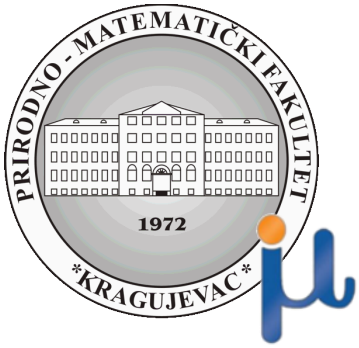
Kragujevac April 2017.



COLORSWITCH

CLONE

AUTORI:

Strahinja Milosavljević

Jovana Radovanović

Sadržaj:

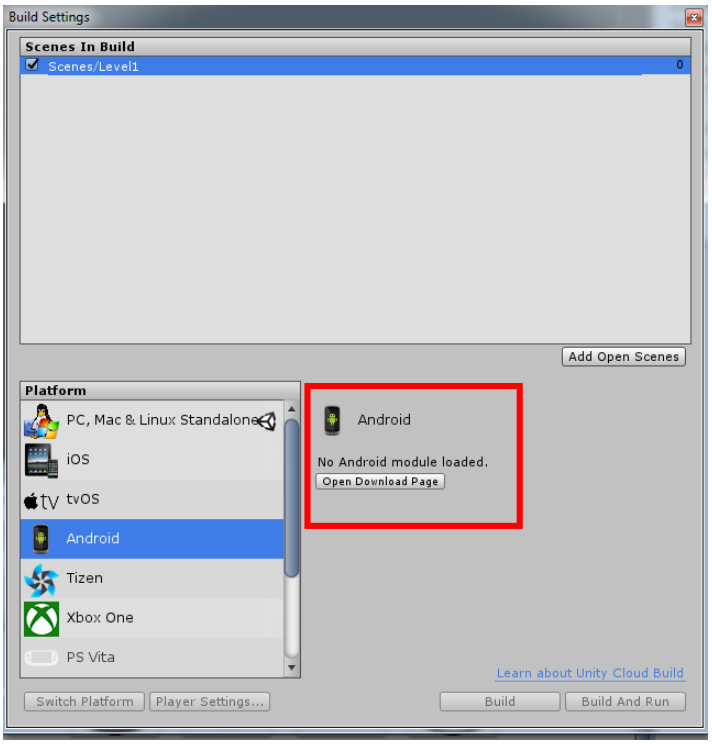
1. Uvod

1.1. Napomene potrebne za izradu projekta

2. Ideja za Color Switch igricu prvenstveno je u vlasništvu “Fortafy Games” i koristi se u ovoj literaturi samo u edukacione svrhe. Mi smo ovde kreirali Clone verziju kako bi pokazali kako ona može da se napravi u Unitiju.
3. Za potrebe kreiranja dizajna (**Front-enda**) ove igrice korišćena je verzija **Unity 5.5.2** i preporučujemo da ažurirate starije verzije na novije zbog nekih pogodnosti i mogućih razlika sa starijim verzijama ukoliko ste imali Unitz pre iyrade ovog projekta.
4. Za kodiranje (**back-End**) pisan je kod u C# u **Visual Studio 2015** ali može i neki drugi editor kao što je npr. Mono Developer-u
5. Voditi računa o čestom čuvanju projekta i Scena kako bi se izbeglo nepredviđeno gubljenje podataka I postignutog učinka. Za potrebe čuvanja scena preporučujemo da se kreira poseban folder u **Assets/Scenes/…** kao i da se daju smislena imena scenama.
6. Davati smislena imena folderima, skriptama, promenljivama i objektima kako biste mogli lakše da pratite strukturu foldera projekta i sam kod.
7. Obratiti pažnju za koju platformu pravite igricu i najbolje bi bilo da se na početku projekta izvrše potrebna podešavanja.
8. Svaka scena da bi mogla da se pokrene kao apk mora prvo biti “izbildovana” a to se radi tako što se odabere **File->Build Settings**
9. Vodite računa da struktura projekta bude uredno raspoređena zbog vase lične preglednosti i boljeg snalaženja kako za Vas tako i za onog ko čita Vaš kod.

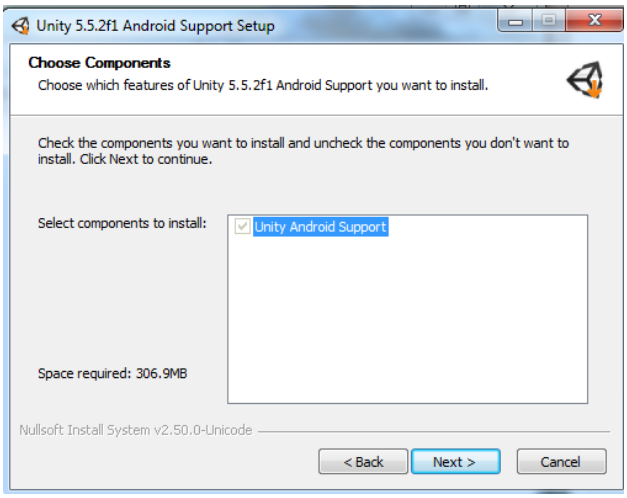
1.2. Odabir platforme i orijentacija ekrana

Da bi se određena scena mogla pokrenuti/uključiti u igricu i da bi mogla da prikaže ono što ste joj dodali neophodno je prvo “Izbildovati” je kao i selektovati platformu za koju želimo da kreiramo igricu, a Unity nam u tome jako olakšava posao jer je vrlo jednostavno. Ove opcije snalaze u **File->Build Settings.** Ako imamo neku scenu za bildovanje onda je dodajemo odabirom na Add Open Scenes (ovo dugme dodaje aktivnu scenu).



Da bismo promenili platformu moramo prvo imati instaliran modul za platformu koju želite da selektujete. Pošto mi želimo da kreiramo krajnju igricu za android kada selektujete opciju “**Android**” videćete da će Vam tražiti modul koji verovatno

nemate. Kliknite na opciju **Open Download Page** i skinuće Vam se neophodna instalacija koju nakon toga pokrenete.

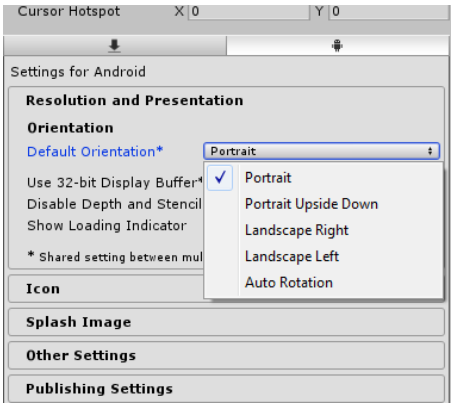


Instalacija će Vam tražiti da ugasite Unity editor kako bi mogao da implementira instalirani modul po završetku tako da to i uradite. Kada se vratite u projekat idaberite opciju **Swtich Platform** koja će Vam sada biti omogućena i **ikonica** za unity će se prebaciti na **Android**.

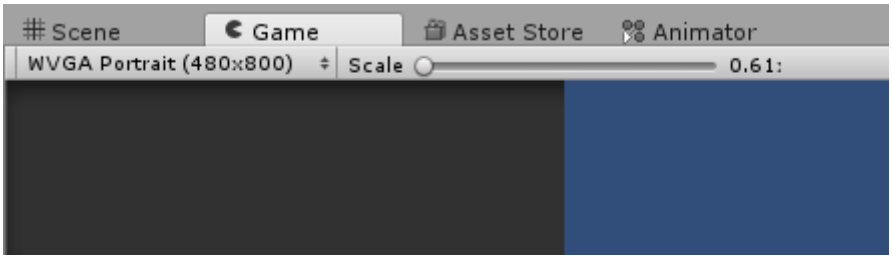
NAPOMENA: NAJBOLJE JE DA PLATFORMU SELEKTUJETE PRE POČETKA IZRADE PROJEKTA ZBOG RAZLIČITIH REZOLUCIJA I DIMENZIJA KAMERE KAKO NE BISTE POSLE NA KRAJU MORALI SVE DA NAMEŠTATE. ZATO ODMA NA POČETKU VAM BUDE SVE PODEŠENO I SPREMNO ZA RAD.

Kada pokrenete ponovo Vaš projekat i u **Build Settings** odaberete android platformu videćete da će sada prozor izgledati drugačije. Omogućeno je i dugme **Player Settings** koje se koristi za neka osnovna podešavanja aplikacije. Pošto ovu igricu pravimo za Android mi ćemo odabrati **Android platformu**. Ali igrica će moći da se pokrene i na **PC računaru**. Vi možete lako da odaberete i **iOS** ili koju god Vi želite platformu.

