‹packt›

PREVOD TREĆEG IZDANJA

Unity 2022

Razvoj mobilnih igara

Izgradnja i publikacija “zaraznih igrica” za Android i iOS

JOHN P. DORAN

kompjuter biblioteka

PREVOD TREĆEG IZDANJA

Unity 2022

Razvoj mobilnih igara

Skenirajte QR kod, registrujte knjigu i osvojite nagradu

Unity je renomiran "igrač" u sferi razvoja igara za mobilne uređaje, a novo izdanje, Unity 2022, sadrži obilje novih i uzbudljivih funkcija. Uz ovu knjigu ćete naučiti da koristite Unity, tako što ćete izgraditi "igricu" za mobilne uređaje i objaviti je na najpopularnijim prodavnicama aplikacija, učeći usput i najnovije funcionalnosti.

Ovu knjigu krasi obiman i praktičan pristup razvoju igara za mobilne uređaje, što će vam pomoći da izgradite igricu Beskrajna trka. Počećete od jednostavnog Unity projekta i uroniti u osnovne aspekte uspešnog razvoja i objavljivanja video igara. Steći ćete određene veštine, kao što je dodavanje komandi, monetizacija igre pomoću Unity Ads-a i kupovine u aplikaciji, stvaranje intuitivnog interfejsa (UI) i elegantna integracija funkcionalnosti društvenih medija. Uz to, Unity analitički alati će vam pružiti dragocen uvid u preference i ponašanje igrača. Takođe, istražićete mogućnosti implementacije proširene stvarnosti koju nudi Unity 2022, da biste svoju igru učinili privlačnijom.

Kada pročitate ovu knjigu, bićete spremni da koristite Unity 2022 za izgrdnju, optimizaciju i publikovanje robustnih multiplatformskih igrica, na jeziku C#, uz razvijen set veština i unapređen progamerski renome.

Šta ćete naučiti

- dizajnirate prilagodljiv korisnički interfejs za mobilne igre
- otkrijete kolizije, primite korisničke unose i kreirate pokrete igrača
- kreirate “zarazne” igre pomoću unosa sa mobilnog uređaja
- dodate prilagođene ikone i opcije prezentacije
- održite interesovanje igrača pomoću Unity paketa obaveštenja za mobilne uređaje
- Integrišete društvene medije u svoje projekte
- funkcionalnošću proširene stvarnosti svoju igru učinite realnijom i zanimljivijom
- Izgradite uzbudljive igre sa naknadnom obradom i partikularnim efektima

ISBN: 978-86-7310-593-2

kompjuter biblioteka
osnovana 1986.
www.kombib.rs

Packt

Unity 2022

Razvoj mobilnih igara

Prevod III izdanja

Izgradnja i publikacija “zaraznih igrica”
za Android i iOS

JOHN P. DORAN

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a
Beograd, Srbija
Tel: 011/2520272
e-pošta: kombib@gmail.com
veb-sajt: www.kombib.rs

Za izdavača:
Mihailo J. Šolajić, direktor

Autor:
JOHN P. DORAN

Prevod: Slavica Prudkov
Lektura: Nemanja Lukić
Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun
Tiraž: 500
Godina izdanja: 2023.

Broj knjige: 570
Izdanje: Prvo
ISBN: 978-86-7310-593-2

Naslov originala:

Unity 2022 Mobile Game Development
Third Edition
ISBN 978-1-80461-372-6

Packt Publishing Ltd.
Birmingham, UK, packt.com

Unity 2022
Razvoj mobilnih igara
Autorizovani prevod sa engleskog jezika.

DEO 1

Podešavanje za igru/razvoj

U ovom delu knjige istražićemo osnovne elemente razvoja Unity igara, sa posebnim fokusom na pripremu za kreiranje mobilnih igara. U poglavljima u ovom delu pružićemo vam neophodno znanje i veštine za podešavanje razvojnog okruženja i vodićemo vas kroz proces izgradnje projekta igre i raspoređivanja na mobilni uređaj.

Do kraja ovog dela imaćete solidnu osnovu znanja o razvoju Unity igara i bićete spremni da pređete na naprednije teme, obrađene u narednim delovima knjige.

Ovaj deo obuhvata sledeća poglavlja:

- *Poglavlje 1, Izgradnja igre*
- *Poglavlje 2, Podešavanje projekta za Android i iOS razvoj*

1

Izgradnja igre

Na početku puta izgradnje mobilnih igara pomoću Unity uređivača je važno da poznajete sam mehanizam, pre nego što uronimo u specifičnosti izgradnje elemenata za mobilne platforme. Iako postoji šansa da ste već izgradili igru i da želite da je prebacite na mobilnu platformu, biće i onih čitaoca koji do sada nisu koristili **Unity,** ili ga možda nisu koristili duže vreme. Ovo poglavlje predstavlja uvod za pridošlice i osveženje za povratnike, a najbolju praksu za one koji su već upoznati sa Unity uređivačem. Iako možete da preskočite ovo poglavlje, ako ste već upoznati sa Unity uređivačem, mislim da je dobra ideja da prođete kroz ceo projekat, da biste znali procese razmišljanja koji stoje iza načina na koji je projekat napravljen, pa da to imate na umu za svoje buduće naslove.

U ovom poglavlju ćemo izgraditi 3D igru beskonačne trke (eng. endless runner) po uzoru na *Temple Run seriju Imangi Studios LLC-a*. U našem slučaju, imaćemo igrača koji će neprekidno trčati u određenom pravcu i izbegavati prepreke koje mu se nađu na putu. Takođe ćemo u igricu stalno dodavati nove elemente, jer možemo lako da dodamo funkcije.

Ovo poglavlje je podeljeno na nekoliko tema. Sadrži jednostavne procese, koje ćete pratiti korak po korak. Sledi skica naših zadataka:

- Postavljanje projekta
- Kreiranje igrača
- Pomeranje igrača kroz C# skript
- Poboljšanje skriptova pomoću atributa i XML komentara
- Funkcija Update u odnosu na funkciju FixedUpdate
- Kamera prati igrača
- Kreiranje osnovne pločice
- Učinite igru beskrajnom
- Kreiranje prepreka

Tehnički zahtevi

U ovoj knjizi koristimo *Unity 2022.1.0b14* i *Unity Hub 3.3.1*, ali postupak bi trebalo da funkcioniše, uz minimalne izmene, i u budućim verzijama uređivača. Ako je izašla nova verzija uređivača, a vi želite da preuzmete verziju koja je korišćena u ovoj knjizi, posetite Unity arhivu za preuzimanje, na adresi `https://unity3d.com/get-unity/download/archive`.

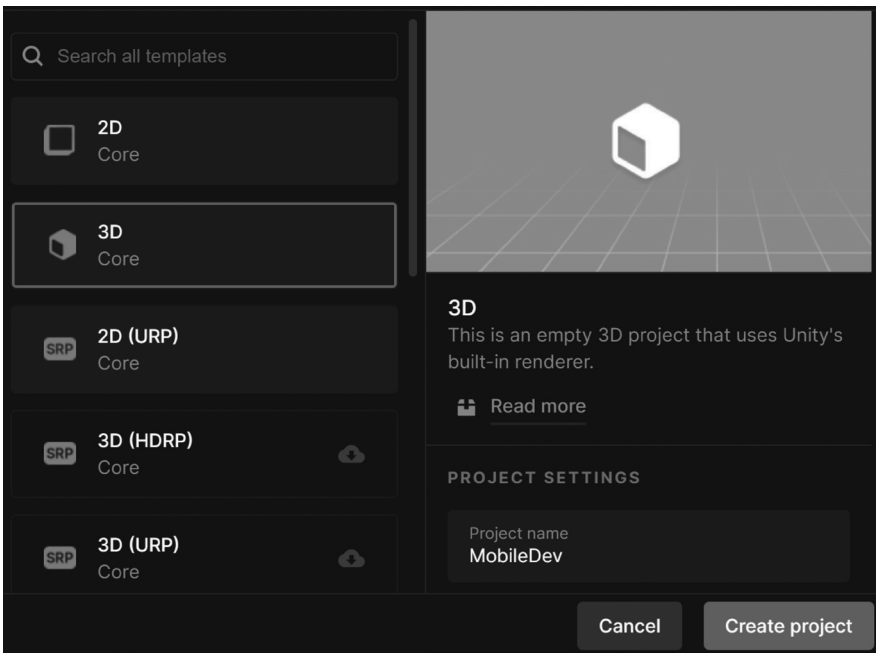
Takođe, sistemski zahtevi za Unity nalaze se na adresi `https://docs.unity3d.com/2022.1/Documentation/Manual/sistem-requirements.html` u odeljku *Unity Editor system requirements*.

Fajlovi koda za ovo poglavlje nalaze se na GitHubu, na adresi `https://github.com/PacktPublishing/Unity-2022-Mobile-Game-Development-3rd-Edition/tree/main/Chapter01`.

Postavljanje projekta

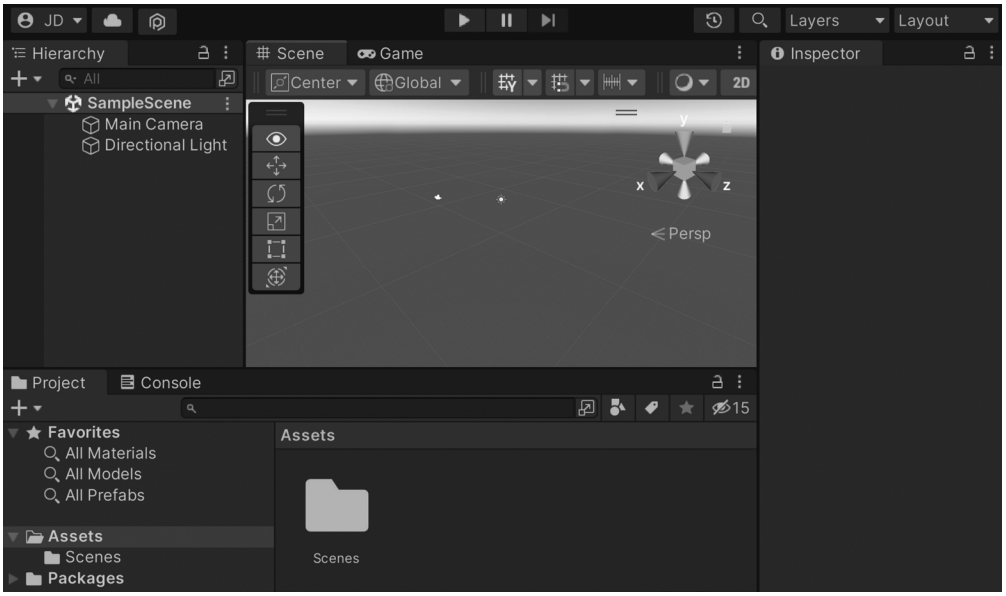
Sada kada imamo na umu naše ciljeve, počnimo izgradnju projekta:

1. Da biste počeli, otvorite Unity Hub na svom računaru.
2. Na početku, kreirate novi projekat klikom na dugme **New**.
3. Pod **Project Name** unesite naziv (ja sam izabrao `MobileDev`) i uverite se da je pod **Templates** izabrana opcija **3D**. Zatim, kliknite na **CREATE** i sačekajte da Unity bude učitan:



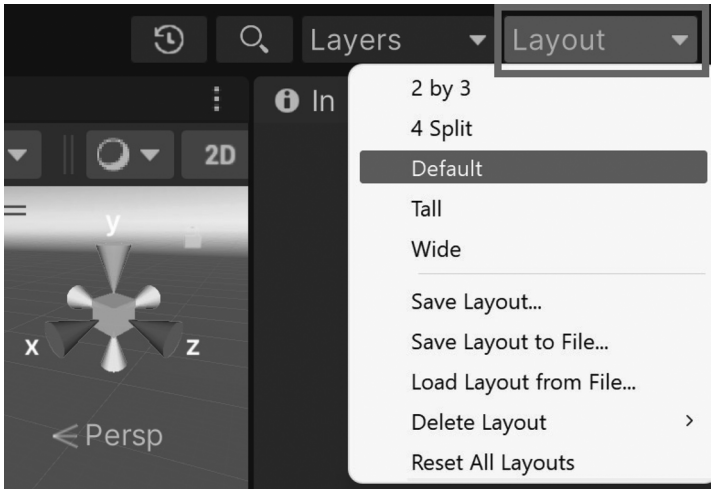
Slika 1.1 – Kreiranje 3D projekta

4. Nakon što je učitavanje završeno, prvi put ćete videti prikazan Unity Editor:



Slika 1.2 – Unity Editor

5. Ako vaš raspored ne izgleda isto kao na prethodnom snimku ekrana, u gornjem desnom delu trake sa alatkama izaberite padajući meni **Layout** i od ponuđenih opcija izaberite **Default**:



Slika 1.3 – Dugme Layout

Sada smo prvi put otvorili Unity i vidimo podrazumevani raspored!

