i u Inspector panelu dodamo skriptu (možemo ju povući i ispustit ili preko gumba „*Add Component*“).

Igraču dodajemo i komponentu „Rigidbody 2D“ kojoj ćemo pristupati preko skripte. Radi jednostavnosti dodati ćemo još jednu skriptu koja će nam služit za kretanje igrača („*PlayerMovement*“).

Da ne prikazujemo cijeli kod, kod skripte *PlayerManager* imamo jedan atribut tipa Animator i njega inicijaliziramo u Start metodi jednostavnom linijom koda *GetComponent<Animator>();* jer objekt glavnog lika ima komponentu tipa Animator i program zna da se radi o toj komponenti. *Update* metoda nam provjerava pritisak tipke korisnika, i provjeravamo za svaku tipku je li stisnuta linijom koda na primjer *Input.GetKeyDown(KeyCode.UpArrow);* provjerava da li je pritisnuta strelica za gore, ako je postavljamo varijablu „State" koju smo prije spomenuli na neku vrijednost (kako bi došlo do promjena stanja animacije). Za skok odredili smo broj 2 pa tako samo napišemo *animator.SetInteger("State", 2);* i ako korisnik pritisne tipku za gore dolazi do puštanja animacije skoka, tako napravimo za ostala stanja. U slučaju da je igrač izgubio puštamo animaciju smrti.
Skripta *PlayerMovement* nam služi kako bi pokretali igrača po osima.

```
void Update()
{
    float horizontal = Input.GetAxis("Horizontal");
    HandleJump();
    HandleMovement(horizontal);
    Flip(horizontal);
}

private void HandleMovement(float horizontal)
{
    rigidbody.velocity = new Vector2(horizontal * movementSpeed, rigidbody.velocity.y);
}
```

Input.GetAxis("Horizontal"); je postavljen za pritisak lijeve i desne tipke ili pritiska slova „A" ili „D", vrijednost joj varira od –1 do 1. Ako pritisnemo tipku poziva se metoda koja pomiče lika po apscisi brzinom vrijednosti *movementSpeed*.

```
void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag.Equals("Ground"))
    {
        isGrounded = true;
        numberOfJumps = prefabNumberOfJumps;
    }
}



public void HandleJump()
{
    if (Input.GetKeyDown(KeyCode.UpArrow) || Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.Space))
    {
        Jump();
    }
}
```

```
public void Jump()
{
    if (numberOfJumps > 0 || isGrounded)
    {
        isGrounded = false;
        rigidbody.velocity = new Vector2(rigidbody.velocity.x, jumpForce);
        numberOfJumps--;
        jumpAudioSource.Play();
    }
}
```

Metoda *Jump* služi za pomicanje igraćeg lika prema gore, provjeravamo da li je igrači lik na podu, ili je broj skokova veći od 0 ako je onda igrači lik skače.

Zadnja metoda koju imamo je metoda *Flip* koja nam služi da se igraći lik okrene ako ide u suprotnom smjeru, a implementacija je prilično jednostavna, u startu postavljamo facingRight na true, i samo mijenjamo vrijednost te varijable, a transform komponentu samo po x osi pomnožimo sa –1.

```
private void Flip(float horizontal)
{
    if (horizontal > 0 && !facingRight || horizontal < 0 && facingRight)
    {
        facingRight = !facingRight;

        Vector2 direction = transform.localScale;
        direction.x *= -1;

        transform.localScale = direction;
    }
}
```

4.4. Implementacija ponavljajuće pozadine

Kako je ova igrica dinamička i nivo nije konstruiran u cijelosti nego se uporno generira mi ne možemo znati koliko će korisnik igrati ovu igricu, beskonačna je. Pa tako ako znamo da se kamera u svakom slučaju pomiče lijevo ili desno i imamo pozadinu koja stoji ćemo u nekom trenutku izaći iz pozadine i pozadine više neće biti.

Kameru smo implementirali tako da se kamera pomiče prema desno cijelo vrijeme, za to je zaslužna sljedeća skripta (*CameraController*).