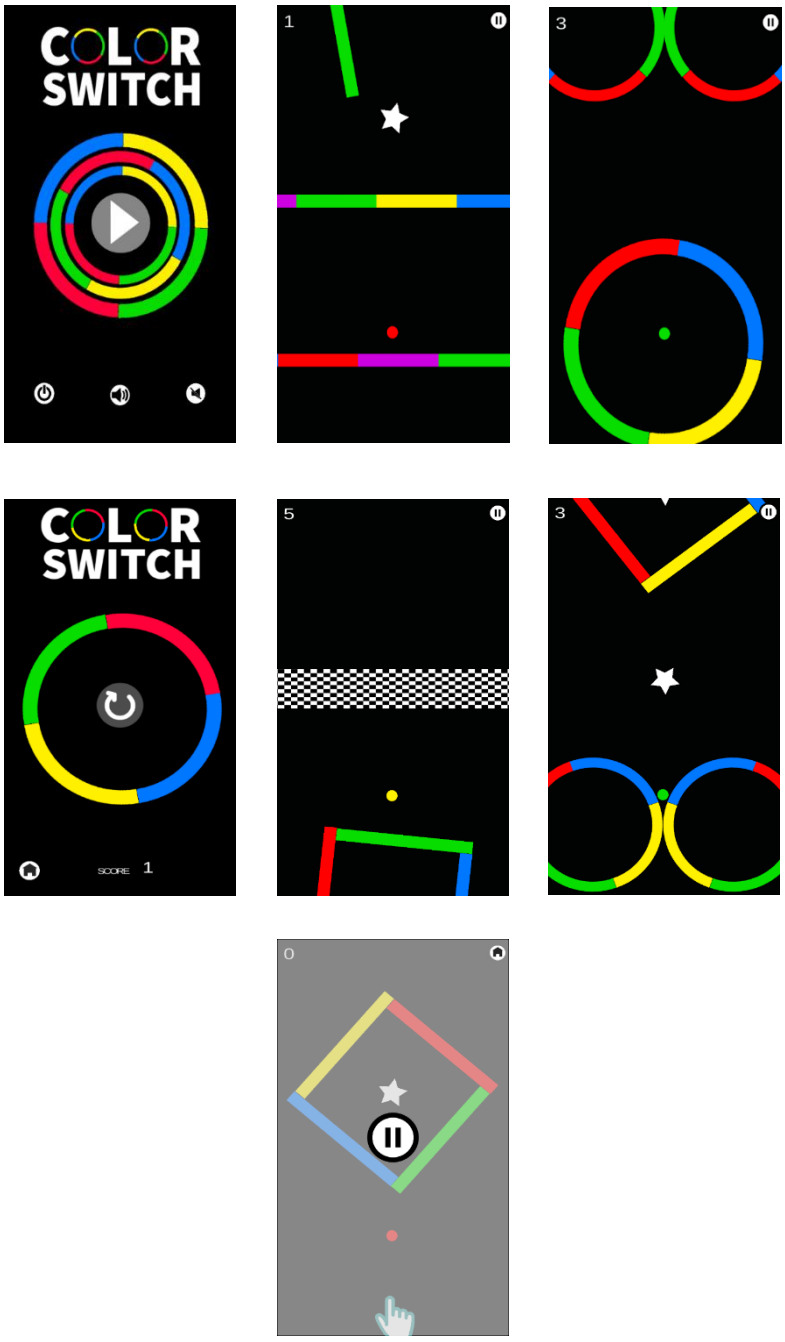
Ako želite da promenite **automatsko rotiranje ekrana** tj orijentaciju ekrana na androidu manuelno idete na **File->Build Settings->Player Settings** u Inspector prozoru odabrati polje za Android I opciju **Default Orijentation postavite** na **Portrait** kao na slici**.**



Ostalo je još samo da podesite rezoluciju **Kamere** tj kako će se Vaša igrica prikazivati. Odaberite opciju kao sa slike.

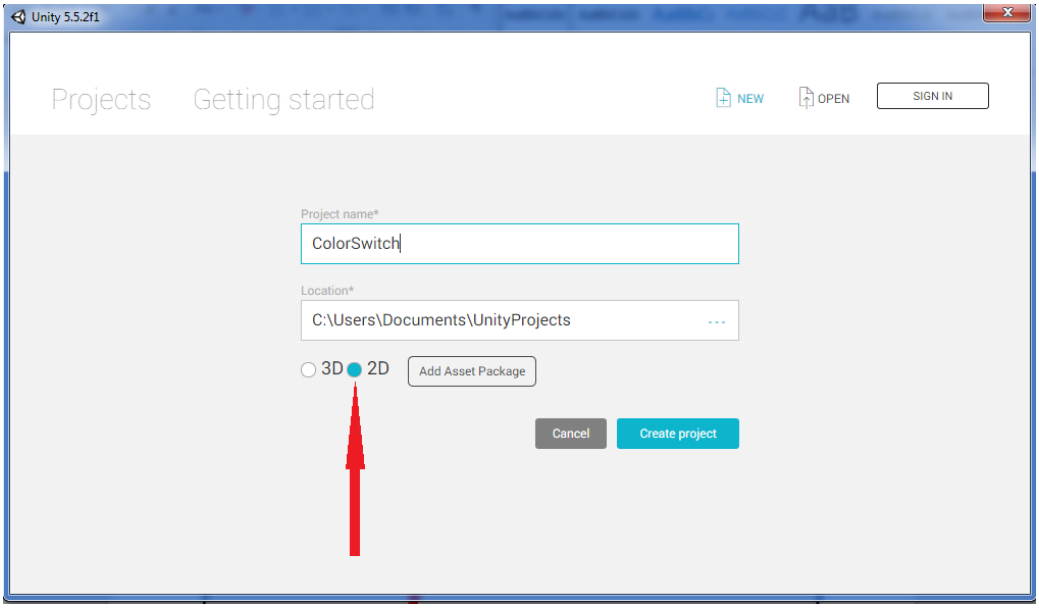


1.3. Kratak pregled finalnog proizvoda



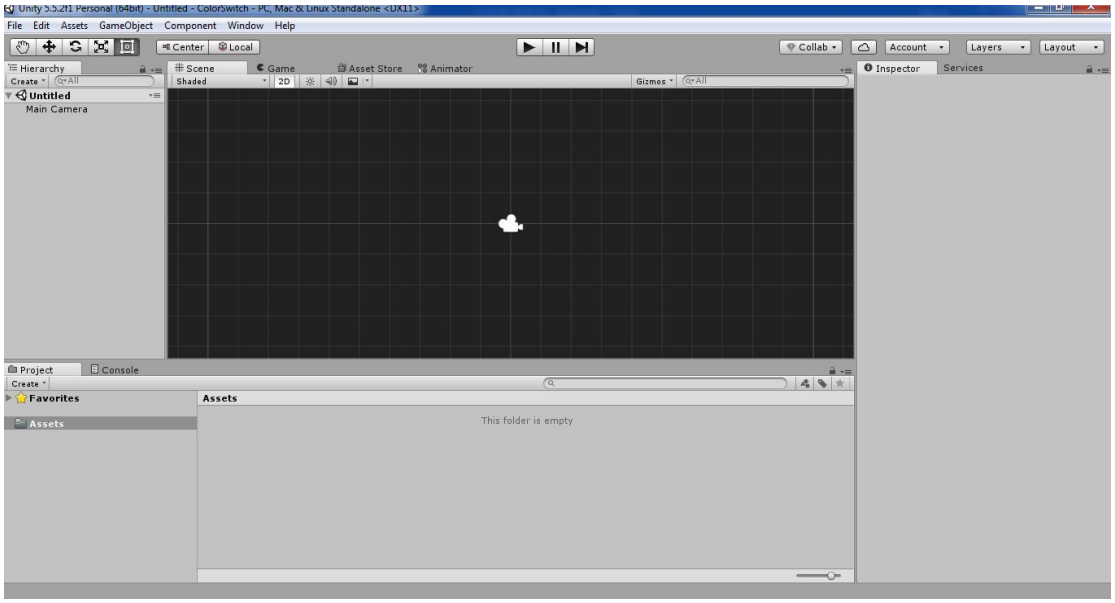
2. Kreiranje loptice

U ovom poglavlju započinjemo izradu naše igrice. Pokrenuti program u kome ćemo kreirati našu igricu **Unity** 5.5.0. Nakon startovanja Unity programa prikazuje se prozor za kreiranje novog projekta, koji će biti ***2D***. U okviru polja ***Project name*** unosimo naziv projekta, u ovom slučaju **ColorSwitch** ,a ***Location*** predstavlja mesto gde će se naš projekat čuvati. Lokaciju možemo lako promeniti odabirom „...“.



Slika 1 Početni prozor

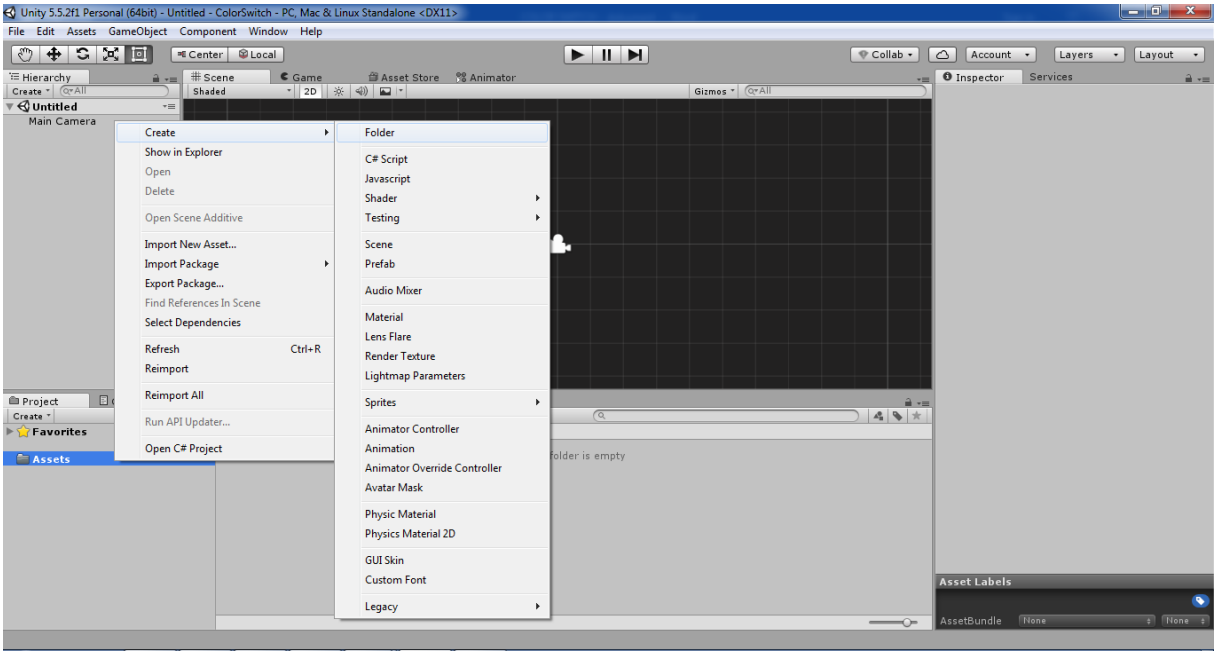
Nakon klika na dugme ***Create project*** otvara se okruženje-Editor Unity programa. **Assets** folder je u početku prazan.