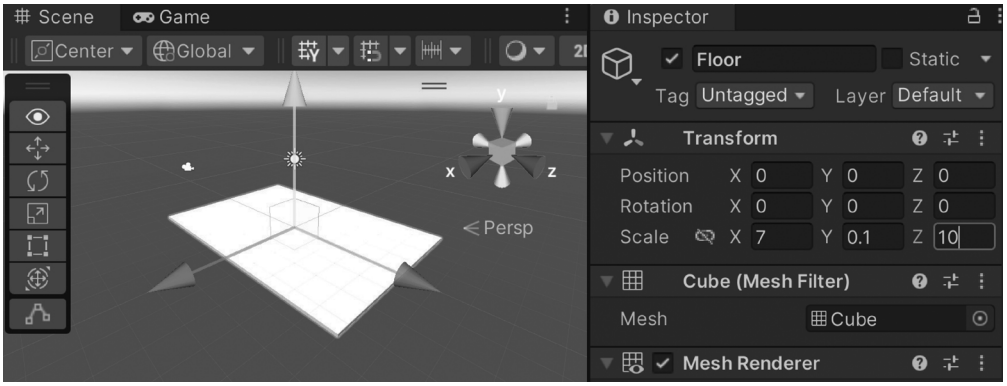
> **Savet**
>
> Ako prvi put koristite Unity, toplo preporučujem da pročitate odeljak *Unity interfejs* u *Unity Manual-u*, kom možete da pristupite na adresi `https://docs.unity3d.com/Manual/UsingTheEditor.html`.

Sada kada imamo otvoren Unity, možemo početi da gradimo projekat.

Kreiranje igrača

Za početak, izgradićemo igrača koji će se uvek kretati napred. Počnimo:

1. Za početak, kreirajte teren po kom će igrač hodati. Da biste to uradili, u gornjem meniju izaberite **GameObject** | **3D Object** | **Cube**.
2. Pređite na prozor **Inspector** i promenite naziv objekta u `Floor`. Zatim, za komponentu **Transform** podesite **Position** na (`0`,`0`,`0`). To možete da uradite upisivanjem vrednosti, ili tako što ćete kliknuti desnim tasterom miša na komponentu **Transform**, a zatim izabrati opciju **Reset Position**.
3. Zatim postavite **Scale** vrednosti objekta na (`7`, `0.1`, `10`):



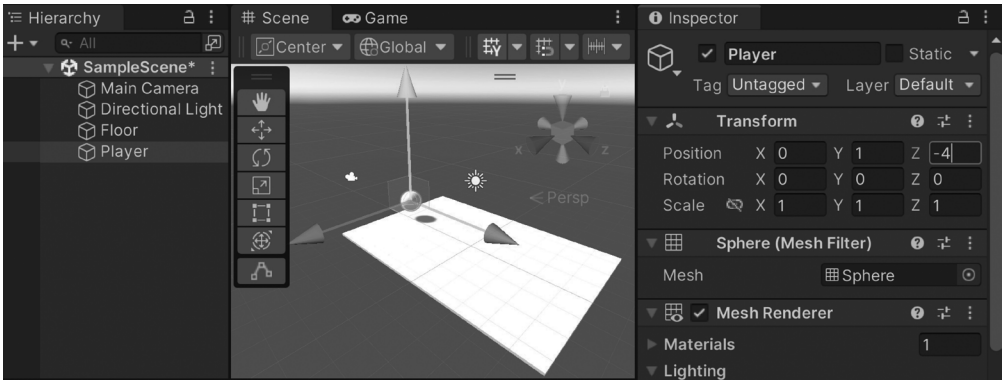
Slika 1.4 – Kreiranje tla

U Unity uređivaču, podrazumevano, `1` jedinica prostora predstavlja 1 metar u stvarnom životu. Dakle, **Scale** vrednosti će učiniti pod dužim nego što je širok (**X** i **Z**) i imamo neku veličinu na zemlji (**Y**), pa će se igrač sudariti i prizemljiti na nju, jer imamo komponentu **Box Collider**, koja je podrazumevano povezana.

Napomena

Kompomenta **Box Collider** je dodata automatski, prilikom kreiranja objekta **Cube,** i potrebno je da se objekti sudare sa njom. Za više informacija o komponenti **Box Collider**, posetite stranicu `https://docs.unity3d.com/Manual/class-BoxCollider.html`.

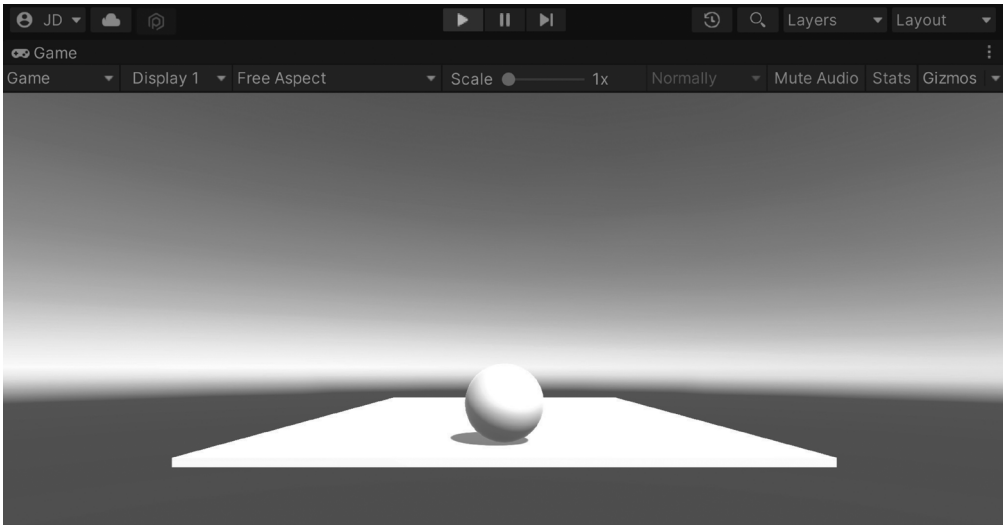
4. Zatim, kreirajte igrača koji će biti sfera. Da biste to uradili kliknite na **GameObject | 3D Object | Sphere**.
5. Promenite naziv sfere na `Player` i postavite vrednost **Position** komponente **Transform** na (`0, 1, -4`):



Slika 1.5 – Pozicioniranje igrača

Na taj način sfera je postavljena malo iznad zemlje i pomerena nazad blizu početne tačke. Imajte na umu da je objekat kamere (pogledajte ikonicu kamere) podrazumevano usmeren ka lopti, jer je pozicioniran na (`0, 1, -1 0`).

6. Želite da se lopta pomera, pa je potrebno da kažete fizičkom mehanizmu da želite da ovaj objekat reaguje na sile, pa je potrebno da dodate komponentu **Rigidbody**. Da biste to uradili, selektujte objekat **Player**, kliknite na meni i izaberite opciju **Component | Physics |Rigidbody**. Da biste videli šta se sada dešava, kliknite na dugme **Play**, koje se nalazi na sredini prve trake sa alatkama:



Slika 1.6 – Trenutno stanje igre

Kao na prethodnom snimku ekrana, trebalo bi da vidite kako lopta pada na zemlju kada pokrenemo igru.

Savet

Možete da onemogućite/omogućite da kartica **Game** zauzima ceo ekran tokom igre klikom na dugme **Maximize On Play** na vrhu, ili kliknite desnim tasterom miša na karticu **Game**, a zatim izaberite opciju **Maximize**.

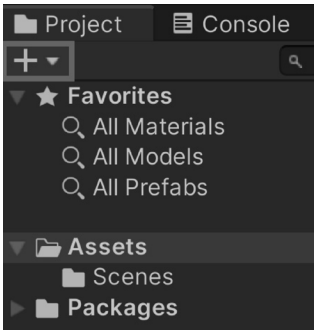
7. Ponovo kliknite na dugme **Play** da biste isključili igru i vratite se na karticu **Scene**, ako se to ne dogodi automatski.

   Sada imamo objekte i za pod i za igrača u igri i rekli smo igraču da reaguje na fiziku! Sada ćemo, pomoću koda, igraču dodati interaktivnost.

Pomeranje igrača kroz C# skript

Želimo da se igrač pomera pa, da bismo to uradili, kreiramo sopstvenu funkcionalnost u skriptu, efektivno kreirajući sopstvenu prilagođenu komponentu u procesu:

1. Da biste kreirali skript, pređite na prozor **Project** i izaberite dugme **Create** u gornjem levom uglu menija, klikom na ikonicu **+**, a zatim izaberite opciju **Folder**:



Slika 1.7 – Lokacija ikonice +

Savet

Takođe, meniju **Create** možete da pristupite desnim klikom na desnu stranu prozora **Project**. Ovim metodom, kliknete desnim tasterom miša, a zatim izaberete opciju **Create** | **Folder**.

2. U tom prozoru, direktorijumu dodelite naziv `Scripts`. Uvek je dobra ideja da organizujete projekte, a ovo će pomoći u tome.

Savet

Ako slučajno pogrešno napišete naziv stavke u prozoru **Project**, možete ga preimenovati tako što ćete kliknuti desnim tasterom miša i izabrati opciju **Rename** ili tako što ćete selektovati objekat, a zatim kliknuti na naziv.

3. Dvostruko kliknite na direktorijum da biste ga otvorili, kreirajte skript tako što ćete kliknuti na **Create** | **C# Script** i promenite naziv novokreirane stavke na `PlayerBehaviour` (bez razmaka).

Napomena

Koristim *bahaviour* umesto *behavior* jer su sve komponente u Unity-u potomci druge klase, pod nazivom `MonoBehaviour` i u tom pogledu pratim smernice Unity-a.