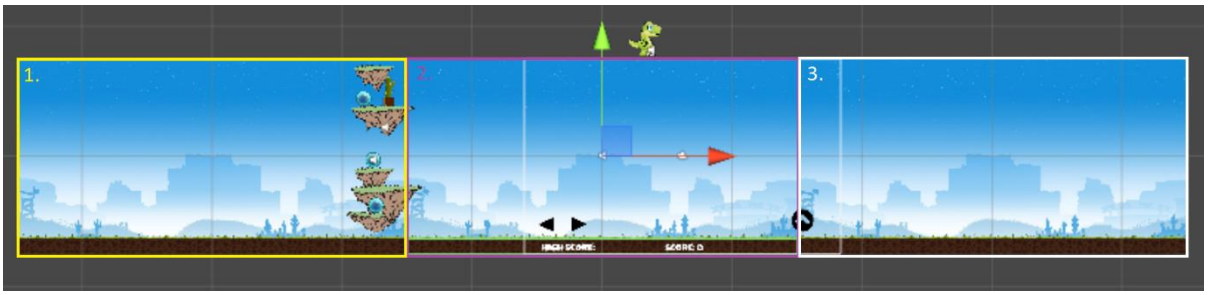
```
public class CameraController : MonoBehaviour
{
    public float panSpeed;
    public Vector3 cameraPossition;
    private bool speedIsChanged;
    private float score;

    void Update()
    {
        score = ScoreChange.scoreValue;

        if (score > 0 && score < 40 && score % 10 == 0 && !speedIsChanged)
        {
            panSpeed = panSpeed + (panSpeed * 0.221f);
            speedIsChanged = true;
        }
        else if (score % 10 != 0)
        {
            speedIsChanged = false;
        }
        cameraPossition = transform.position;
        cameraPossition.x += panSpeed * Time.deltaTime;
        transform.position = cameraPossition;
    }
}
```

Ova skripta pomiče kameru prema desno s time da njena brzina ovisi o rezultatu igre. Uz malo testiranja odredili smo proizvoljne brojeve pomicanja kamere kako igra ne bi bila preteška ni prelagana. Glavni dio koda koji je zaslužan za micanje kamere je *transform.position* koji postavlja poziciju kamere na novu vrijednost u metodi *Update*.
Sada ako pokrenemo igru samo sa jednom pozadinom primijetit ćemo da nakon nekog vremena pozadina nestaje, to je zato što je kamera izašla iz pozadine.

Kako bi riješili problem da nam se pozadina stalno prikazuje, morati ćemo osmislit neki drugi način, a taj način je da tu istu pozadinu kopiramo još dva puta i zrcalimo ju i postavimo ju na rubove glavne pozadine to možete vidjeti na sljedećoj slici:



Slika 16. *Prikaz iste pozadine zrcaljene dva puta*

Ako ponovno pokrenemo scenu pozadina će malo više potrajati ali opet neće potrajati značajno više, ali ovakav način nam daje mogućnost da izmjenjujemo pozadine odnosno slojeve (*eng.* Layer) pozadina (1, 2, 3). Kreirajmo jedan objekt i kao djecu tog objekta stavimo te tri pozadine kao što je prikazano na sljedećoj slici.

Slika 17. *Objekt imena FullBackground sa tri pozadine poredane jedna do druge*

Kada smo to napravili vrijeme je da napravimo skriptu koja će nam omogućiti ponavljajuću pozadinu. Pogledajmo skriptu *ScrollingBackground*: [5]

```
public class ScrollingBackground : MonoBehaviour
{
    public float paralaxSpeed;

    public float backgroundSize;

    private Transform cameraTransform;
    private Transform[] layers;

    private float viewZone = 10;
    private int leftIndex;
    private int rightIndex;

    private float lastCameraX;

    private void Start()
    {
        cameraTransform = Camera.main.transform;
        lastCameraX = cameraTransform.position.x;
        layers = new Transform[transform.childCount];
        for (int i = 0; i < transform.childCount; i++)
        {
            layers[i] = transform.GetChild(i);
        }
        leftIndex = 0;
        rightIndex = layers.Length - 1;
    }

    private void Update()
    {
        float deltaX = cameraTransform.position.x - lastCameraX;
        transform.position += Vector3.right * (deltaX * paralaxSpeed);
        lastCameraX = cameraTransform.position.x;
        if (cameraTransform.position.x > (layers[rightIndex].transform.position.x - viewZone))
        {
          ScrollRight();
        }
    }
    private void ScrollRight()
    {
        layers[leftIndex].position = Vector3.right * (layers[rightIndex].position.x + backgroundSize);
        rightIndex = leftIndex;
        leftIndex++;
        if (leftIndex == layers.Length)
        {
            leftIndex = 0;
        }
    }
}
```