Slika 2 O kruženje Unity programa

Prvi korak pri kreiranju igrice je snimanje glavne scene. **Scene** ćemo čuvati u okviru foldera ***Scenes***, koji će biti podfolder foldera ***Assets*** (desni klik na folder

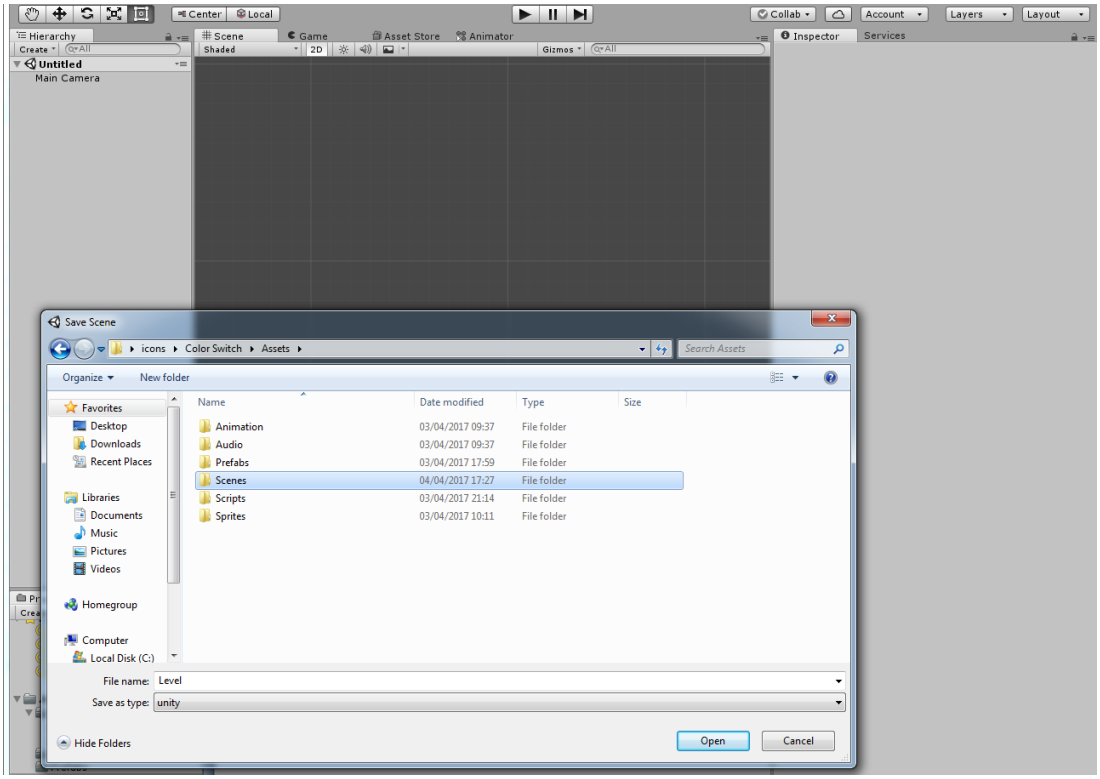
Assets ->Create ->Folder-> imenujemo ga Scenes).



Slika 3 Kreiranje foldera Scenes

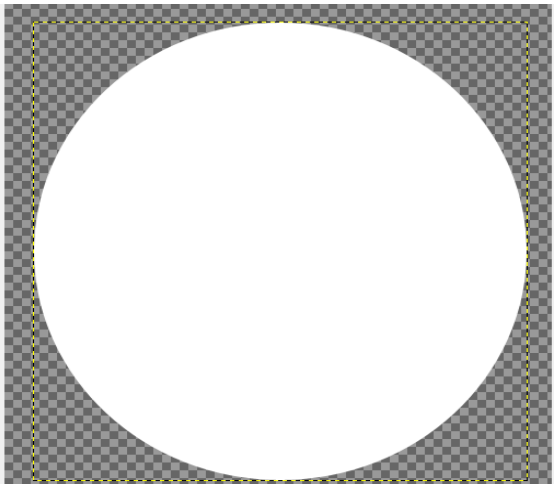
Scenu snimamo na sledeći način:

File->Save Scenes->nakon čega biramo folder Scenes ->scenu čuvamo pod nazivom Level.



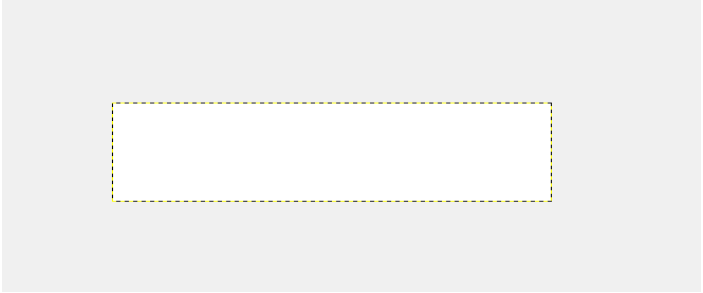
Slika 4 Snimanje scene Level

Sledeći korak jeste kreiranje. Uvozimo dva sprajta, u podfolder Assets/*Sprites/.* Prvi pod nazivom ***sphere*** , koji predstavlja lopticu i ima sledeći izgled:



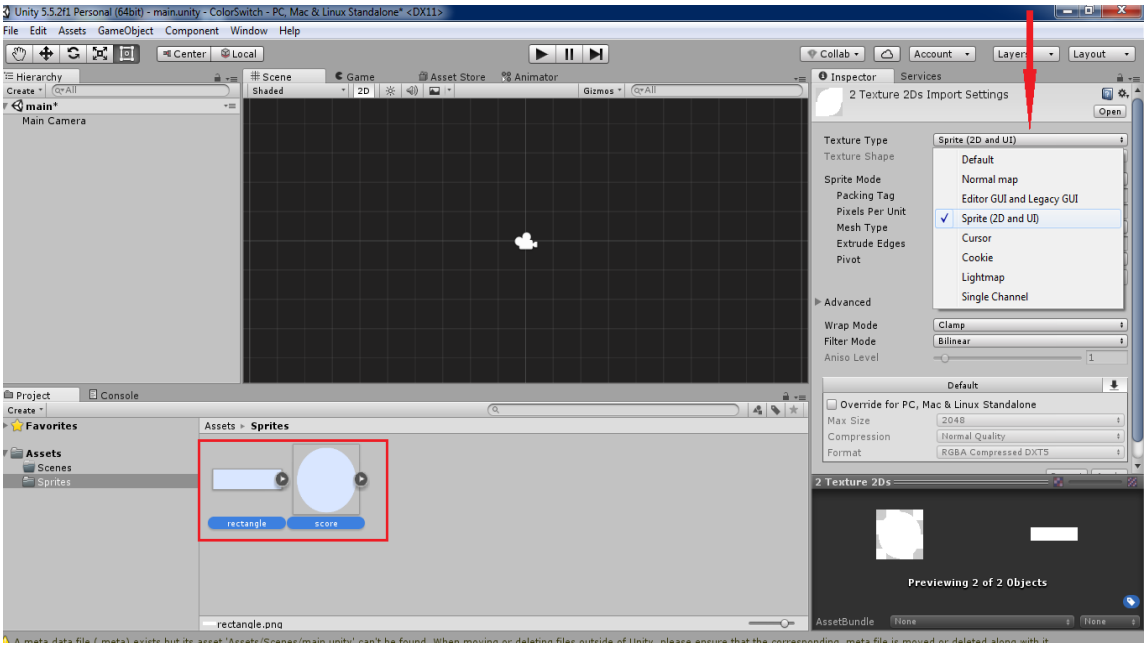
Slika 5 Sphere sprite

Drugi pod nazivom ***rectangle*** koji ćemo koristiti za kreiranje prepreka/objekata



Slika 6 Rectangle sprite

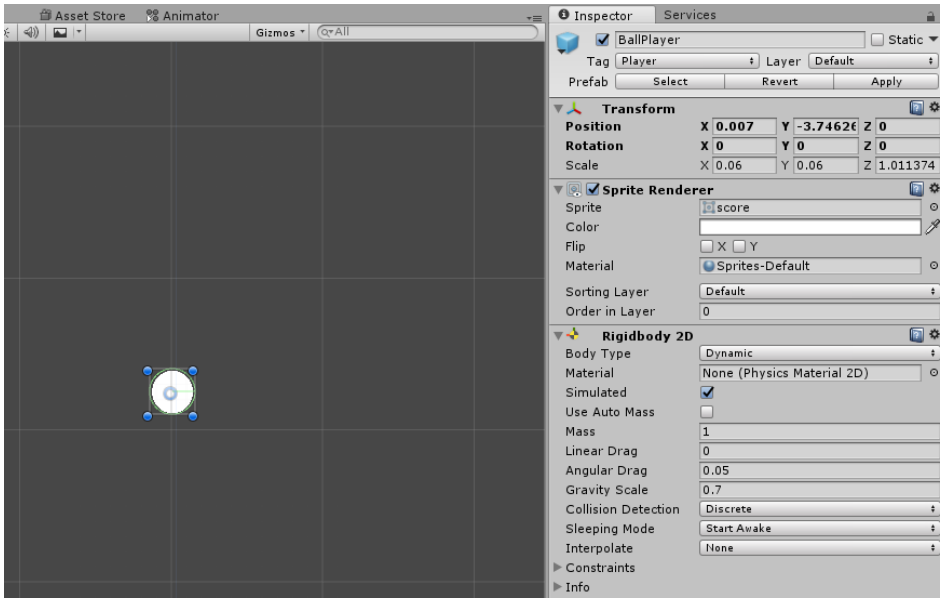
Nakon što prebacimo sprite-ove u folder **Sprites**, selektujemo oba sprite-a, i u Inspector prozoru sa desne strane za ***Texture Type*** postavimo ***Sprite(2D and UI)*** , kao na sledećoj slici:



Slika 7 Podešavanje za dva dodata sprite-a

Iz foldera Sprites oba sprite-a prevučemo na glavnu scenu (*Level*), u ***Hierarchy*** prozor. Klikom na sprite ***sphere*** možemo sada da mu promenimo ime i nazvaćemo ga ***BallPlayer*** kao igračeva loptica.