4. Dvostruko kliknite na skript da biste otvorili uređivač skripta (IDE) po vašem izboru i dodajte mu sledeći kod:

```
using UnityEngine;

public class PlayerBehaviour : MonoBehaviour
{

```

```
    // A reference to the Rigidbody component
    private Rigidbody rb;

    // How fast the ball moves left/right
    public float dodgeSpeed = 5;

    // How fast the ball moves forward  automatically
    public float rollSpeed = 5;

    // Start is called before the first frame update
    void Start()
    {
        // Get access to our Rigidbody component
        rb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void Update()
    {
        // Check if we're moving to the side
        var horizontalSpeed =
            Input.GetAxis("Horizontal") * dodgeSpeed;

        rb.AddForce(horizontalSpeed, 0, rollSpeed);

    }
}
```

U prethodnom kodu imamo nekoliko promenljivih koje ćemo koristiti. Promenljiva `rb` je referenca za GameObject komponentu `Rigidbody`, koju smo prethodno dodali. Daje nam mogućnost da pomerimo objekat, što ćemo koristiti u funkciji `Update`. Takođe, imamo dve promenljive, `dodgeSpeed` i `rollSpeed`, koje diktiraju brzinu igrača, kada se kreće levo/desno ili napred.

Pošto naš objekat ima samo jednu komponentu `Rigidbody`, dodeljujemo promenljivu `rb` jednom u funkciji `Start`, koja je pozvana kada je GameObject učitan u scenu na početku igre.

Zatim, koristimo funkciju `Update` za proveru da li naš igrač pritiska tastere da bi se pomerio levo ili desno, na osnovu Unity sistema **Input Manager**. Podrazumevano, funkcija `Input.GetAxis` će nam vratiti negativnu vrednost, pomeranjem na `-1` ako pritisnemo *A* ili strelicu ulevo. Ako pritisnemo strelicu udesno ili *D*, dobićemo pozitivnu vrednost do `1`, a unos će se pomeriti prema `0` ako ništa nije pritisnuto. Zatim, množimo tu vrednost sa vrednošću promenljive `dodgeSpeed` da bismo povećali brzinu i lakše videli kretanje objekta.

Napomena

Za više informacija o sistemu Input Manager, pogledajte stranicu `https://docs.unity3d.com/Manual/class-InputManager.html`.

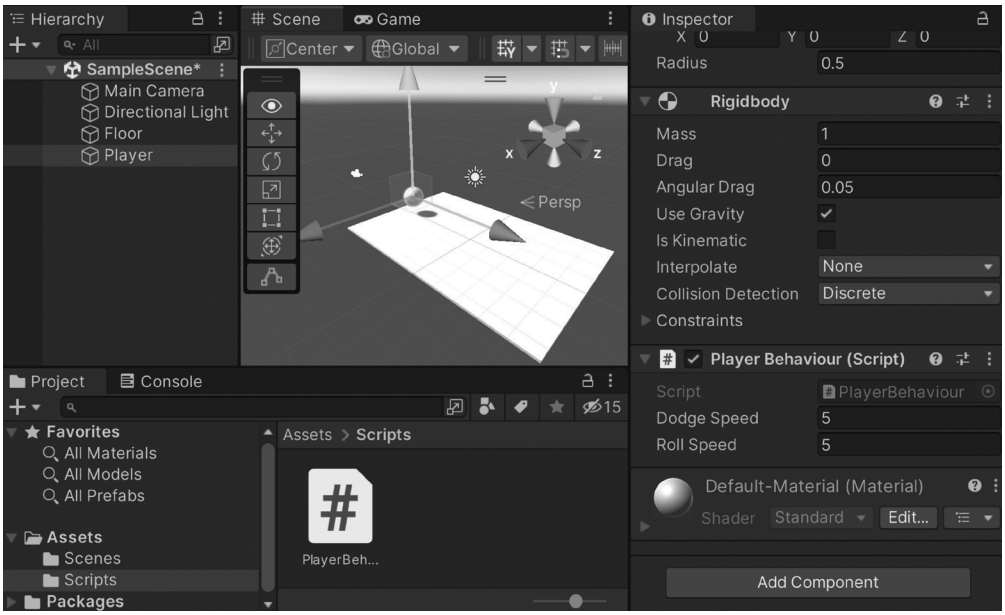
Konačno, kada imamo tu vrednost, primenićemo silu na `horizontalSpeed` jedinice na *X* osi i na `rollSpeed` na *Z* osi, za loptu.

5. Sačuvajte skript i vratite se u Unity Editor.
6. Sada je potrebno da dodelite ovaj skript igraču tako što ćete izabrati objekat `Player` u prozoru **Hierarchy**, a zatim, u prozoru **Inspector**, prevucite i otpustite skript `PlayerBehaviour` iz prozora **Project** tako da bude povrh objekta `Player`.

Napomena

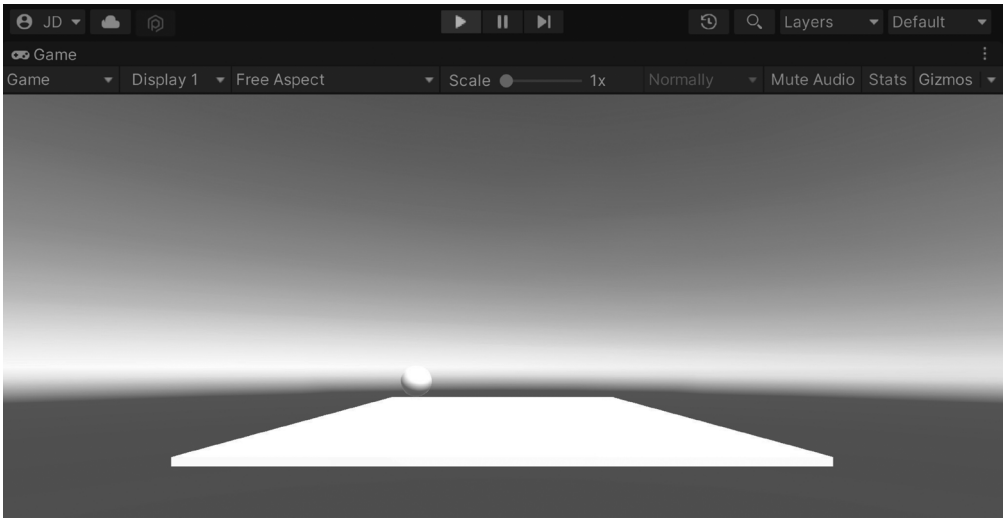
Imajte na umu da, kada pišemo skriptove, ako deklarišemo promenljivu kao `public`, ona će biti prikazana u prozoru **Inspector**, da bismo mogli da je podesimo. Obično postavljamo promenljivu kao `public` kada želimo da dizajneri podese vrednosti za potrebe igranja, ali to, takođe, dozvoljava drugim skriptovima da pristupe svojstvu u kodu. Podrazumevano, promenljive i metodi su `private`, što znači da mogu da budu korišćeni samo u okviru klase. Za više informacija o modifikatorima pristupa posetite stranicu `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/access-modifiers`.

Ako sve prođe dobro, trebalo bi da vidite da je skript prikazan na objektu, na sledeći način:



Slika 1.8 – Dodata komponenta PlayerBehaviour

7. Sačuvajte scenu tako što ćete kliknuti na **File** | **Save**. Nakon toga, pokrenite igru i koristite strelice levo i desno da biste videli kako se igrač kreće u skladu sa vašim unosom, a bez obzira na sve podrazumevano se kreće unapred:



Slika 1.9 – Trenutno stanje igre

Sada možete da vidite da se lopta automatski kreće, a naš unos biva ispravno primljen

Poboljšanje skriptova pomoću atributa i XML komentara

Sada bismo mogli da prestanemo da koristimo skript klase `PlayerBehaviour`, ali želim da objasnim kako možemo da poboljšamo kvalitet i stil koda. To postaje posebno korisno kada gradite projekte u timovima. Pošto ćete raditi sa drugim ljudima, neki od njih će raditi na kodu sa vama. Zatim, postoje dizajneri i umetnici koji neće raditi na kodu sa vama, ali će ipak koristiti elemente koje ste programirali.

Kada pišemo skriptove želimo da budu što otporniji na greške. Taj proces započinjemo tako što ćemo promenljivu `rb` podesiti kao `private`, pa sada korisnik neće moći da je izmeni van te klase. Želimo da naši saradnici modifikuju promenljive `dodgeSpeed` i `rollSpeed`, a možda želimo i da im damo neke savete o tome šta je to i/ili kako će biti korišćeno. Da bismo to uradili u prozoru **Inspector**, možemo da iskoristimo ono što nazivamo **atribut**.

Korišćenje atributa

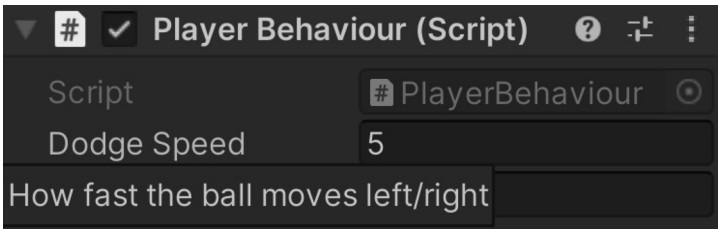
Atributi su elementi koje možemo da dodamo na početak deklaracije promenljive, klase ili funkcije, što nam omogućava da im priložimo dodatnu funkcionalnost. Postoji mnogo atributa unutar Unity-a, ali možemo da napišemo i sopstvene atribute, a sada ćemo govoriti o onima koje ja najčešće koristim.

Atribut Tooltip

Ako ste neko vreme koristili Unity, možda ste primetili da neke komponente u prozoru **Inspector**, kao što je `Rigidbody`, imaju lepu funkciju — ako pređete mišem preko naziva promenljive videćete opis šta su promenljive i/ili kako da ih koristite. Prvo ćete naučiti kako možemo postići isti efekat u sopstvenim komponentama, pomoću atributa `Tooltip`. Ako to uradimo za promenljive `dodgeSpeed` i `rollSpeed`, izgledaće otprilike ovako:

```
[Tooltip("How fast the ball moves left/right")]
public float dodgeSpeed = 5;

[Tooltip("How fast the ball moves forward  automatically")]
public float rollSpeed = 5;
```

Sačuvajte prethodni skript i vratite se u uređivač:



Slika 1.10 – Primer atributa Tooltip

Sada, kada označimo promenljivu pomoću miša i ostavimo kursor preko stavke, biće prikazan tekst koji smo postavili. To je odlična navika, jer će vaši saradnici znati čemu služe vaše promenljive, bez potrebe da pregledaju sam skript.

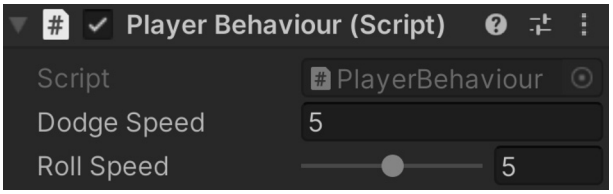
> **Napomena**
>
> Za više informacija o atributu `Tooltip` posetite stranicu `https://docs.unity3d.com/ScriptReference/TooltipAttribute.html`.

Atribut Range

Još jedan atribut, koji možemo da koristimo da zaštitimo kod, je atribut `Range`. On nam omogućava da navedemo minimalnu i maksimalnu vrednost promenljive. Pošto želimo da se igrač uvek kreće napred, možda bismo želeli da ograničimo igračevo kretanje unazad. Da bismo to uradili, možemo da dodamo sledeću istaknutu liniju koda:

```
[Tooltip("How fast the ball moves forward  automatically")]
[Range(0, 10)]
public float rollSpeed = 5;
```

Sačuvajte skript i vratite se u uređivač:



Slika 1.11 – Primer atributa Range

Sada smo dodali klizač pored naše vrednosti i možemo ga prevlačiti da bismo podesili vrednost između minimalne i maksimalne. To ne samo da štiti promenljivu od promene u nevažeće stanje, već čini da dizajneri mogu lako da podese vrednosti povlačenjem klizača.