U *Start* metodi inicijaliziramo sljedeće:

- Pozicija kamere – *cameraTransform*
- Slojevi pozadine – *layers*
- Indeks lijeve pozadine – *leftIndex*
- Indeks desne pozadine – *rightIndex*

Metoda *ScrollRight* nam služi kako bi prebacivali slojeve pozadine prema desno. Linija koda:

```
layers[leftIndex].position = Vector3.right * (layers[rightIndex].position.x + backgroundSize);
```

Mijenjamo poziciju sloja pozadine koji je skroz lijevo na poziciju skroz desno tako što sloju pozadine koja se nalazi skroz desno dodajemo duljinu jedne pozadine. Sada smo promijenili poziciju skroz lijeve pozadine na skroz desnu pozadinu. Nakon toga indeks desno jednak je indeksu lijevo jer je sloj pozadine prešao desno, lijevi indeks je sada za jedan veći. Također napravimo provjeru ako je lijevi indeks jednak broju pozadina tada je lijevi indeks jednak 0.
Update metoda provjerava je li potrebna zamjena slojeva pozadina. Linija koda koja nam provjerava je li potrebna zamjena je sljedeća:

```
if (cameraTransform.position.x > (layers[rightIndex].transform.position.x - viewZone))
{
    ScrollRight();
}
```

Ako je trenutna pozicija kamere veća od pozicije sloja pozadine koja se nalazi desno sa oduzetom proizvoljnom vrijednošću pogleda od 10 (*eng. viewZone)* tada pozadinu koja se nalazi skroz lijevo postavi na poziciju sloja pozadine koji se nalazi skroz desno zbrojenom sa veličinom jednog sloja pozadine.
Također vidimo da u *Update* metodi imamo i sljedeće tri linije koda:

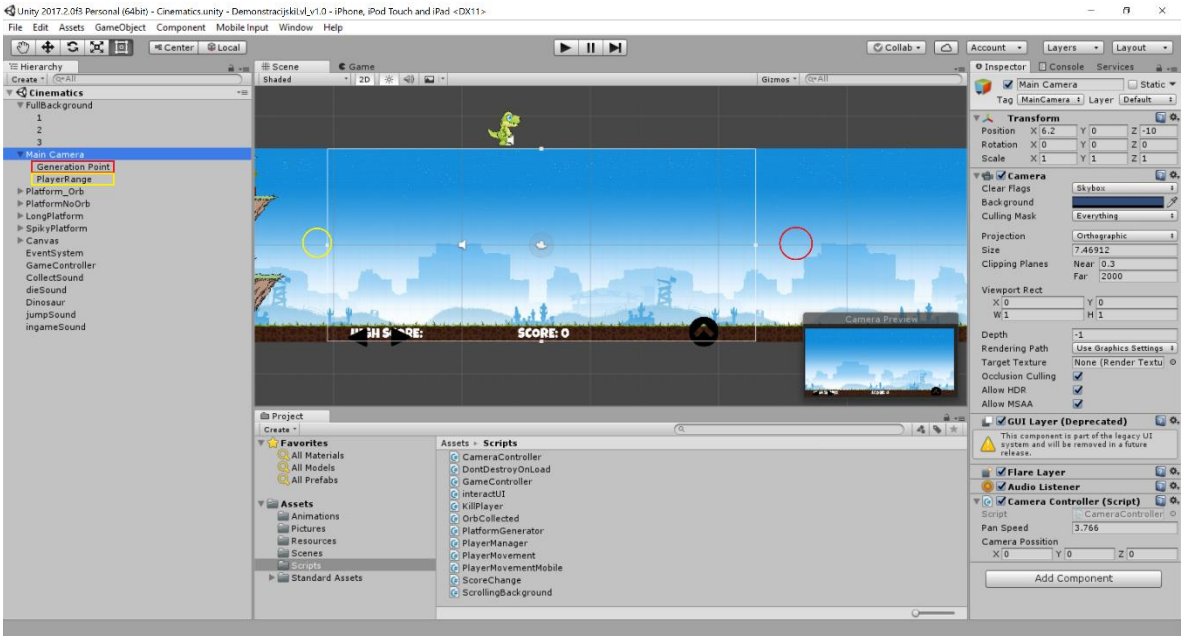
```
float deltaX = cameraTransform.position.x - lastCameraX;
 transform.position += Vector3.right * (deltaX * paralaxSpeed);
lastCameraX = cameraTransform.position.x;
```