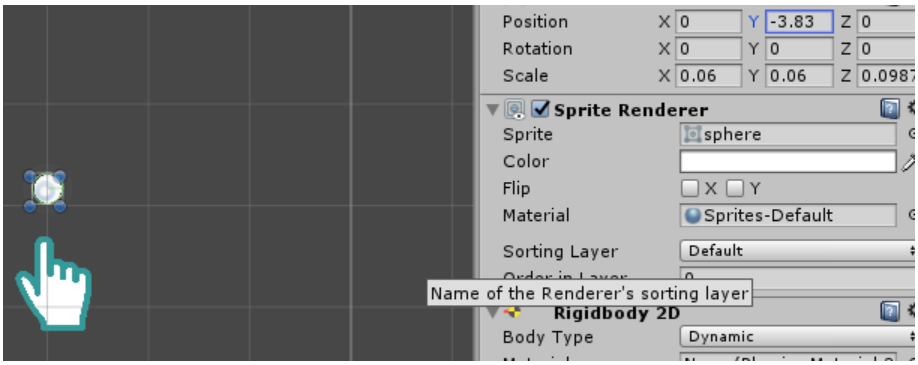
U ***Hierarchy*** prozoru, dobijamo mogućnost podešavanja tog sprite-a u okviru ***Inspector*** prozora (isto važi i za sprite ***rectangle***).



Slika 8 Sprite-ovi prebačeni na main scenu

Kada kreiramo objekat, Unity mu automatski dodeljuje komponentu ***Transform*** pomoću koje možemo da podešavamo:

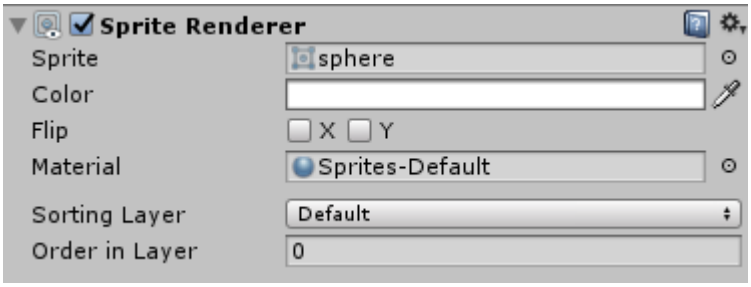
- **Position** – menjamo poziciju objekta
- **Rotation** - rotiramo naš objekat
- **Scale** – menjamo dimenzije objekta



Slika 9 Transform komponenta za sphere

Pozicije i dimenzije postavljamo proizvoljno.

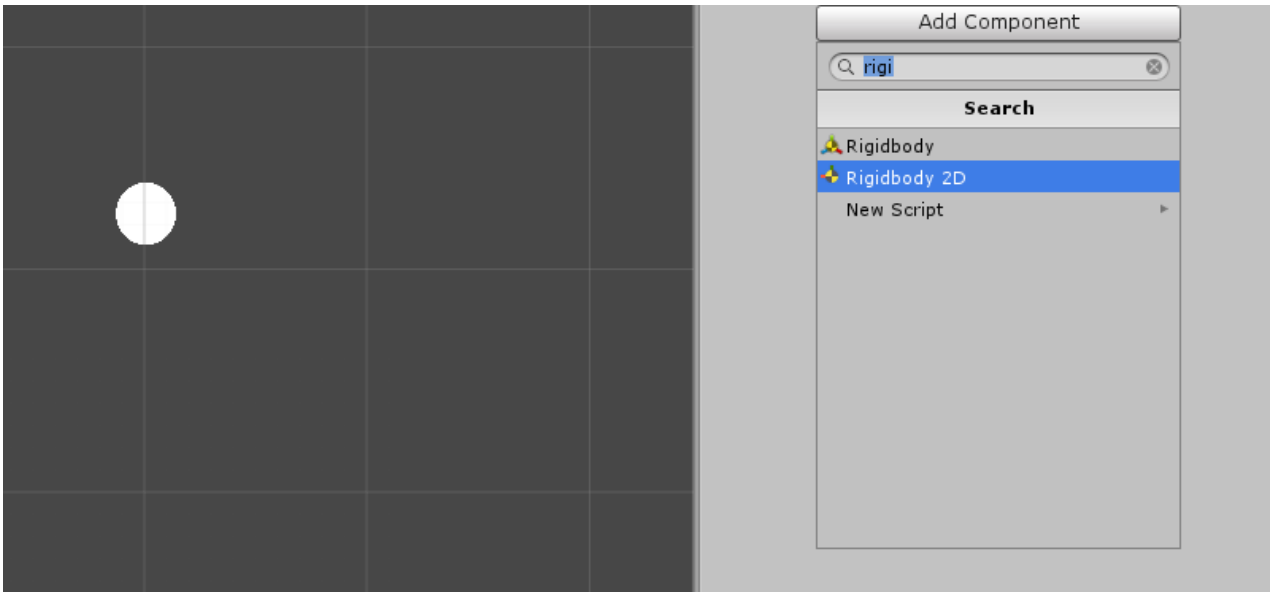
Kada je reč o sprite-u kao u našem slučaju, pored ***Transform*** komponente Unity dodaje ***Sprite Renderer*** komponentu, koja je pokazivač na sprite i u kome se pored ostalih mogućnosti nalazi i polje ***Color*** kao i ime sprajta:



Slika 10 Sprite Render komponenta za sphere

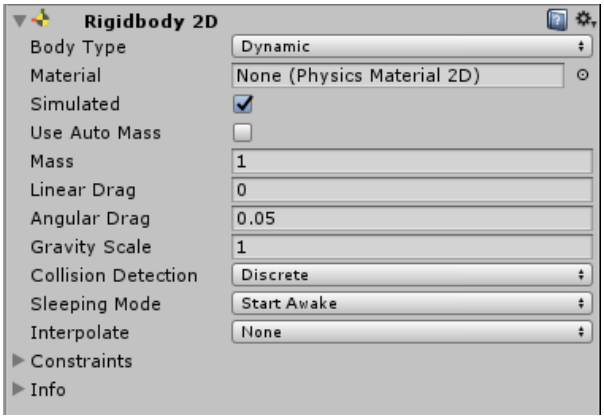
Sledeće što je potrebno sprite-ovima dodati je fizika.

1. Selektujemo ***BallPlayer*** u ***Hierarchy*** *prozoru -> Component -> Phisics 2D -> Rigidbody 2D ili*
2. *Klikom na* ***BallPlayer*** u Inspector prozoru odaberemo opciju ***Add Component*** i nađemo ***RigidBody2D*** što ćemo mi uraditi.



Slika 11 Dodavanje Rigidbody komponente

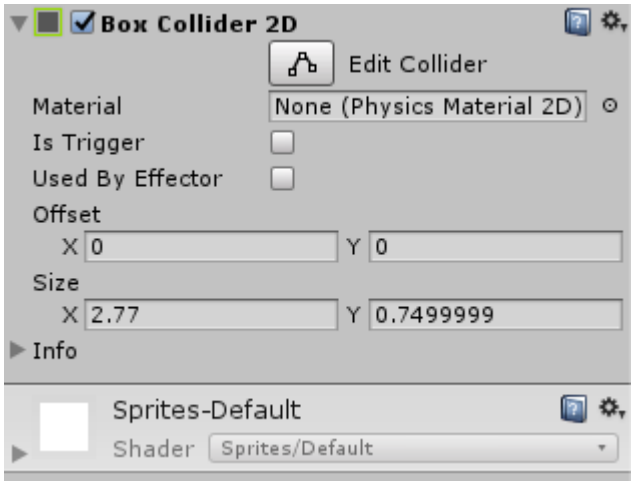
Nakon čega nam se u Inspector prozoru za selektovani sprite dodaje ***Rigidbody2D*** komponenta:



Slika 12 Rigidbody 2D za sphere

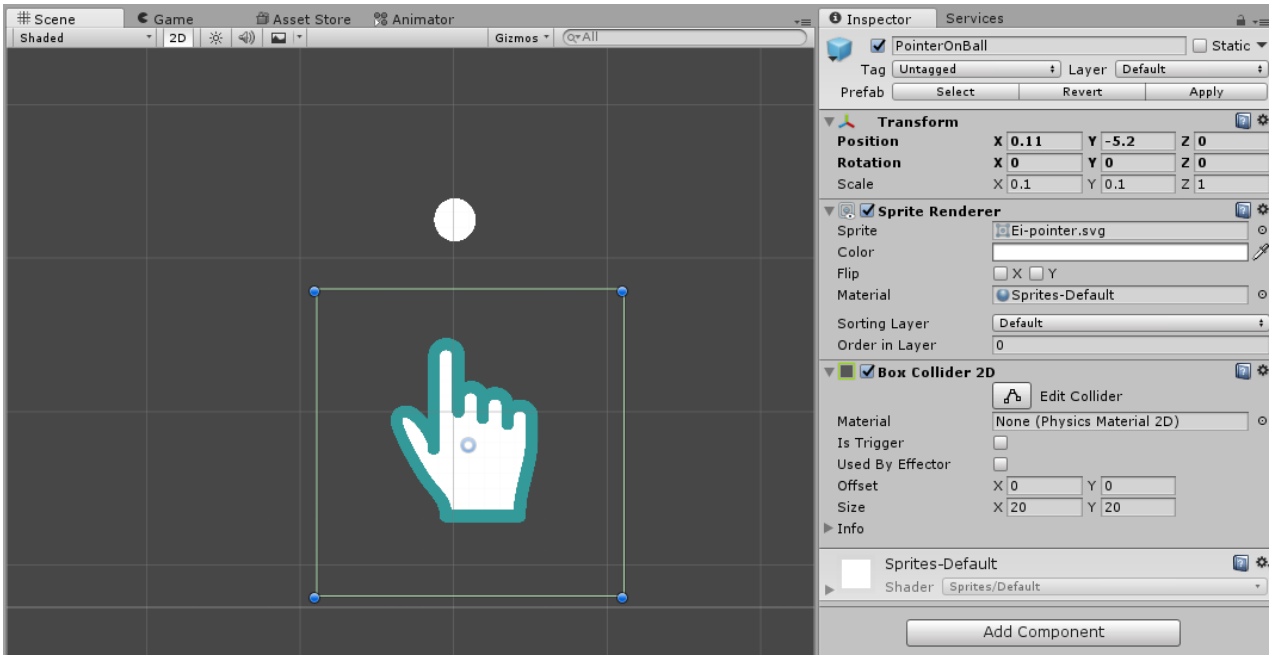
Ova komponenta stavlja sprite-ove pod kontrolu engine-a za **fiziku**. To omogućava da sprite bude pod uticajem gravitacije*(**Gravity Scale** je 1 – koliko jako će gravitacija uticati na objekat)*. Ukoliko sada pokrenemo projekat, klikom na dugme ***play***, primetićemo da naša loptica pada, jer na nju sada deluje sila gravitacije. Međutim mi želimo da naša loptica ostane u mestu i da ne pada. Zato ćemo za sada kreirati podlogu koja će joj omogućiti da ne propadne već da stoji u mestu kada dodirne sprite ***PointerOnBall*** .