Atribut RequireComponent

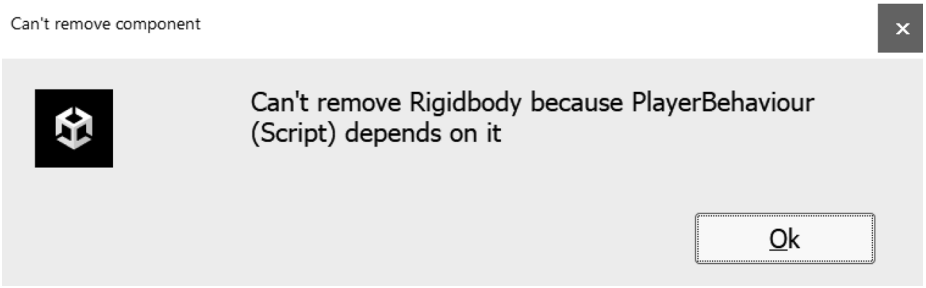
Trenutno, koristimo komponentu `Rigidbody` da bismo kreirali skript. Kada ste član tima, iako drugi članovi možda i ne čitaju vaše skriptove i dalje se očekuje da će ih koristiti prilikom kreiranja igre. Nažalost, to znači da mogu da urade nešto što će imati neželjene rezultate, kao što je uklanjanje komponente `Rigidbody`, što bi uzrokovalo greške kada je skript pokrenut. Srećom, imamo atribut `RequireComponent`, koji možemo da upotrebimo da bismo to sprečili.

Izgleda otprilike ovako:

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class PlayerBehaviour : MonoBehaviour
```

Dodavanjem ovog atributa navodimo da kada uključimo ovu komponentu u GameObject, a ona nema priključenu komponentu `Rigidbody` na svoj objekat GameObject, komponenta će biti dodata automatski. Takođe, ako pokušamo da uklonimo komponentu `Rigidbody` iz ovog objekta uređivač će nas upozoriti da to nije moguće, osim ako prvo ne uklonimo komponentu `PlayerBehaviour`. Imajte na umu da to funkcioniše za bilo koju klasu koja je proširena iz klase `MonoBehaviour`; samo zamenite `Rigidbody` onim što želite da zadržite.

Sada, ako otvorimo Unity Editor i pokušamo da uklonimo komponentu `Rigidbody` desnim klikom na nju u prozoru **Inspector** i selektovanjem opcije **Remove Component**, biće prikazana sledeća poruka:



Slika 1.12 – Prozor Can't remove component

To je upravo ono što želimo, a to osigurava da će komponenta biti prisutna, pa nećemo morati da dodajemo `if` provere svaki put kada želimo da koristimo komponentu.

Imajte na umu da ranije nismo koristili atribut `Tooltip` u privatnoj promenljivoj `rb`. Pošto nije prikazan u uređivaču, nije ni potreban. Međutim, to možemo poboljšati pomoću XML komentara.

XML komentari

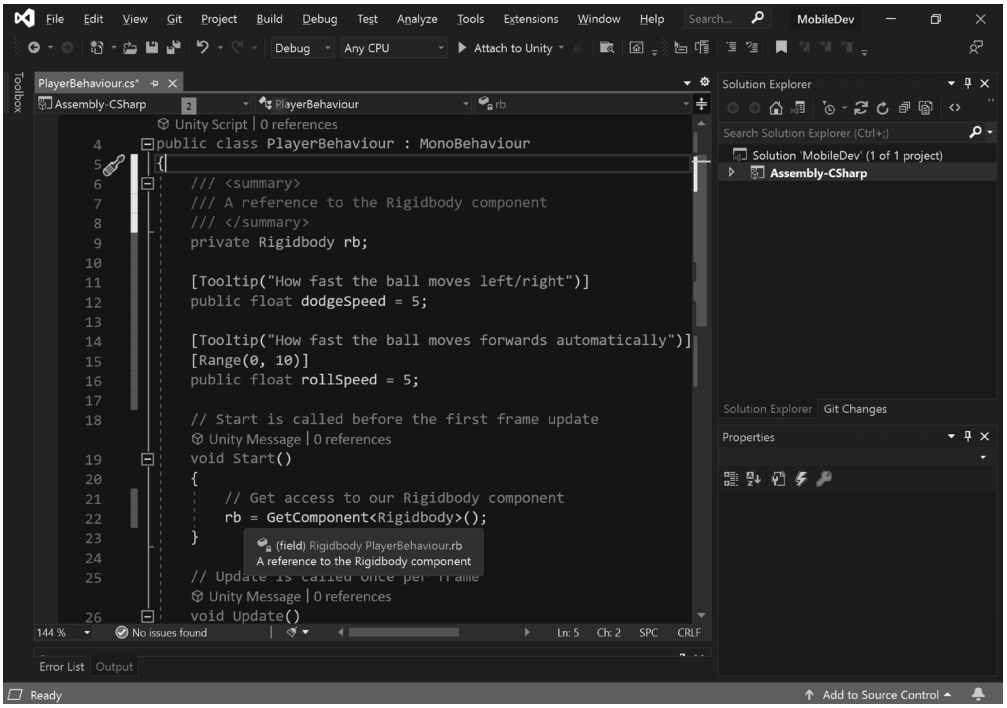
XML komentarima možemo postići nekoliko korisnih stvari, što nismo mogli tradicionalnim komentarima koje smo ranije koristili. Kada koristimo promenljive/funkcije umesto koda u uređivaču Visual Studio, sada ćemo videti komentar o tome. Te dodatne informacije i detalji će ostalim koderima iz vašeg tima pomoći da ispravno koriste vaš kod.

XML komentari izgledaju otprilike ovako:

```
/// <summary>
/// A reference to the Rigidbody component
/// </summary>
private Rigidbody rb;
```

Možda se čini da je za korišćenje ovog formata potrebno mnogo više pisanja, ali ja zapravo nisam otkucao sve. XML komentari su standardna funkcija jezika C#, pa ako koristite MonoDevelop ili Visual Studio i unesete `///`, akcija će automatski generisati blokove rezimea za vas (i potrebne oznake `param`, ako postoje parametri potrebni za nešto kao što je funkcija).

Zašto bismo to želeli da radimo? Pa, ako selektujemo promenljivu u IntelliSense-u, ona će nam prikazati sledeće informacije:



Slika 1.13 – Primer kratkog opisa iz XML komentara

To je velika pomoć kada drugi ljudi koriste vaš kod i na taj način Unity osoblje piše svoj kod. To, takođe, možemo proširiti na funkcije i klase da bismo osigurali da je kod dodatno samo-dokumentovan.

Nažalost, XML komentari nisu prikazani u prozoru Inspector i atribut `Tooltip` ne možete upotrebljavati za neke aspekte projekata, kao što su funkcije. Imajući to na umu, ja koristim atribut `Tooltip` za javna uputstva i/ili za nešto što će biti prikazano u prozoru **Inspector**, a XML komentare za sve ostalo.

> **Napomena**
>
> Ako ste zainteresovani za više informacija o XML komentarima, slobodno pogledajte stranicu `https://msdn.microsoft.com/en-us/library/b2s063f7.aspx`.

Sada kada smo videli načine za poboljšanje formatiranja koda, pogledajmo kako možemo da poboljšamo performanse pomoću funkcija `Update` koje pruža Unity.

Funkcija Update u odnosu na funkciju FixedUpdate

Sledeće što je potrebno da pregledamo je kod kretanja. Možda ste primetili da trenutno koristimo funkciju `Update` da bismo pomerili našeg igrača. Kako se u komentaru iznad navodi, funkcija `Update` je pozvana jednom po frejmu koji igra pokreće. Ono što je potrebno uzeti u obzir je da je učestalost pozivanja funkcije `Update` promenljiva, što znači da se vremenom može promeniti. To zavisi od brojnih faktora, uključujući hardver koji je u upotrebi. To znači da što je više puta pozvana funkcija `Update`, to je računar bolji. Želimo dosledno iskustvo za sve naše igrače, a jedan od načina da to postignemo je funkcija `FixedUpdate`.

Funkcija `FixedUpdate` je slična funkciji `Update`, sa nekim ključnim razlikama. Prva je da se ova funkcija poziva u fiksnim vremenskim koracima, što znači isto vreme između poziva. Takođe, važno je napomenuti da se fizički proračuni vrše nakon pozivanja funkcije `FixedUpdate`. To znači da objekti zasnovani na fizici, koji modifikuju kod, treba da budu izvršeni unutar funkcije `FixedUpdate`, osim jednokratnih događaja, kao što je skakanje:

```
/// <summary>
/// FixedUpdate is a prime place to put physics
/// calculations happening over a period of time.
/// </summary>

void FixedUpdate()
{
    // Check if we're moving to the side
    var horizontalSpeed = Input.GetAxis("Horizontal") *
                          dodgeSpeed;

    rb.AddForce(horizontalSpeed, 0, rollSpeed);

}
```

Prilagođavanjem koda za upotrebu funkcije `FixedUpdate`, brzina kretanja lopte bi trebalo da bude mnogo ravnomernija.

> **Napomena**
>
> Za više informacija o funkciji `FixedUpdate`, posetite stranicu `https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html`.

Spajanje

Uz sve ovo o čemu smo govorili, sada možemo da imamo konačnu verziju skripta, koji izgleda ovako:

```
using UnityEngine;

/// <summary>
/// Responsible for moving the player automatically and
/// receiving input.
/// </summary>
[RequireComponent(typeof(Rigidbody))]
public class PlayerBehaviour : MonoBehaviour
{
    /// <summary>
    /// A reference to the Rigidbody component
    /// </summary>
    private Rigidbody rb;

    [Tooltip("How fast the ball moves left/right")]
    public float dodgeSpeed = 5;

    [Tooltip("How fast the ball moves
        forward  automatically")]
    [Range(0, 10)]
    public float rollSpeed = 5;

    // Start is called before the first frame update
    public void Start()
    {
        // Get access to our Rigidbody component
        rb = GetComponent<Rigidbody>();
    }

    /// <summary>
    /// FixedUpdate is a prime place to put physics
    /// calculations happening over a period of time.
    /// </summary>

    void FixedUpdate()
    {
        // Check if we're moving to the side
        var horizontalSpeed = Input.GetAxis("Horizontal") *
                              dodgeSpeed;
```

```
            rb.AddForce(horizontalSpeed, 0, rollSpeed);