Ove linije nam služe kako bi napravili efekt paralakse (*eng.* parallax effect), tj. Pomičemo pozadinu glavnu (roditelja) u smjeru kretanja kamere, a to radi sljedeća linija koda:
*transform.position += Vector3.right * (deltaX * paralaxSpeed);*

4.5. Alokacija i dealokacija objekata

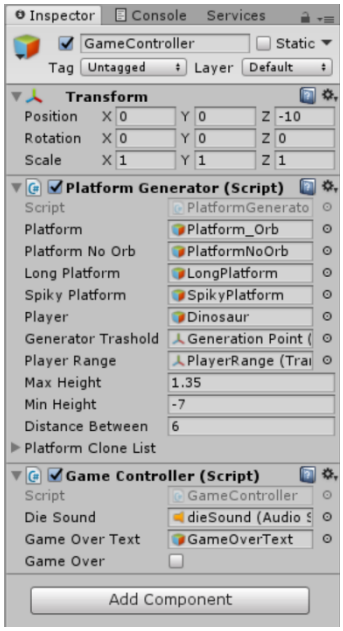
Kao i za svaku igru pa tako i za ovu potrebno je voditi računa o alokaciji i dealokaciji objekata kako bi igra imala bolje performanse i ne bi došlo do nepotrebnih grešaka ili rušenja igre. Budući da nam je pozadina dinamička potrebno će nam biti alocirati objekte jer ne znamo koliko će osoba igrati ovu igru pošto je beskonačna.

Za početak kao djecu glavnoj kameri dodati ćemo dva objekta, prvi po imenu *Generation Point*, drugi *PlayerRange*. *Generation Point* brinuti će o tome gdje će se objekti (platforme) alocirati, dok *PlayerRange* nam služi za provjeru pozicije igraćeg lika (ako je igraći lik s lijeve strane izašao iz igre, igra završava).



Slika 18. *Prikaz objekata koji su djeca objektu Main Camera*

Također dodajmo još jedan objekt po imenu *GameController* i dodajmo mu dvije C# skripte kao na sljedećoj slici.



Slika 19. *GameController objekt sa dvije C# skripte*

Opisati ćemo skriptu *Plarform Generator* koja generira platforme koje smo prethodno kreirali i upravlja završetkom igre preko *GameController* skripte. [6]

```
public class PlatformGenerator : MonoBehaviour {

    public GameObject platform;
    public GameObject platformNoOrb;
    public GameObject longPlatform;
    public GameObject spikyPlatform;
    public GameObject player;

    public Transform generatorTrashold;
    public Transform playerRange;

    public float maxHeight;
    public float minHeight;
    public float distanceBetween;

    private GameObject platformClone;
    public List<GameObject> platformCloneList;
    private int randomNumber;
    private float score;
    public static Sprite mySprite;


    void Start()
    {
        transform.position = new Vector3(transform.position.x + distanceBetween, transform.position.y, transform.position.z);
        mySprite = Resources.Load<Sprite> ("triangle") as Sprite;
    }

    void Update()
    {
        this.score = ScoreChange.scoreValue;
        AddPlatformAndMoveTrashHold();
        ChangeOrbTagAndRender();
        KillPlayer();
        DestroyPlatform();
    }
```

```
public void AddPlatformAndMoveTrashHold()
{
    if (transform.position.x < generatorTrashold.position.x)
    {
        distanceBetween = Random.Range(7, 8);
        randomNumber = Random.Range(1, 11);
        if (score == 0)
        {
            randomNumber = 7;
        }
        if (randomNumber > 6)
        {
            platformClone = Instantiate(platform,
                new Vector3(transform.position.x, Random.Range(minHeight, maxHeight)),
                transform.rotation);
        }
        else if (randomNumber == 3)
        {
            platformClone = Instantiate(platformNoOrb,
                new Vector3(transform.position.x, Random.Range(minHeight, maxHeight)),
                transform.rotation);
        }
        else if (randomNumber <= 2 && score > 100)
        {
            platformClone = Instantiate(spikyPlatform, new Vector3(transform.position.x, -0.71f),
                transform.rotation);
        }
        else
        {
            platformClone = Instantiate(longPlatform,
                new Vector3(transform.position.x, Random.Range(minHeight, maxHeight)),
                transform.rotation);
        }

        platformCloneList.Add(platformClone);
        transform.position = new Vector3(transform.position.x + distanceBetween, transform.position.y, transform.position.z);
    }
}

public void ChangeOrbTagAndRender()
{
    Transform platformGeneratedOrb = platformClone.transform.Find("orb");
    platformGeneratedOrb.tag = "orb";

    if (score < 30)
    {
        platformGeneratedOrb.GetComponent<Renderer>().material.color = Color.magenta;
    }
    else if (score >= 30 && score <= 50)
    {
        platformGeneratedOrb.GetComponent<Renderer>().material.color = Color.white;
    }
    else if (score > 100)
    {
        platformGeneratedOrb.GetComponent<SpriteRenderer>().sprite = mySprite;
    }
}

public void KillPlayer()
{
    if (player.transform.position.x < playerRange.transform.position.x)
    {
        GameController.instance.NotAlive();
    }
}
```

```
    public void DestroyPlatform()
    {
        if (platformCloneList != null && platformCloneList[0].transform.position.x < playerRange.transform.position.x)
        {
            Destroy(platformCloneList[0]);
            platformCloneList.RemoveAt(0);
        }
    }
}
```