Iz tog razloga selektujemo sprite iz foldera ***Assets/Sprites/pointeronball****, i* dodajemo u Hijararhiju kao objekat, a zatim mu dodajemo komponentu **BoxCollider2D** *(**Add Component -> Box Collider 2D**)*. Postavićemo ga ispod loptice. Takođe dodajemo i **BoxCollider2D** na sprajt **rectangle** kao sa slike 13.



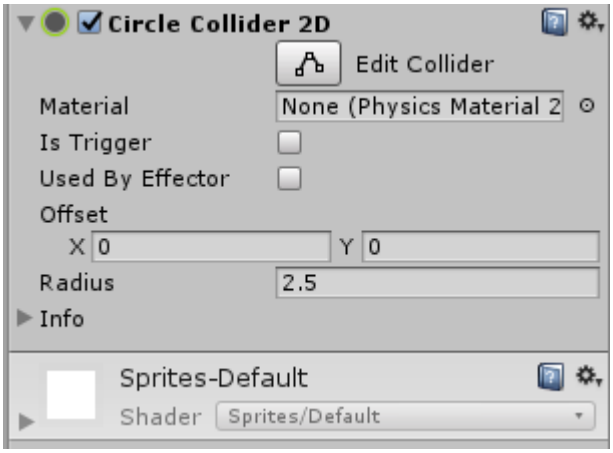
Slika 13 Box Collider 2D za rectangle sprite

Dakle, BoxCollider2D dodajemo kao komponentu za sprite-ove kojima želimo da omogućimo „težinu“ odnosno fiziku sudaranja sa drugim objektima (kolizijom). Nakon dodavanja BoxCollider2D-a, sprite ***rectangle****,* postaje oivičen zelenom linijom, što se može primetiti u ***Scene*** prozoru:



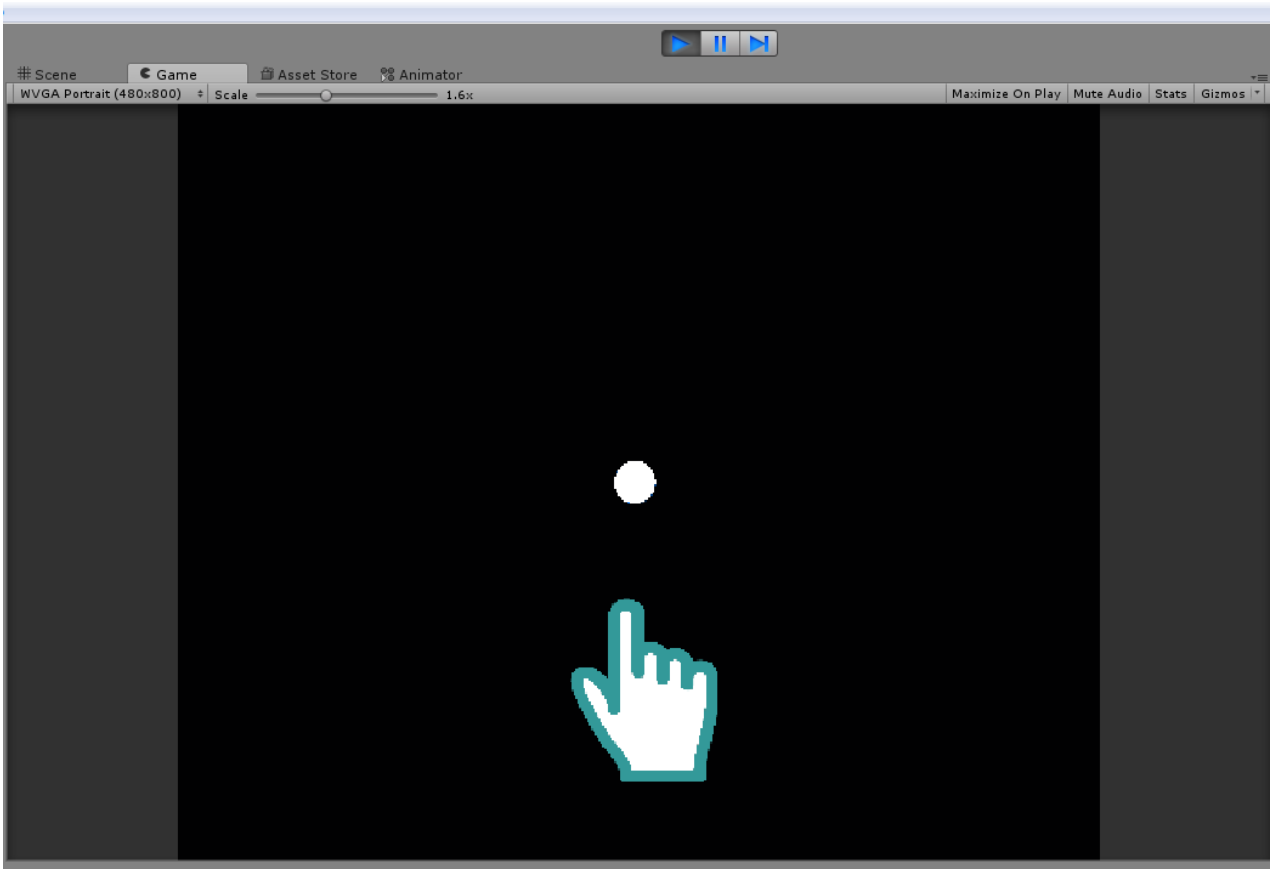
Slika 14 Izgled sprite-a nakon dodavanja Box Collider – a 2D

Za sprite ***BallPlayer*** dodajemo (u skladu sa oblikom) ***CircleCollider2D*** *(Add Component -> Circle Collider 2D).*



Slika 15 Circle Collider 2D za sphere

Kada pokrenemo projekat videćemo da dolazi do kolizije ova dva objekta:



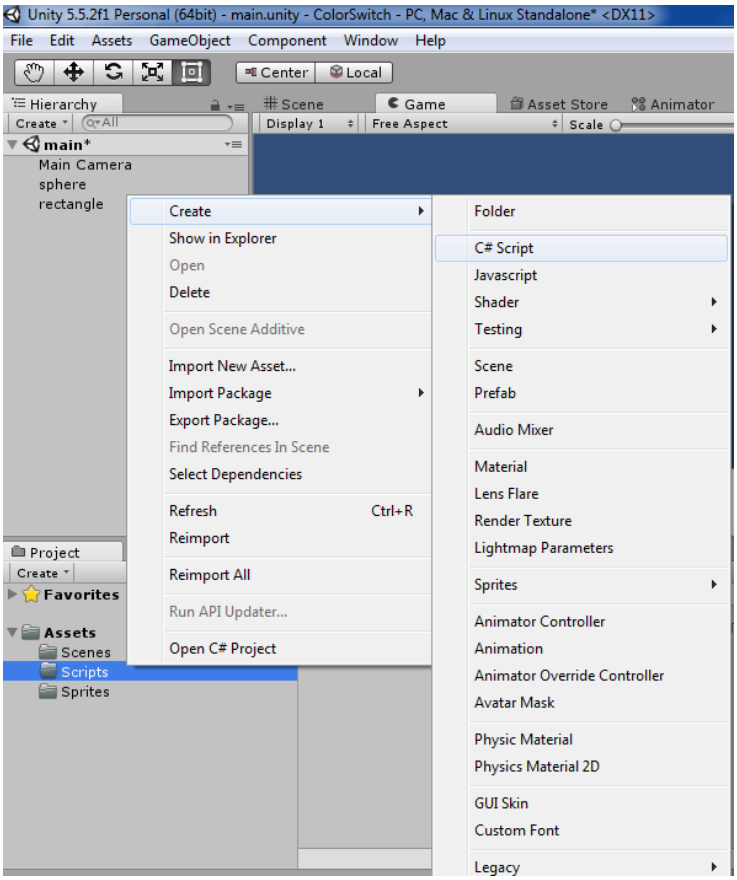
Slika 16 Loptica se zaustavlja pri dodiru sa objektom PointerOnBall

2.1. Kretanje loptice – igrača

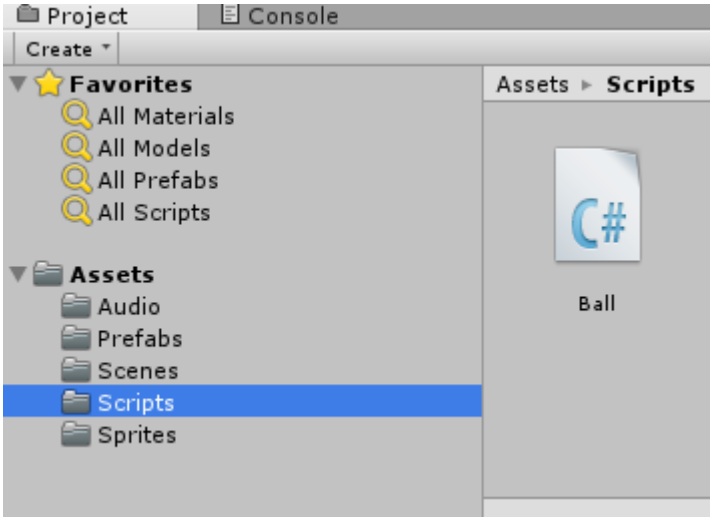
Sledeća funkcionalnost koja je jako važna kod igrice **Color Switch** jeste da omogućimo da se loptica svaki put kada kliknemo na taster miša/touch kreće nagore. Da bismo ovo omogućili kreiramo *C#* skriptu, u okviru foldera **Assets/*Scripts/*** na sledeći način: Desni klik na koren foldera

Scripts -> Create -> C# Script -> Ball.cs.