        }
    }
```

Nadam se da se i vi slažete da ovo čini kod lakšim za razumevanje i korišćenje. Sada možemo da pređemo na dodatne funkcije igre!

Kamera prati igrača

Trenutno naša kamera ostaje na istom mestu dok igra traje. To ne funkcioniše baš najbolje za ovu igru, pošto se igrač kreće dok igra traje. Postoje dva glavna načina da pomeramo kameru. Možemo samo da pomeramo kameru i učinimo je potomkom igrača, ali to neće funkcionisati zbog činjenice da bi kamera imala istu rotaciju kao i lopta, što bi prouzrokovalo da se kamera stalno okreće i verovatno bi izazvala vrtoglavicu i dezorijentaciju igrača. Zbog toga ćemo verovatno želeti da koristimo skript da bismo je pomerali. Srećom, možemo prilično lako da modifikujemo kameru, pa učinimo to odmah:

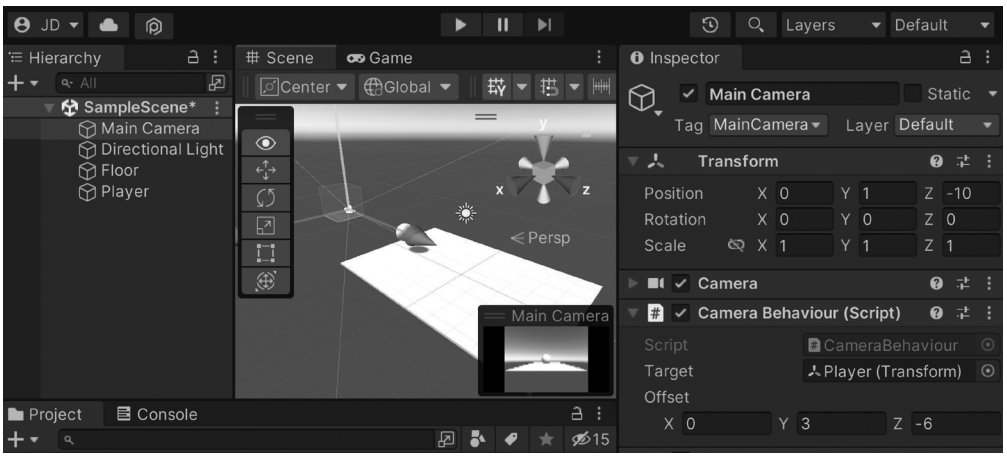
1. U prozoru Project kreirajte nov C# skript, pod nazivom `CameraBehaviour`. Koristite sledeći kod:

```
using UnityEngine;

/// <summary>
/// Will adjust the camera to follow and face a target
/// </summary>
public class CameraBehaviour : MonoBehaviour
{
    [Tooltip("What object should the camera be looking
        at")]
    public Transform target;

    [Tooltip("How offset will the camera be to the
        target")]
    public Vector3 offset = new Vector3(0, 3, -6);

    /// <summary>
    /// Update is called once per frame
    /// </summary>
    private void Update()
    {
        // Check if target is a valid object
        if (target != null)
        {
```

```
            // Set our position to an offset of our
            // target
            transform.position = target.position +
                offset;