Imamo atribute za platforme, objekte *generatorTrashold* i *playerRange* koje smo prije prikazali, maxHeight i minHeight vrijednosti koje brinu o poziciji generiranih platformi vertikalno, distanceBetween za horizontalno, također imamo pomoćni objekt tipa GameObject, listu GameObjecta i randomNumber koji nam služi kako bi instancirali određenu platformu ovisno o tom broju.

Poziciju kamere dobivamo pomoću transform.position, u *Update* metodi hvatamo trenutni rezultat, dodajemo platformu, mijenjamo izgled orbova, ubijamo igrača i uništavamo platforme (dealociramo ih).

AddPlatformAndMoveTrashold metoda nam generira platforme tako da provjeravamo ako je pozicija kamere na apscisi manja od pozicije generatorTrasholda na x osi.

```
Instantiate(platform, new Vector3(transform.position.x, Random.Range(minHeight, maxHeight)), transform.rotation);
```

Instantiate klonira objekt, ako ovako zadamo uzimamo trenutnu poziciju generatorTrasholda i alociramo platformu na koordinati koja je zadana.

Klonove platforme spremamo u listu kako bi kasnije dealocirali alociranu platformu. Također pomićemo generatorTrashold za distanceBetween na osi apscise.

ChangeOrbTagAndRender služi nam za promjenu alocirane platforme odnosno njenog djeteta (orb) na oznaku „orb” i mjenjamo njezin Renderer ili SpriteRenderer.

KillPlayer metoda provjerava da li je pozicija igrača na osi apscise manja od playerRange objekta. Ako je pozivamo metodu *NotAlive* iz klase *GameController* preko njezinog statičkog atributa „*instance*”.

DestroyPlatform metoda služi za dealokaciju platforme iz liste na indeksu 0, jednostavnim pozivom metode koju Unitya - *Destroy* kojoj predajemo parametar tipa GameObject.

4.6. Zapis rezultata igre i postavki

Kako smo do sada vidjeli kako svaka „kugla" nazvati ćemo ju „orb" koju skupljamo donosi neke bodove ovisno o trenutnom rezultatu, napraviti ćemo jednu klasu koja će nam služit za promjenu trenutnog rezultata kao i zapis najvećeg rezultata. Za početak napravimo dva UI objekta i postavimo ih u scenu kao na slici (*eng.* Score, HighScore). Također napravimo jedan prazan objekt i nazovimo ga *Score Manager*, njemu predamo skriptu kao komponentu i zaslužan je za upravljanje skriptom.

Slika 20. *UI tekstualni elementi u sceni*

Kada smo to napravili, napravimo klasu po imenu *ScoreChange*. Sada ćemo objasniti kako pomoću te klase mijenjamo i spremamo rezultat.

```
public class ScoreChange : MonoBehaviour
{
    public static float scoreValue;
    public Text txtScore;
    public Text txtHighScore;

    void Start()
    {
        scoreValue = 0;
        txtHighScore.text = "HIGH SCORE: " + PlayerPrefs.GetFloat("HighScore", 0);
    }
    void Update()
    {
        ChangeHighScore();
        txtScore.text = "SCORE: " + scoreValue;
    }

    public void ChangeHighScore()
    {
        if (scoreValue > PlayerPrefs.GetFloat("HighScore"))
        {
            PlayerPrefs.SetFloat("HighScore", scoreValue);
            txtHighScore.color = Color.green;
            txtHighScore.text = "HIGH SCORE: " + PlayerPrefs.GetFloat("HighScore");
        }
    }
}
```