Skriptu ***Ball.cs*** pokrećemo preko *Visual Studija.* Osnovne metode koje kreira Unity su ***Start()***, koja se poziva prilikom pokretanja skripte i ***Update()*** metode. Umesto Update() metode, koristimo **FixedUpdate()** koja se uglavnom primenjuje kod vremenski osetljivih kodova ili za fiziku kao što je ovde slučaj.



Slika 17 Kreiranje C# skripte

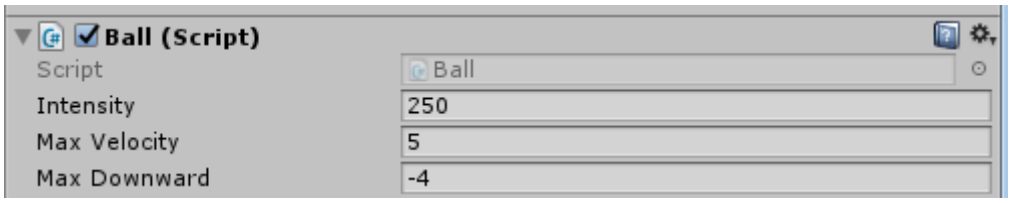


Slika 18 Ball.cs

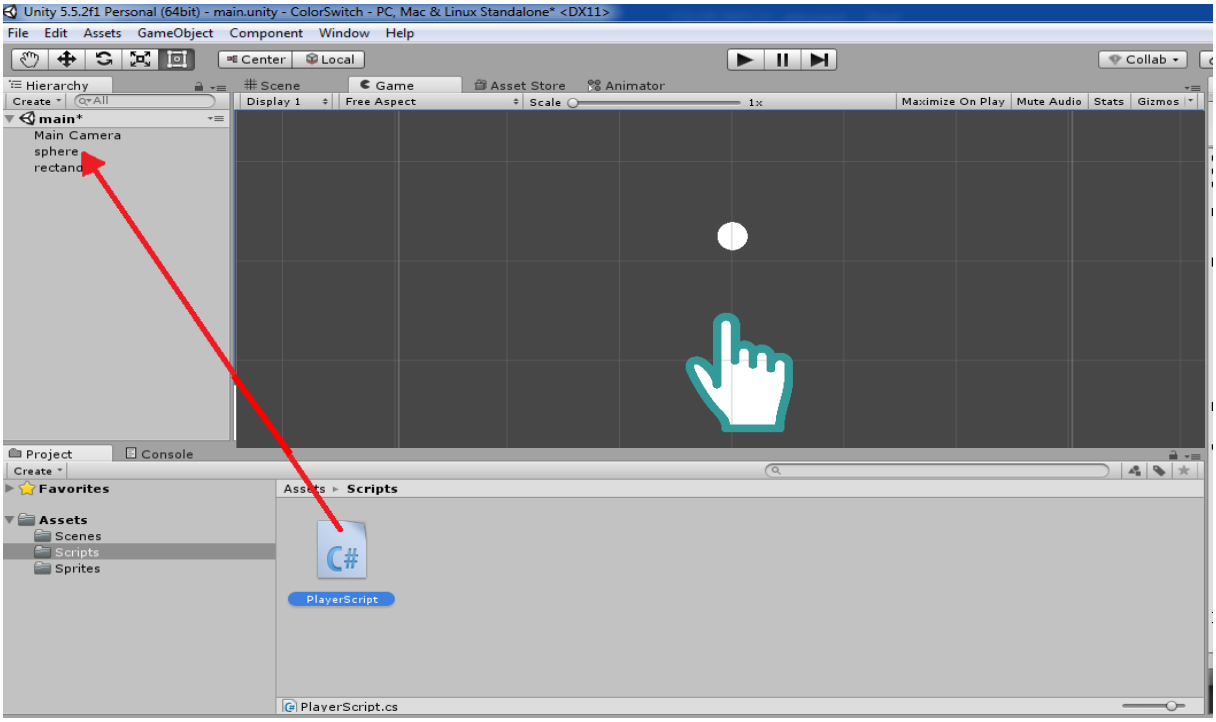
Skripta ***Ball.cs*** ima sledeći sadržaj:

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

public class Ball : MonoBehaviour
{

    public float intensity; //jačina sa kojom loptica odskače
    private Rigidbody2D rb2d; //referenca na kompnentu Rigidbody2D
    public float MaxVelocity ; // dokle najviše da promeni poziciju po y osi -skok
    public float MaxDownward;    // dokle da padne

    // Use this for initialization
    void Start()
    {
       rb2d = GetComponent<Rigidbody2D>();
    }

    void FixedUpdate()
    {
        if (Input.GetMouseButtonDown(0)) //ako kliknemo levim tasterom misa
        {
            //add force upward, or add impulse upward
            if (rb2d != null)
            {
                rb2d.AddForce(Vector2.up * intensity);
            }
        }
        if (rb2d.velocity.y > MaxVelocity) // da onemoguci lopti da skace previsoko
        {
            rb2d.AddForce(Vector2.down * 30);
        }

        if (rb2d.velocity.y <  MaxDownward) // max velocity kada pada y ispod 0 dokle moze
da pada
        {
            rb2d.AddForce(Vector2.up * intensity);
        }
    }