            // Change the rotation to face target
            transform.LookAt(target);
        }
    }
}
```

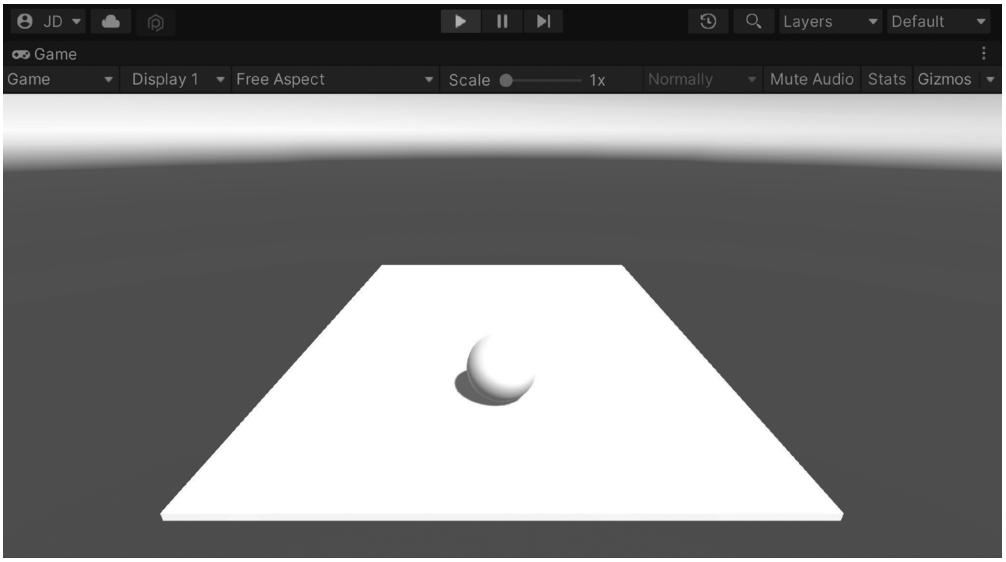
Ovaj skript će postaviti poziciju objekta za koji je vezan na poziciju cilja sa pomakom. Nakon toga, promeniće rotaciju objekta tako da bude okrenut prema cilju. Oba parametra su označena kao `public`, tako da mogu da budu podešeni u prozoru **Inspector**.

2. Sačuvajte skript i vratite se u Unity Editor. Izaberite objekat **Main Camera** u prozoru **Hierarchy**. Zatim, u prozoru **Inspector** dodajte objektu komponentu `CameraBehaviour`. To možete da uradite tako što ćete prevući i otpustiti skript iz prozora **Project** na GameObject ili tako što ćete kliknuti na dugme **Add Component** na dnu prozora **Inspector**, uneti naziv komponente, a zatim pritisnuti *Enter* da biste potvrdili kada je istaknuta.
3. Nakon toga, prevucite i otpustite objekat `Player` iz prozora **Hierarchy** u svojstvo **Target** skripta, u prozoru **Inspector**:



Slika 1.14 – Podešavanje komponente CameraBehaviour

4. Sačuvajte scenu i igrajte igru:



Slika 1.15 – Trenutno stanje igre

Kamera sada prati igrača dok se kreće. Slobodno podesite promenljive i vidite kako to utiče na izgled kamere, da biste dobili osećaj koji biste najviše želeli za projekat. Nakon toga, možemo da imamo mesto za kretanje lopte, o čemu ćemo govoriti u sledećem odeljku.

Izgradnja osnovne pločice

Želimo da naša igra bude beskonačna, a da bismo to postigli potrebno je da imamo delove koje možemo da širimo da bismo izgradili okruženje; uradimo to sada:

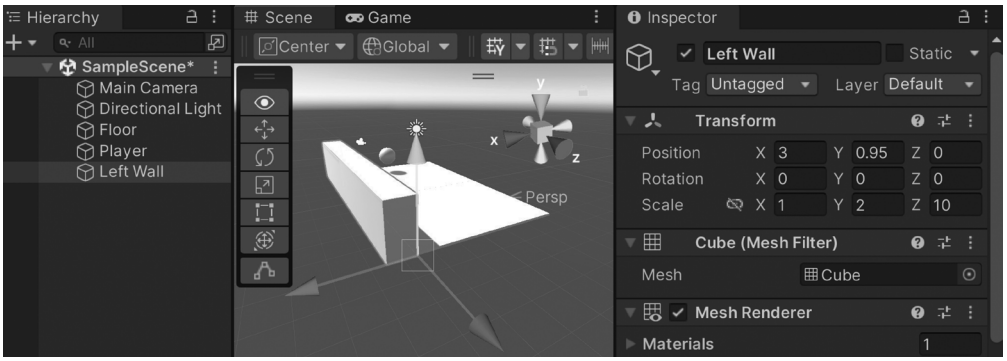
1. Prvo je potrebno da kreirate jedan ponovljivi deo za igru trkača. Da biste to uradili, dodajte nekoliko zidova na pod koji već imamo. U prozoru **Hierarchy** izaberite objekat **Floor** i duplirajte ga pritiskom na *Ctrl+D* u Windows-u ili *Command+D* u macOS-u. Promenite naziv novog objekta u `Left Wall`.
2. Promenite komponentu **Transform** objekta **Left Wall** podešavanjem vrednosti **Scale** na (`1,2,10`). Selektujte alatku **Move** klikom na dugme sa strelicama na preklopu alata ili pritiskom tastera *W*.

Napomena

Nedavni dodatak Unity-u je koncept **Overlay**, koji je zamenio originalnu traku sa alatkama. Za više informacija o njima i kako da ih koristite, posetite stranicu `https://docs.unity3d.com/2022.1/Documentation/Manual/overlays.html`.

Za više informacija o ugrađenim prečicama Unity-a posetite stranicu `https://docs.unity3d.com/Manual/UnityHotkeys.html`.

3. Želimo da se ovaj zid poklopi sa podom, pa držite pritisnut taster *V* za ulazak u režim *Vertex Snap*. U režimu Vertex Snap, možete da izaberete bilo koji od vrhova na mreži i da ga pomerite u istu poziciju drugog vrha na drugom objektu. To je zaista korisno kao osiguranje da između objekata nema rupa.
4. U režimu Vertex Snap izaberite unutrašnju ivicu i prevlačite je dok ne udari u ivicu poda. Alternativno, možete da podesite vrednosti **Position** na (`3, 0, 95, 0`):

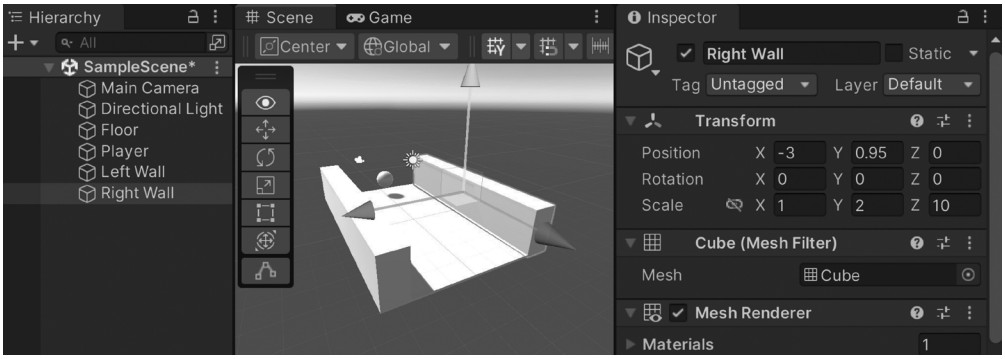


Slika 1.16 – Podešavanje objekta Left Wall

Napomena

Za više informacija o kretanju objekata kroz scenu, uključujući više detalja o režimu Vertex Snap, posetite stranicu `https://docs.unity3d.com/Manual/PositioningGameObjects.html`.

5. Zatim, duplirajte ovaj zid i postavite drugi objekat na drugu stranu (`-3, 0, 95, 0`) i dodelite mu naziv `Right Wall`:



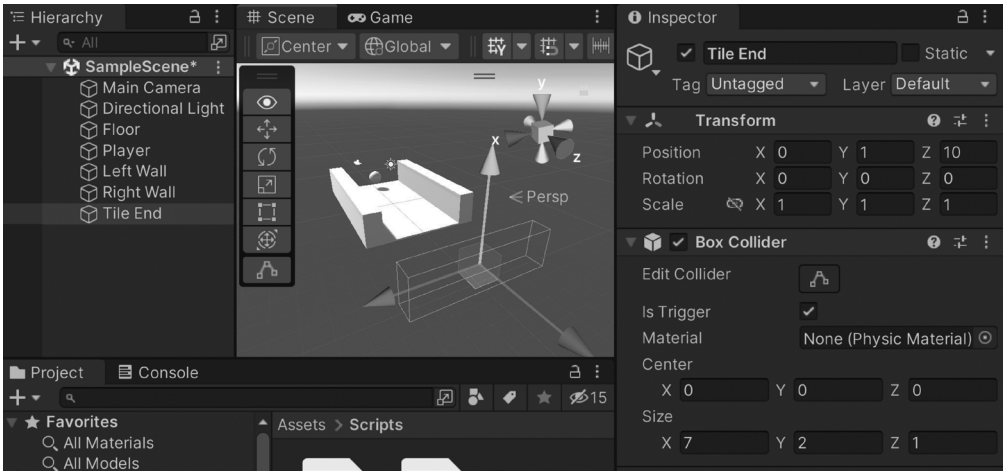
Slika 1.17 – Postavljanje objekta Right Wall

Kao što vidite na prethodnom snimku ekrana, sada štitimo igrača od pada sa leve i desne ivice oblasti za igru. Zbog načina na koji su zidovi postavljeni, ako pomerimo objekat `Floor` biće pomereni i zidovi.

> **Napomena**
>
> Za informacije o pomeranju Unity kamere ili navigaciji do prikaza **Scene** posetite stranicu `https://docs.unity3d.com/Manual/SceneViewNavigation.html`.

Zbog načina na koji je ova igra dizajnirana, nakon što lopta prođe preko jedne pločice, nije potrebno da pločica ostane na toj poziciji. Ako je ostavimo, igra će vremenom postati sporija, jer ćemo imati mnogo elemenata u okruženju igre koji koriste memoriju, pa je dobra ideja da uklonimo elemente koje više ne koristimo. Takođe, potrebno je da imamo neki način da otkrijemo kada bi trebalo da dodamo nove pločice da bismo nastavili putanju kojom igrač može da se kreće.

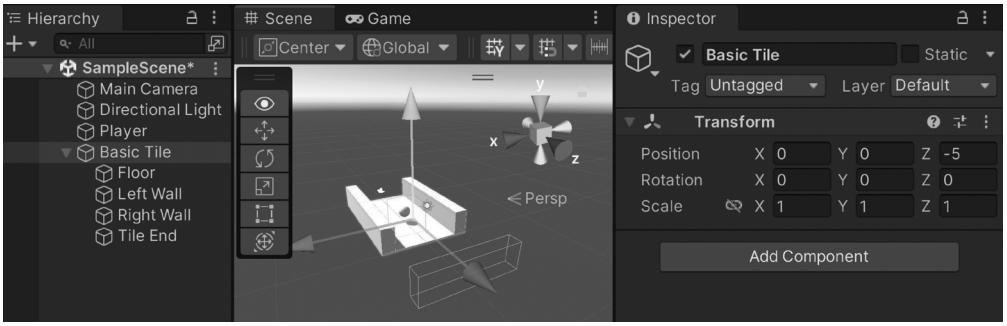
6. Sada, takođe, želimo da znamo gde se ovaj deo završava, pa ćemo dodati objekat koji sadrži okidač sudarača. Selektujte opciju **GameObject** | **Create Empty** i objektu dodelite naziv `Tile End`.
7. Zatim, dodajte komponentu **Box Collider** objektu **Tile End**. U prozoru **Inspector**, pod komponentom **Box Collider**, podesite vrednosti **Scale** na (`7`, `2`, `1`) tako da odgovara veličini prostora kojim igrač može da se kreće. Vidite da postoji zeleni okvir oko tog prostora, koji pokazuje gde može doći do sudara. Podesite svojstvo **Position** na (`0`, `1`, `10`) da biste dosegli kraj pločice. Na kraju, proverite svojstvo **Is Trigger** tako da će mehanizam kolizije pretvoriti sudarač (eng. collider) u okidač (eng. trigger), koji će moći da pokrene događaje koda kada je udaren, ali neće sprečiti igrača da prolazi kroz njega:



Slika 1.18 – Natpis

Kao što sam ranije ukratko pomenuo, ovaj okidač služi da saopšti igri da je igrač završio prelazak preko ove pločice. Postavljen je iza pločice, zbog činjenice da želimo da i dalje vidimo pločice dok ne prođu ono što kamera može da vidi. Govorimo mehanizmu da ukloni ovu pločicu iz igre, ali ćemo o tome detaljnije govoriti kasnije u poglavlju.

8. Sada kada ste kreirali sve objekte, želite da grupišete objekte kao jedan deo za koji možete da kreirate duplikate. Da biste to uradili, kreirajte praznu instancu GameObject tako što ćete kliknuti na **GameObject** | **Create Empty** i novom objektu dodelite naziv `Basic Tile`. Podesite vrednosti **Position** ovog novog objekta na (`0`, `0`, `0`).
9. Zatim, kliknite na prozor **Hierarchy** i u njega prevucite i otpustite objekte **Floor**, **Tile End**, **Left Wall** i **Right Wall** tako da ih učinite potomcima objekta **Basic Tile**.
10. Trenutno, kamera može da vidi početak pločica, pa da biste to ispravili, postavite vrednosti **Basic Tile Position** na (`0`, `0`, `-5`). Kao što vidite na sledećem snimku ekrana, sada će cela pločica biti pomerena unazad:

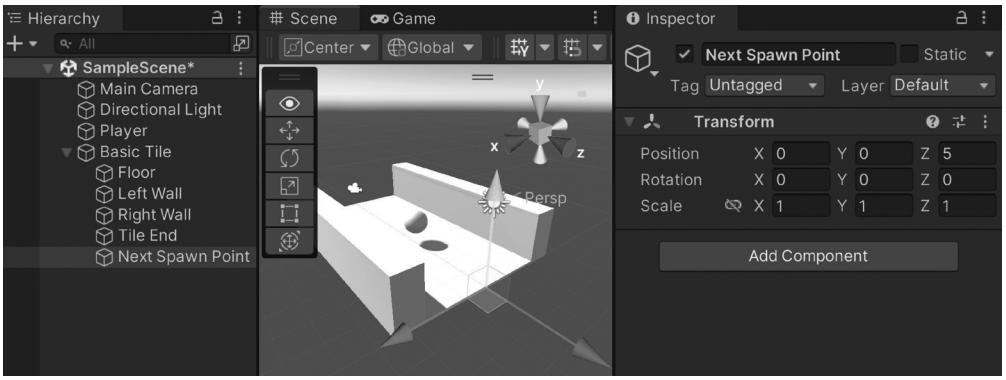


Slika 1.19 – Pomeranje pločice unazad

11. Konačno, potrebno je da znate na kojoj poziciji bi trebalo da kreirate sledeći deo, pa kreirajte još jedan prazan GameObject tako što ćete kliknuti na **GameObject | Create Empty** ili pritisnite tastere *Ctrl+Shift+N*. Novi objekat učinite potomkom objekta **Basic Tile**, dodelite mu naziv `Next Spawn Point` i postavite njegove vrednosti **Position** na (`0`, `0`, `5`).

> **Napomena**
>
> Imajte na umu da kada modifikujemo objekat koji ima roditeljski objekat, pozicija je relativna u odnosu na roditeljski objekat, a ne njegovu poziciju.

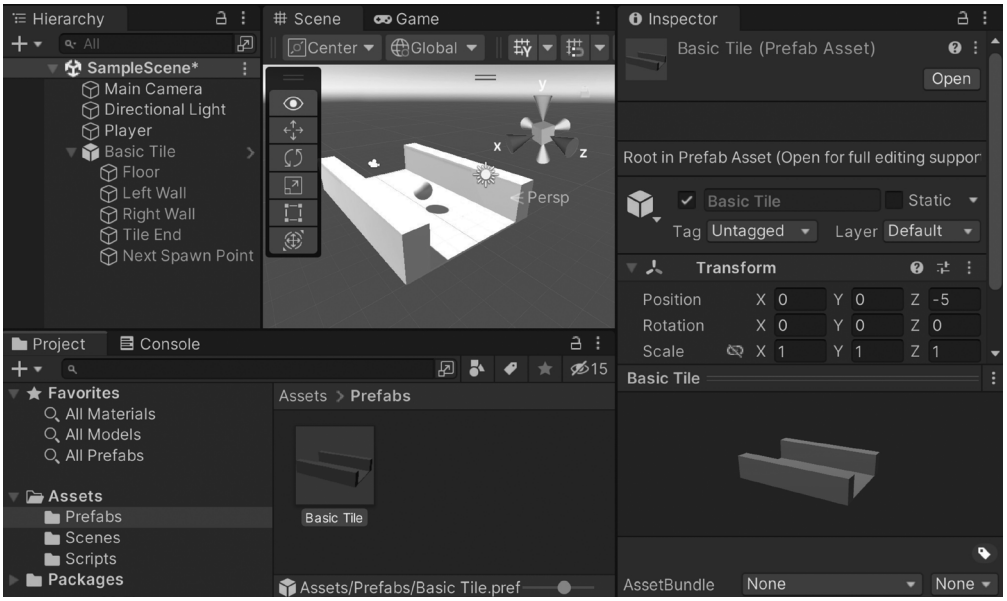
Kao što vidite, pozicija tačke proširenja će sada biti na ivici aktuelne pločice:



Slika 1.20 – Next Spawn Point pozicija

12. Sada imamo jednu pločicu koja je u potpunosti završena. Umesto da je više puta dupliramo ručno, koristićemo Unity koncept **Prefabs**. Prefab, ili prefabrikovani objekti, su nacrti objekata GameObjects i komponenti, koje možemo da pretvorimo u fajlove i koji mogu da budu duplirani. Postoje i druge zanimljive funkcije koje imaju Prefab objekti, ali ćemo o njima govoriti kad ih budemo koristili.

    U prozoru **Project** otvorite direktorijum **Assets**, a zatim kreirajte novi direktorijum, pod nazivom `Prefabs`. Zatim, prevucite i otpustite objekat **Basic Tile** iz prozora **Hierarchy** u prozor **Project** unutar direktorijuma **Prefabs**. Ako je tekst za naziv objekta **Basic Tile** u prozoru **Hierarchy** postao plave boje, znajte da je ispravno kreiran:



Slika 1.21 – Kreiran je Basic Tile Prefab

Sada imamo prefab pločice za koju možemo da kreiramo duplikate pomoću koda, da bismo proširili okruženje.

Beskonačna igra

Sada kada imamo osnovu, učinimo da možemo da nastavimo da trčimo umesto da se zaustavimo nakon kratkog vremena, tako što ćemo postaviti kopije ove osnovne pločice jednu ispred druge:

1. Za početak, imate prefab, tako da možete da izbrišete originalni objekat **Basic Tile** u prozoru **Hierarchy**, tako što ćete ga selektovati, a zatim pritisnuti taster *Delete*.
2. Potrebno je da imate mesto za kreiranje svih ovih pločica i potencijalno upravljanje informacijama za igru, kao što je rezultat igrača. U Unity-u to obično nazivamo `GameManager`. U prozoru **Project** kliknite na direktorijum **Scripts** i kreirajte novi C# skript pod nazivom `GameManager`.
3. Otvorite skript u svom IDE-u i koristite sledeći kod:

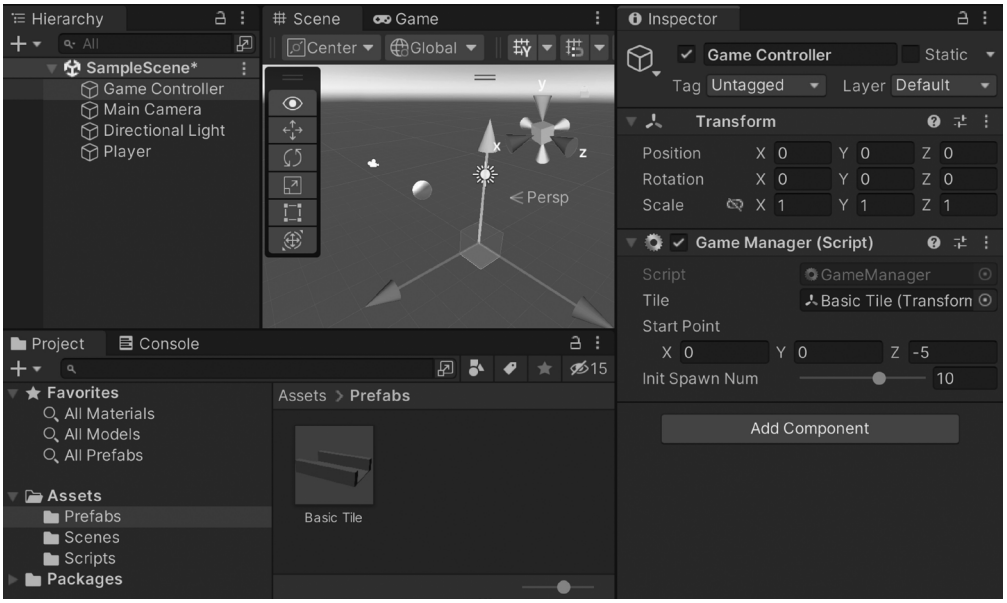
```
using UnityEngine;