Kreirana klasa ima statičku *float* varijablu scoreValue, postavili smo ju na static jer time omogućavamo drugim klasama ili metodama da pristupaju ovom atributu direktno, ne moramo alocirati objekt tipa *ScoreChange* već pristupamo direktno *ScoreChange.scoreValue;* također u klasi imamo dva tekstualna objekta koja samo povučemo i ispustimo u skriptu budući da su

vidljivosti *public*. U *Start* inicijaliziramo vrijednost rezultata na 0, a također postavljamo vrijednost najvećeg rezultata (*eng.* HighScore) na vrijednost PlayerPrefab-a odnosno ako nije postavljena kao inicijalna vrijednost je 0. *Update* metoda poziva metodu *ChangeHighScore* koja provjerava trenutnu vrijednost rezultata i uspoređuje ju sa najvećom vrijednosti rezultata. Ako je trenutna vrijednost rezultata veća od najveće vrijednosti (HighScore), tada se PlayerPrefab varijabla „HighScore" postavlja na trenutnu vrijednost rezultata. Kako bi korisnik znao da je prešao svoj osobni rekord također se tekst *HighScore* oboja u zelenu boju.

5. Zaključak

Unity je odličan alat za brzu, jednostavnu i fleksibilnu izradu igara. Ako imamo opće znanje o programiranju i pogledamo dokumentaciju na njihovoj službenoj stranici možemo na lagan način kreirati razne stvari. Unity također nije zahtjevan kada dođu u pitanju performansi, lagano je kreiranje dealokacije i ostalih stvari kao što su objekti, zvuk i slično tako da možemo napraviti kreativne igre u relativno kratkom vremenu. U radu je prikazana izrada 2D platformne igre. Sve skripte vezane uz igricu pisane su u programskom jeziku C#, spriteovi i audio datoteke preuzete su s Interneta. Igra se sastoji od dvije scene jedna scena je glavni izbornik koji sadrži UI objekte za interakciju korisnika koji može započet igru, postaviti opcije u igri i izaći iz igre. Prikazali smo kako se izrađuju animacije i kako određeni pritisak tipke utječe na promjenu stanja varijable koja mijenja trenutnu animaciju igraćeg lika. Glavni naglasak igre bio je za demonstraciju rješavanja problema „beskonačne" pozadine na način da pozadinu kopiramo i napravimo tri sloja pozadine i prebacujemo početni indeks na krajnji indeks kako bi ona bila dinamička. Vidjeli smo kako možemo alocirati i dealocirati objekte, prilikom izrade igre potrebno je posvetiti pažnju prilikom dealokacije objekata da ne bi došlo do prepunjavanja radne memorije. Cilj nam je stvoriti igru koja neće zahtijevati veće performanse računala tokom vremena, zato je potrebno voditi posebnu pažnju prilikom dealokacije objekata da to bude pravovremeno i proračunato. Vidjeli smo kako se rješava dealokacija koristeći metodu *Destroy*, tu metodu koristili smo pravovremeno (u trenutku kad nam je platforma s orbom izašla iz onoga što korisnik vidi (objekt *PlayerRange* vidi *slika 18*.) na svome zaslonu odmah smo dealocirali te platforme jer nam više nisu potrebne).

Popis literature

1. Marin Rabar (2016.), Izrada 2D platformske računalne igre u Unity Engine-u, Poveznica: *https://repozitorij.etfos.hr/islandora/object/etfos:957/preview*, dostupno 13.08.2018.
2. Godbold, A., & Jackson, S. (2016), *Mastering Unity 2D Game Development*, dostupno 13.08.2018.
3. Unity 3D, Poveznica: *https://docs.unity3d.com/Manual*, dostupno 06.08.2018.
4. Calabrese, Dave (2014.), *Unity 2D game development*, dostupno 13.08.2018.
5. Datt 2300 Game Development, Programming in Unity, Poveznica: *https://datt2300.wordpress.com/2018/02/08/week-6-playtesting/*, dostupno 13.08.2018.
6. Github, *Unity: A complete 2d game tutorial (ClonyBird)*, Poveznica: *https://github.com/mnapier/ClonyBird*, dostupno 13.08.2018.

Popis slika