}
```

Sada je potrebno ovu skriptu dodeliti objektu ***BallPlayer,*** a to radimo tako sto skriptu prevučemo na objekat ili odabirom objekta pa **Add component** i nađemo skriptu po imenu.. U inspector prozoru dodeliti veličine trima promenljivama po vašem nahođenju ali na slici 19 je dat prikaz kako smo mi našli „zlatnu“ sredinu.



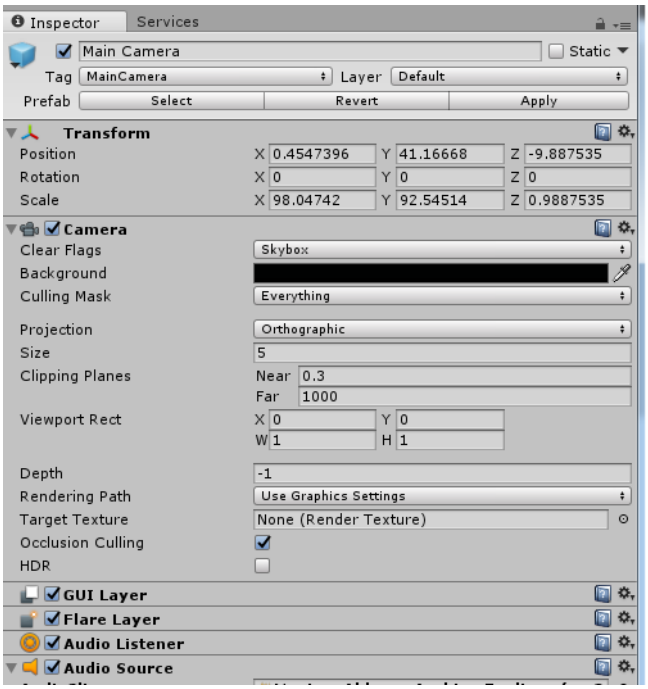
Slika 19 Podešavanje promenljivih za kretanje loptice



Slika 20 Prevlačenje skripte na objekat

Sada se loptica na klik miša kreće na gore, međutim nju ne prati kamera. Da bismo to obezbedili ***Main Cameru*** prevučemo na objekat ***BallPlayer***, čime taj objekat postaje roditelj objekata *Main Camera*. Na slici 21 je prikazano kako se ***Main Camera*** dodeljuje objektu ***BallPlayer*** čime postaje loptica roditelj kameri i time smo rešili problem praćenja loptice kamerom.

Slika 21 Main camera kao dete loptice



Slika 22 Main Camera

Podesiti da kamera **Main Camera** bude veličine otrpilike 5. To se radi tako što se promeni vrednost polja ***Size*** u prozoru Inspector.

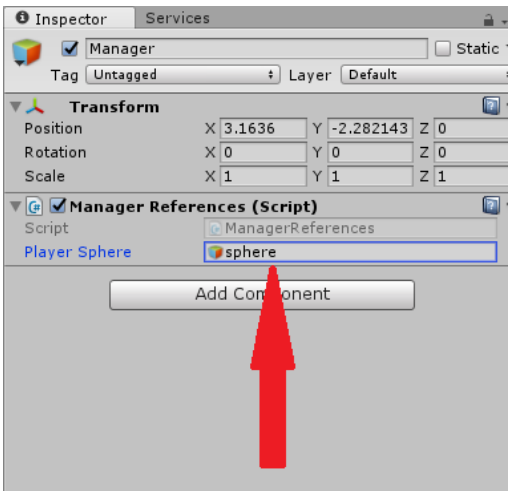
2.2. Menjanje boje loptice

Zbog boljeg vizuelnog efekta, boju pozadine, koja je plava, menjamo u **crnu**, na sledeći način: klikom na ***Main Camera*** u **Hiararchy** panelu, dobijamo mogućnost podešavanja pozadine u okviru ***Inspector*** panela. Tu okviru komponente ***Camera***, imamo ***Background*** sekciju gde biramo boju pozadine, postavite da bude **crna**.

Trenutna boja loptice je bela. Naš cilj je da omogućimo da loptica menja boje nasumično ali pošto ne zna koje boje sme da koristi mi moramo to da joj kažemo na osnovu predefinisanog intervala boja odnosno niza. Iz tog razloga kreiramo prazan objekat koji ćemo nazvati **Menager** *(Game Object -> Create Empty -> objekat imenujemo Menager).* Ovo je objekat koji će kontrolisati celu scenu i biće referenca na poene, na lopticu itd. Sada kreiramo skriptu ***MenagerReferences.cs*** u folderu ***Assets/Scripts/*** koju dodeljujemo objektu ***Menager***. Skripta izgleda ovako:

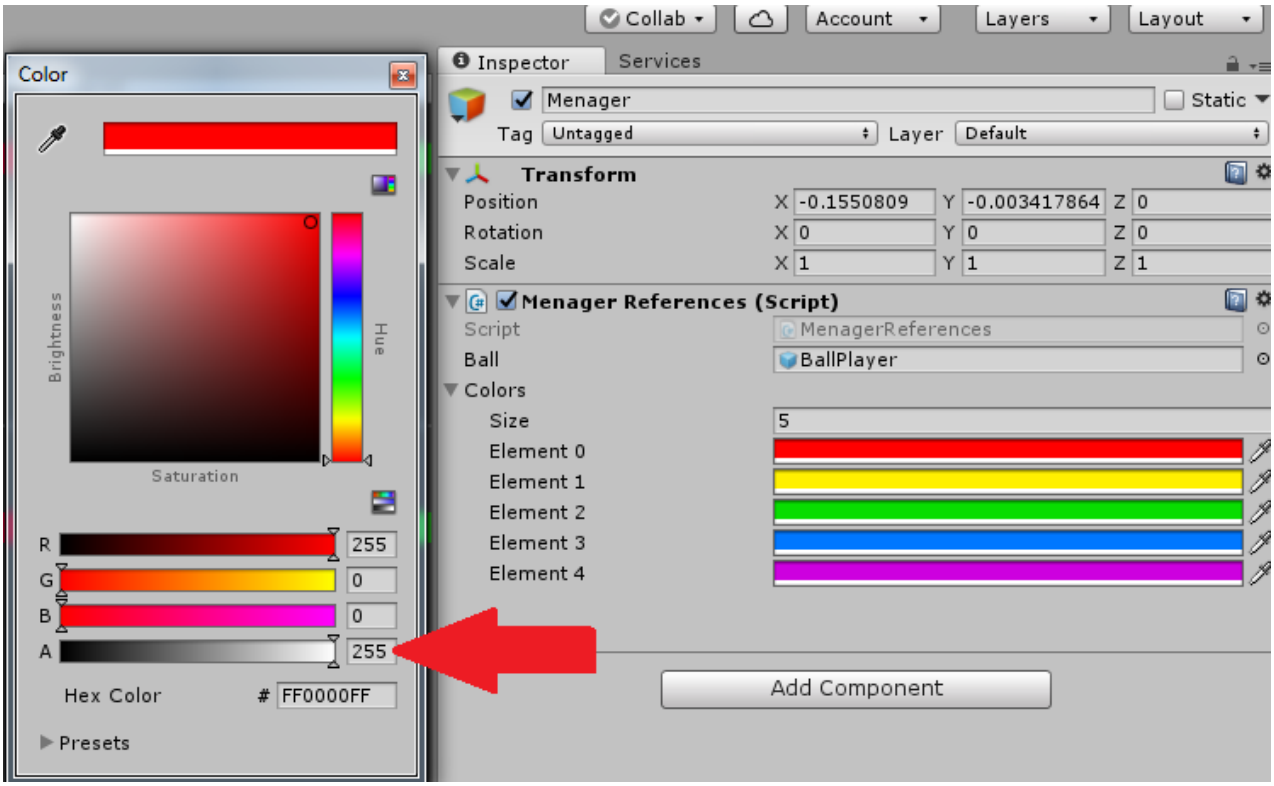
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MenagerReferences : MonoBehaviour
{
    public GameObject ball; //referenca na objekat sphere
    public Color[] Colors;     //referenca na niz boja
    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Nakon sto u skripti navedemo referencu na neki objekat iz Unity programa, npr. referenca na objekat ***BallPlayer***, u okviru Inspector prozora te skripte pojavljuje se polje sa nazivom reference, gde je potrebno prevući objekat, na koji se ta referenca odnosi.



Slika 23 Dodavanje reference skripti

Nakon dodavanja reference na niz boja, u Unity programu, u okviru komponte koja definiše ***ManagerReferences*** skriptu, dobijamo mogućnost definisanja prvo veličine niza, a zatim i boja koje želimo da se nadju u nizu ***Colors***:



Slika 24 Dodavanje boja

Strelica na prethodnoj slici ukazuje da alfa uvek mora da ima vrednost **255** da bi loptica mogla da primi boju inače će imati boju crnu kao osnova i neće se primećivati na kameri. Sada je potrebno dodeliti boje objektima. Loptica treba da ima random boje, svaki put kada se igrica startuje. Da bismo to obezbedili modifikujemo skriptu ***Ball.cs*** na sledeci način:

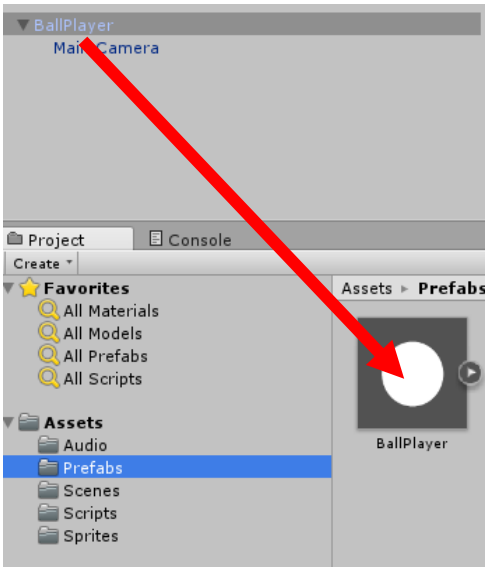
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Ball : MonoBehaviour
{
    public float intensity; //jačina sa kojom loptica odskače
    private Rigidbody2D rb2d; //referenca na kompnentu Rigidbody2D
    public float MaxVelocity ; // dokle najviše da promeni poziciju po y osi -skok
    public float MaxDownward;    // dokle da padne

    // Use this for initialization
    void Start()
    {
       rb2d = GetComponent<Rigidbody2D>();
       InitializeColor();
    }

    void FixedUpdate()
    {
        if (Input.GetMouseButtonDown(0)) //ako kliknemo levim tasterom misa
        {
            //add force upward, or add impulse upward
            if (body != null)
            {
                Rb2d.AddForce(Vector2.up * intensity);
            }
        }
```

```
        if (rb2d.velocity.y > MaxVelocity) // da onemoguci lopti da skace previsoko
        {
            rb2d.AddForce(Vector2.down * 30);
        }

        if (rb2d.velocity.y <  MaxDownward) // max velocity kada pada y ispod 0 dokle moze
da pada
        {
            rb2d.AddForce(Vector2.up * intensity);
        }
    }

    void InitializeColor()
    {
        //pravimo referencu na objekat Manager
              MenagerReferences refs =
              GameObject.Find("Menager").GetComponent<MenagerReferences>();
        if (refs!=null)
        {
              //broj boja
            int colorCount = refs.Colors.Length;
              //random boje u opsegu od 0 do maksimalnog broja boja
            int randomIndex = Random.Range(0,colorCount-1);
            //kreiranje nove random boje
            Color newColor = refs.Colors[randomIndex];
              //posto je loptica(sphere) Sprite, uzimamo tu komponentu i dodeljujemo joj
       boju
            SpriteRenderer render = GetComponent<SpriteRenderer>();
            render.color = newColor; //nova boja
        }
   }

}
```

2.3. Kreiranje Prefab elemenata

Kopiranje obejekata će sigurno proizvesti duplikate, ali osobine duplikata se moraju posebno menjati. Da bi se izbeglo ovako nešto, Unity podržava ***Prefeb assets*** tip sredstva koje omogućava da sačuvamo **GameObject** objekte zajedno sa komponentama i svojstvima.

Na kraju treba da kreiramo ***Prefab*** folder u ***Assets/Prefabs/*** kako bi tu smeštali kreirane objekte da ih ne bi duplirali i pojedinačno svakom podešavali opcije u Inspector prozoru prevlačenjem iz hijararhije projekta u folder prefabs dobijamo objekat sačuvan za ubuduće korišćenje na onoliko mesta koliko mi želimo sa predefinisanim opcijama. Sada Lopticu ćemo prevući u taj folder kako bi mogli kasnije istu lopticu za naredni nivo da samo prebacimo iz ovog foldera kao na slici 25.



Slika 25 Kreiranje Prefabs-a

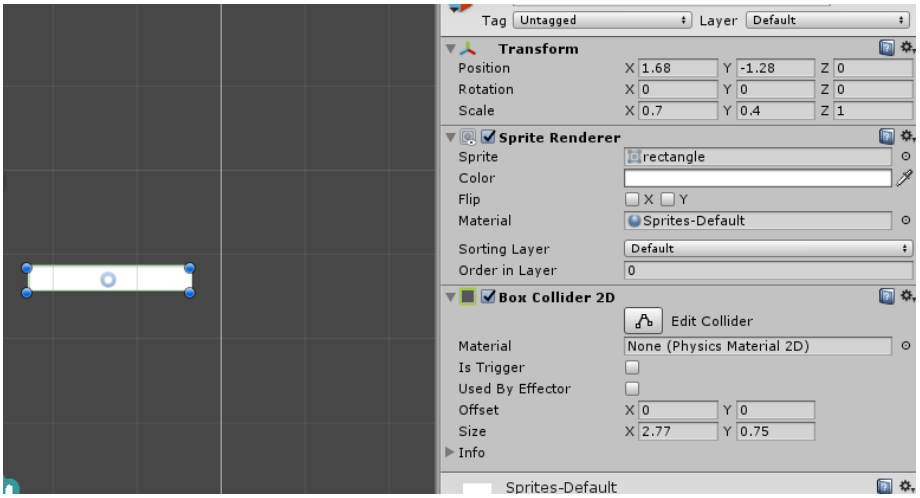
Ono što je Važno za Prefabs elemente jeste da ukoliko se nakon kreiranja prefab-a promeni neka vrednost objektu mora se odabrati u gornjem uglu opcija *„**Apply**“* kako bi se nove vrednosti dodelile svim istim objektima na sceni ukoliko želimo ako ne onda ostavljamo samo za taj element novu vrednost bez odabira apply opcije za sve.