/// <summary>
/// Manages the main gameplay of the game
/// </summary>
public class GameManager : MonoBehaviour
{
```

```
    [Tooltip("A reference to the tile we want to
        spawn")]
    public Transform tile;

    [Tooltip("Where the first tile should be placed
        at")]
    public Vector3 startPoint = new Vector3(0, 0, -5);

    [Tooltip("How many tiles should we create in
        advance")]
    [Range(1, 15)]
    public int initSpawnNum = 10;

    /// <summary>
    /// Where the next tile should be spawned at.
    /// </summary>
    private Vector3 nextTileLocation;

    /// <summary>
    /// How should the next tile be rotated?
    /// </summary>
    private Quaternion nextTileRotation;

    /// <summary>
    /// Start is called before the first frame update
    /// </summary>
    private void Start()
    {
        // Set our starting point
        nextTileLocation = startPoint;
        nextTileRotation = Quaternion.identity;

        for (int i = 0; i < initSpawnNum; ++i)
        {
            SpawnNextTile();
        }
    }


    /// <summary>
    /// Will spawn a tile at a certain location and
    /// setup the next position
    /// </summary>
```

```
    public void SpawnNextTile()
    {
        var newTile = Instantiate(tile,
            nextTileLocation, nextTileRotation);

        // Figure out where and at what rotation we
        /// should spawn the next item
        var nextTile = newTile.Find("Next Spawn
            Point");
        nextTileLocation = nextTile.position;
        nextTileRotation = nextTile.rotation;
    }
}
```

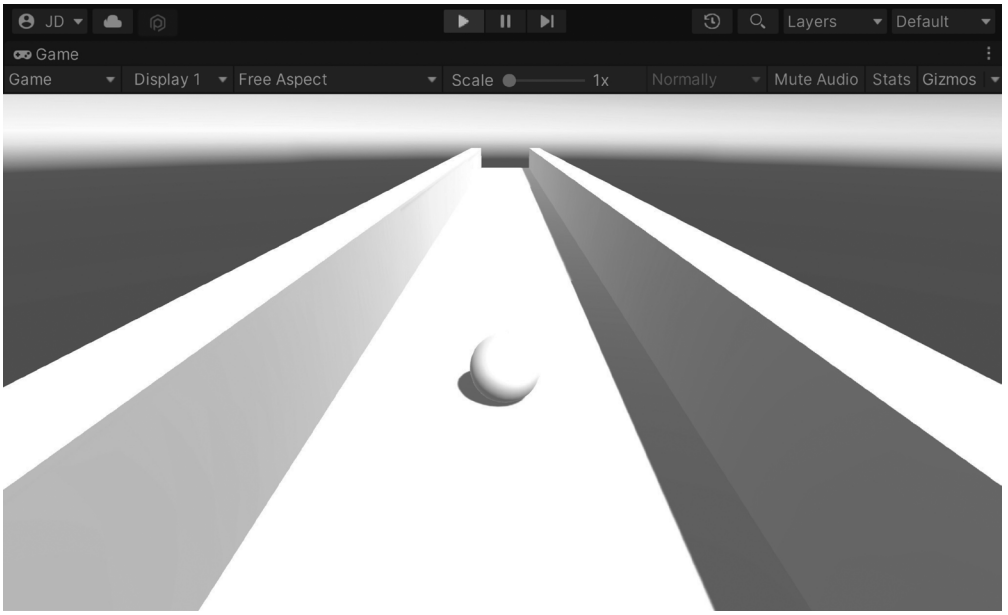
Ovaj skript će kreirati brojne pločice, jednu za drugom, na osnovu svojstava `tile` i `initSpawnNum`.

4. Sačuvajte skript i vratite se u Unity Editor. Odatle kreirajte novu instancu `empty GameObject` i dodelite joj naziv `Game Controller`, opciono resetujte pozicije, ako želite, u organizacione svrhe. Prevucite objekat i otpustite ga u **prozoru Hierarchy**. Radi jasnoće, resetujte poziciju, ako želite. Zatim, objektu priključite skript **Game Manager**, a zatim postavite svojstvo **Tile** prevlačenjem i otpuštanjem **Basic Tile** prefaba iz prozora **Project** u slot **Tile**:



Slika 1.22 – Dodeljivanje svojstva Tile

5. Sačuvajte scenu i pokrenite projekat:



Slika 1.23 – Trenutno stanje igre

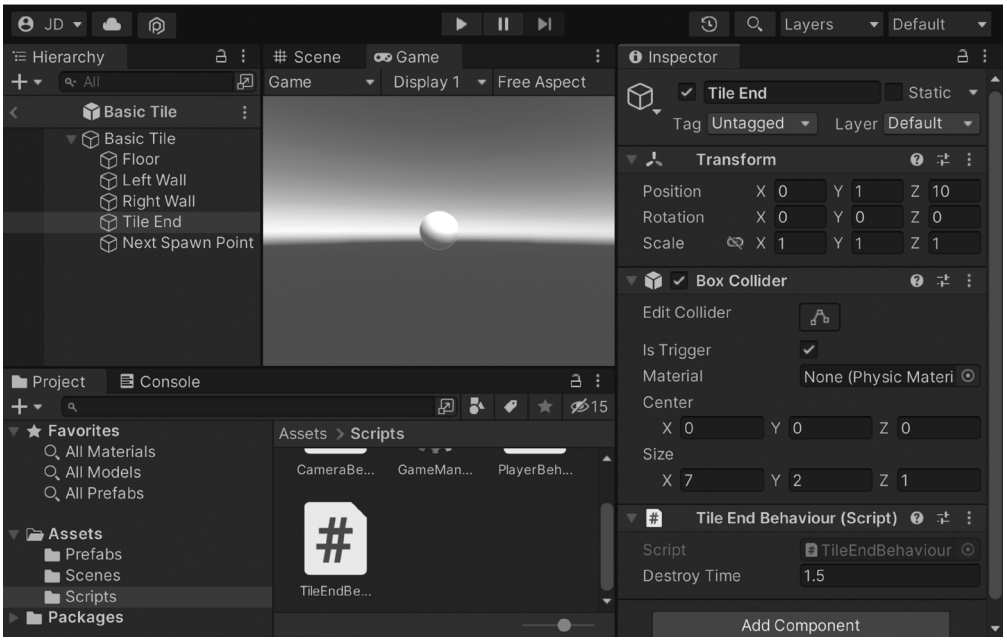
Odlično, ali sada je potrebno da kreiramo nove objekte posle ovih, a ne želimo da kreiramo mnogo njih. Bolje je da kada dođemo do kraja pločice kreiramo novu pločicu i uklonimo prethodnu. Kasnije ćemo više raditi na optimizaciji, ali na taj način uvek imamo približno isti broj pločica u igri u bilo kom trenutku.

6. U prozoru **Project**, u direktorijumu **Scripts**, kreirajte novi skript pod nazivom `TileEndBehaviour`, pomoću sledećeg koda:

```
using UnityEngine;

/// <summary>
/// Handles spawning a new tile and destroying this
/// one upon the player reaching the end
/// </summary>
public class TileEndBehaviour : MonoBehaviour
{
    [Tooltip("How much time to wait before destroying
        " + "the tile after reaching the end")]
    public float destroyTime = 1.5f;

    private void OnTriggerEnter(Collider other)
```

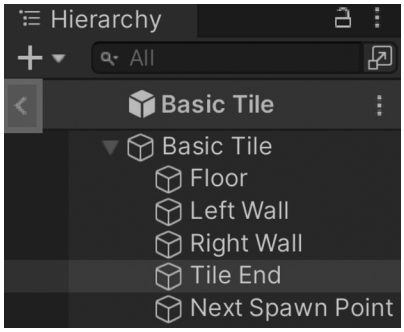
```
        {
            // First check if we collided with the player
            if(other.gameObject.GetComponent
            <PlayerBehaviour>())
            {
                // If we did, spawn a new tile
                var gm = GameObject.FindObjectOfType
                    <GameManager>();
                gm.SpawnNextTile();