3. Dodavanje drugog objekta

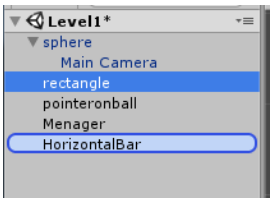
Dodaćemo običan ***Rectangle*** sprajt (ako ga niste dodali u prethodnom poglavlju) iz foldera **Assets/Sprites/** na scenu kako bi videli kako se loptica sudara sa drugim objektom i šta treba da objekat radi kao I podešavanja za objekte od kojih će se kasnije praviti razne druge prepreke.

3.1. Kreiranje objekta za koliziju

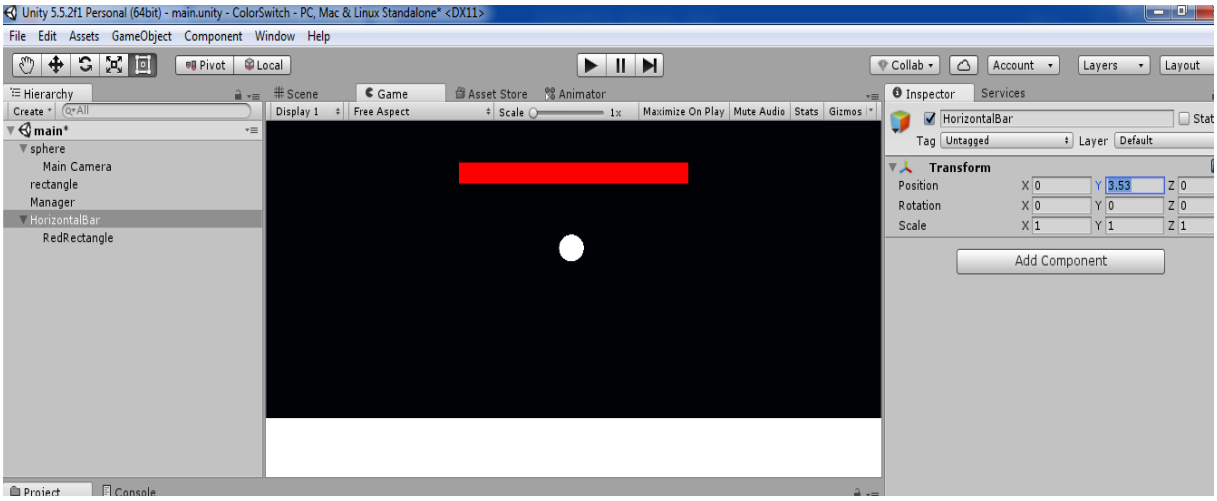
Nakon što smo dodali ***rectangle*** treba da podesimo širinu i visinu pravougaonika kao na slici. Posle kada ovaj objekat dodelimo jednom objektu i kreiramo više onda možemo da ga obrišemo iz hijararhije.



Slika 26 Podešavanja rectangle objekta

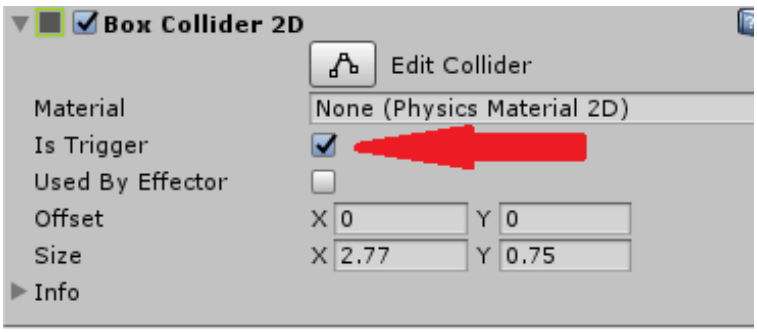


Zatim kreiramo prazan objekat ***HorizontalBar*** (*GameObject ->CreateEmpty).* Ovom objektu kao podobjekat dodajemo prethodno kreirani sprite ***rectangle,*** i imenujemo ga u ***RedRectangle***. Bojimo ga u crveno, a u okviru komponente ***Transform*** za taj objekat postavimo ***Position*** X:0 i Y:0. Zatim objekat ***HorizontalBar*** pozicioniramo iznad loptice kao na slici ispod:



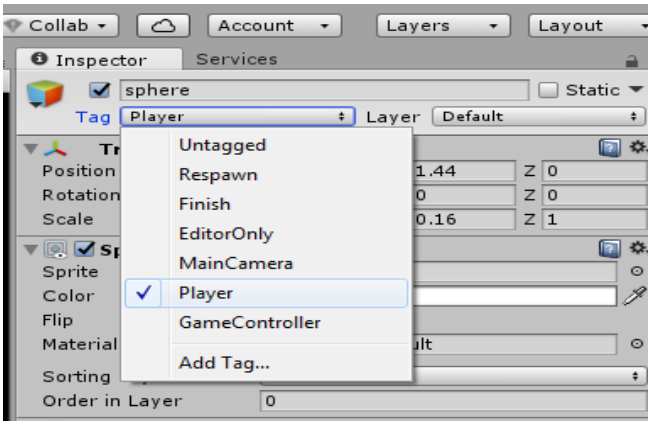
Slika 27 Dodavanje prve prepreke

Objektu ***RedRectangle*** sada dodajemo komponentu ***BoxCollider2D***, kako bi detektovali sudar sa lopticom. Kako bi loptica prolazila kroz objekat ***RedRectangle*** u komponenti *BoxCollider2D* potrebno je čekirati ociju ***Is Trigger*** da bi bila ***true*** .



Slika 28 Dodavanje BoxCollider 2D komponente objektu RedRectangle

Za objekat ***BallPlayer*** potrebno je postaviti tag na ***Player,*** Unity sam nudi u predefinisanim tagovima i tag ***Player*** ,ali i mi sami možemo da kreiramo tagove ako nam trebaju:



Slika 29 Za sphere tag Player

Sledeći korak jeste kreiranje skripte za koliziju sa drugim objektima i koristi se f-ja **OnTriggerEnter2D()**, koju dodajemo objektu ***RedRectangle,*** i imenujemo je **Assets/Scripts/*TriggerCollision.cs*** .

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

public class TriggerCollision : MonoBehaviour
{
    //za prepreke takodje koristimo boje definisena u MenagerReferences
    public ColorUsed colorIndex; //u Unity-ju imamo mogucnost selektovanja željene boje

    void Start()
    {
        //referenciramo se na skriptu ManagerReferences
        ManagerReferences refs =
GameObject.Find("Menager").GetComponent<ManagerReferences>();
        if (refs != null)
        {
            Color newColor = refs.Colors[(int)colorIndex];
            gameObject.GetComponent<SpriteRenderer>().color = newColor;//dodeljujemo
selektovanu boju objektu RedRectangle
        }
    }
```

```
    //funkcija koja detektuje koliziju izmedju prepreke na koju je postavljen triger,
    //i Collider2D objekta, u ovom slucaju loptice(koja ima Circle Collider2D komponentu)
    void OnTriggerEnter2D(Collider2D other)
    {
        //ako je objekat koji dolazi u koliziju loptica(koja ima tag Player)
        if (other.gameObject.tag == "Player")
        {
            //implementiramo logiku igre
            Color playerColor = other.gameObject.GetComponent<SpriteRenderer>().color;
//boja loptice
            Color thisColor = this.gameObject.GetComponent<SpriteRenderer>().color; //boja
prepreke
            if (playerColor != thisColor)
            {
                Debug.Log("GAME OVER!"); //ispis u konzoli a kasnije ćemo dodati šta će da
radi kada se sudari sa drugom bojom
            }
        }
    }
}
```

Verovatno će Vam bojiti u skriptu red gde je tip ***ColorUsed***. To je zato što nismo još definisali opseg boja koje smemo da koristimo odakle će on uzimati trenutnu boju objekta. Zato modifikujemo skriptu **MenagerReferences.cs** sa sledećim kodom koji dodajemo pre klase jer je u pitanju ***enum***. ***Enum*** predstavlja skup imenovanih integer promenljiva. Kreću od 0 i raste za +1. Voditi računa jer enum elementi dobijaju **integer** vrednosti iako su **imenovani** po **želji** i to istim redom kako su definisani, dakle prvi element ima vrednost indeksa 0 naredni 1... Nama su ovde boje pa ćemo definisati istim redom koji smo dodelili i **Menager** objektu.

```
public enum ColorUsed
{
    Red,
    Yellow,
    Green,
    Blue,
    Purple
}

public class MenagerReferences : MonoBehaviour
{
            .
            .
            .
}
```

3.2. Kretanje prepreke

Kako prepreka ne bi bila statična, potrebno je omogućiti njeno pomeranje levo i desno. Za to pišemo novu skriptu, koju dodeljujemo objektu ***HorizontalBar*** , roditelju objekta ***RedRectangle***, pod nazivom ***HorizontalMovement.cs*** .Ovom skriptom ćemo definisati koliko levo, odnosno desno će prepreka moći da se pomera.

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

```
public class HorizontalMovement : MonoBehaviour
{

    public float rangeMax = 10;
    public float direction = 1; //ako je 1 pomera se desno, -1 levo
    public float speed = 1; //brzina varira od objekta do objekta I posle se dodaje sve
brže I teže za više nivoe
    public float initialPositionX; //početna pozicija objekta pošto se pomera samo po X
osi

    // Use this for initialization
    void Start()
    {
        initialPositionX = this.transform.position.x;
    }

    // Update is called once per frame
    void Update()
    {
        //pomeranje prepreke
        this.transform.position = new Vector2(this.transform.position.x + (direction *
Time.deltaTime * speed),this.transform.position.y);

        if (this.transform.position.x > initialPositionX + rangeMax)
        {
            direction = -1;
        }
        if (this.transform.position.x < initialPositionX - rangeMax)
        {
            direction = 1;
        }
    }
}
```