                // And destroy this entire tile after a
                // short delay
                Destroy(transform.parent.gameObject,
                    destroyTime);
            }
        }
    }
```

7. Sada, da biste dodelili skript prefabu, možete da pređete u prozor **Project**, a zatim otvorite direktorijum **Prefabs**. Dvostruko kliknite na objekat **Basic Tile** da biste otvorili njegov uređivač. Na kartici **Hierarchy** izaberite objekat **Tile End**, a zatim u njega dodajte komponentu **Tile End Behaviour**:



Slika 1.24 – Dodavanje komponente Tile End Behaviour

8. Kliknite na strelicu ulevo pored naziva prefaba da biste se vratili na osnovnu scenu:

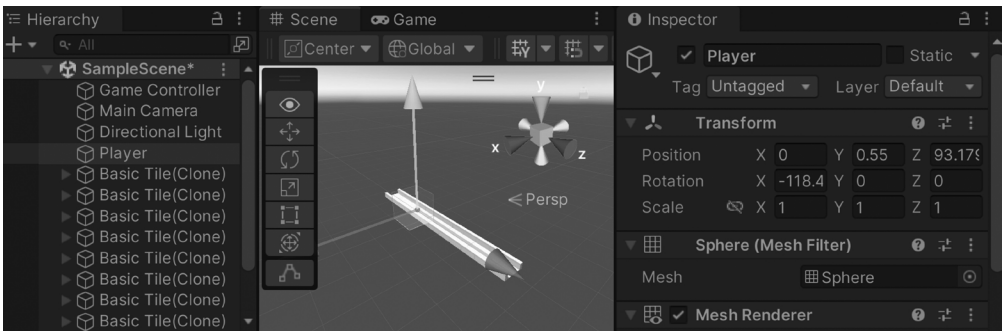


Slika 1.25 – Lokacija strelice ulevo

Savet

Takođe, možete da otvorite prefab uređivač tako što ćete izabrati prefab objekat iz prozora **Project** i na kartici **Inspector** kliknuti na dugme **Open Prefab**.

9. Sačuvajte scenu i pokrenite igru. Sada ćete primetiti da dok igrač nastavlja da se kreće, nove pločice se pojavljuju; ako pređete na karticu **Scene** dok igrate igricu, videćete da će se pločica uništiti kad lopta pređe preko nje:



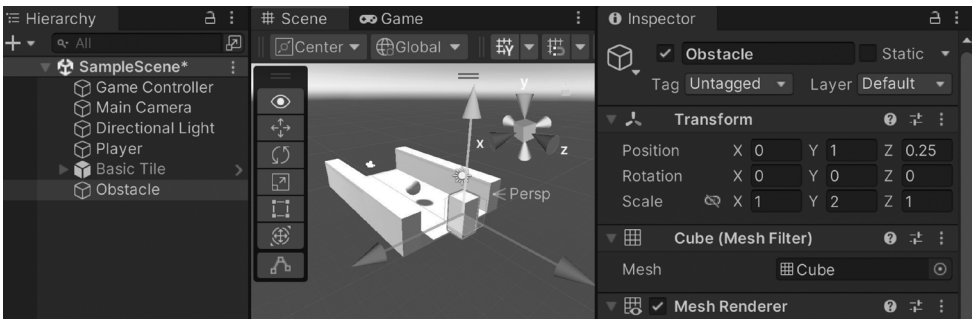
Slika 1.26 – Pločice su automatski uništene

To će osigurati da će ispred igrača biti pločice koje je potrebno da pređe! Ali naravno, to je samo beskrajna ravna linija. U sledećem odeljku ćemo videti kako da igru učinimo mnogo zanimljivijom.

Kreiranje prepreka

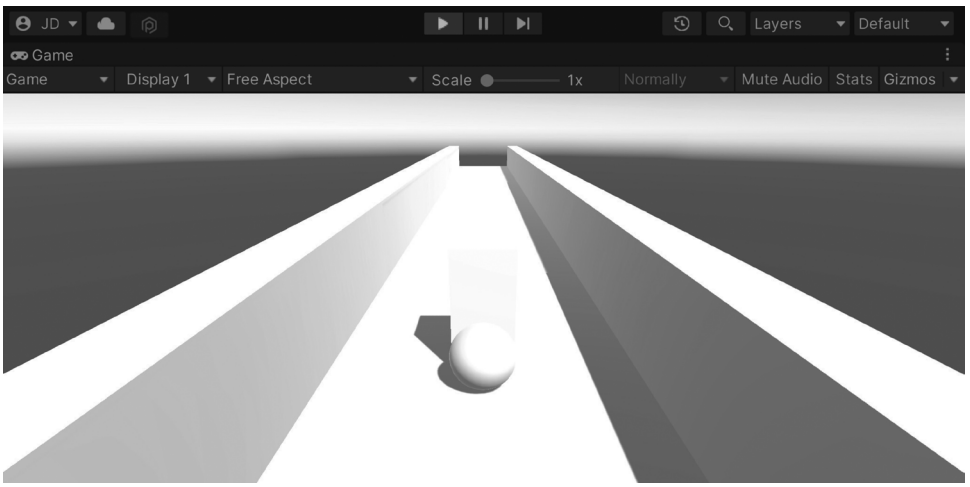
Odlično je što imamo neke osnovne pločice, ali je dobra ideja da damo igraču nešto da uradi, ili u našem slučaju, nešto što treba da izbegne. To će igraču pružiti neku vrstu izazova i osnovni cilj igre, a to je u našem slučaju izbegavanje prepreka. U ovom odeljku ćete naučiti da prilagodite svoje pločice da biste dodali prepreke koje igrač treba da izbegne. Dakle, pogledajmo korake:

1. Kao što ste kreirali prefab za osnovnu pločicu, kreirajte jednu prepreku pomoću koda. Želim da olakšam da vidim prepreku u svetu i da se uverim da nije prevelika, pa ću prevući i otpustiti **Basic Tile** prefab u svet.
2. Zatim kreirajte kocku, tako što ćete kliknuti na **GameObject | 3D Object | Cube**. Ovom objektu dodelite naziv `Obstacle`. Promenite vrednost **Scale** za **Y** na `2` i postavite ga iznad platforme, na poziciju (`0`, `1`, `0`, `25`):



Slika 1.27 – Dodavanje prepreka

3. Možete da pokrenete igru da biste videli kako će to funkcionisati:

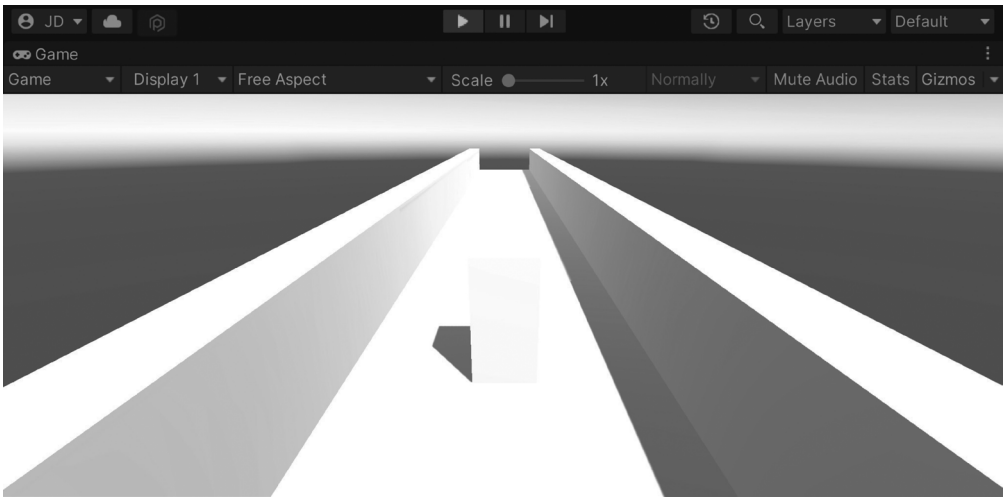


Slika 1.28 – Prepreke zaustavljaju igrača

4. Kao što vidite na prethodnom snimku ekrana, igrač se zaustavlja ali se ništa ne dešava. U ovom slučaju, želimo da igrač izgubi igru kada udari ovu prepreku, a da zatim igra ponovo krene; da biste to uradili, potrebno je da napišete skript. U prozoru **Project** kliknite na direktorijum **Scripts** i kreirajte novi skript, pod nazivom `ObstacleBehaviour`. Koristite sledeći kod:

```
using UnityEngine;
using UnityEngine.SceneManagement; // LoadScene

public class ObstacleBehaviour : MonoBehaviour
{
    [Tooltip("How long to wait before restarting the
        game")]
    public float waitTime = 2.0f;

    private void OnCollisionEnter(Collision collision)
    {
        // First check if we collided with the player
        if (collision.gameObject.GetComponent
        <PlayerBehaviour>())
        {
            // Destroy the player
            Destroy(collision.gameObject);

            // Call the function ResetGame after
            // waitTime has passed
            Invoke("ResetGame", waitTime);
        }
    }

    /// <summary>
    /// Will restart the currently loaded level
    /// </summary>
    private void ResetGame()
    {
        // Get the current level's name
        string sceneName =
            SceneManager.GetActiveScene().name;

        // Restarts the current level
        SceneManager.LoadScene(sceneName);
    }
}
```

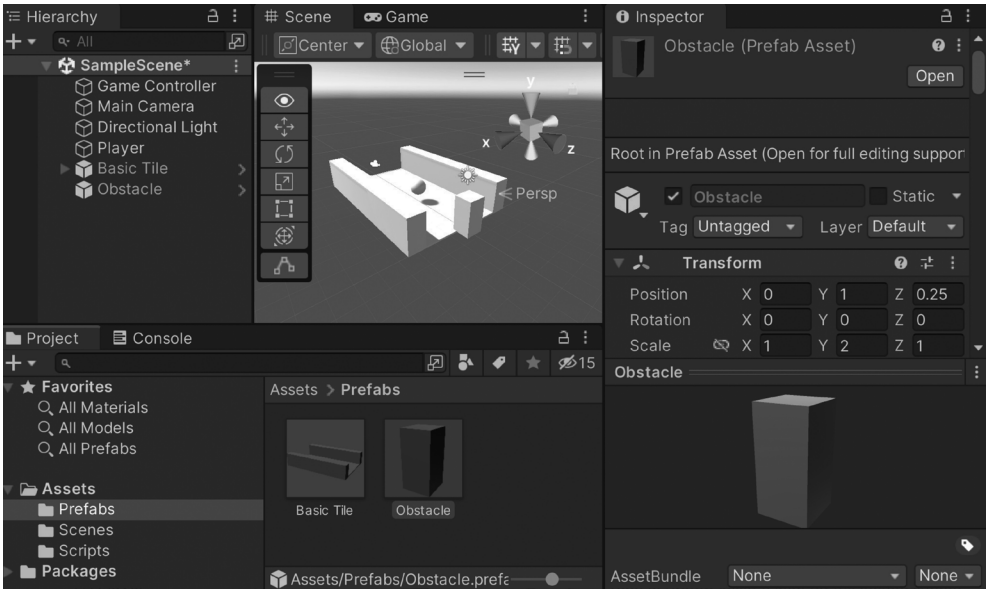
5. Sačuvajte skript i vratite se u uređivač, prilažući skript `Obstacle` GameObject objektu koji ste upravo kreirali.
6. Sačuvajte scenu i isprobajte igru:



Slika 1.29 – Prepreke uništavaju igrača

Kao što vidite na prethodnom snimku ekrana, kada udari u prepreku igrač je uništen, a zatim, posle nekoliko sekundi, igra ponovo počinje. Naučićete da koristite sisteme čestica i druge elemente da biste ovo poboljšali, ali u ovom trenutku funkcioniše, a to je ono što želimo.

7. Sada, kada znamo da radi ispravno, možemo da objekat učinimo prefabom. Kao što ste uradili za originalnu pločicu, prevucite i otpustite objekat **Obstacle** sa kartice **Hierarchy** na karticu **Project**, u direktorijum **Prefabs**:

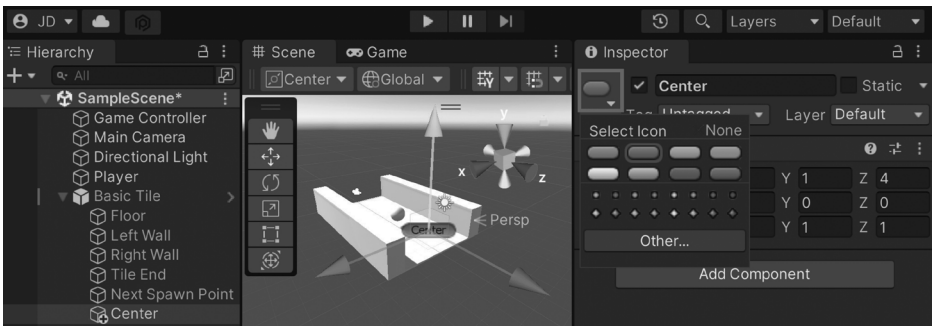


Slika 1.30 – Kreiranje prefaba Obstacle

8. Zatim, uklonite objekat **Obstacle**, jer će biti dodat prilikom kreiranja pločice. Da biste to uradili, izaberite objekat **Obstacle** u **prozoru Hierarchy**, a zatim pritisnite taster *Delete*.

9. Kreirajte markere da biste označili gde želite da postavite prepreke. Proširite objekat **Basic Tile** da biste prikazali njegove potomke, a zatim duplirajte objekat `Next Spawn Point` i za novi objekat promenite **Position** na (`0`, `1`, `4`). Zatim, promenite naziv objekta na `Center`.

10. Nakon toga, da biste pomogli da objekat bude prikazan unutar prozora **Scene**, otvorite prozor **Inspector** i kliknite na ikonicu *sive kocke*, a zatim u meniju **Select Icon**, izaberite boje koju želite (ja sam izabrao plavu). Kada to uradite, videćete tekst unutar uređivača ako ste blizu objekta (ali, podrazumevano, neće biti prikazan na kartici **Game**):

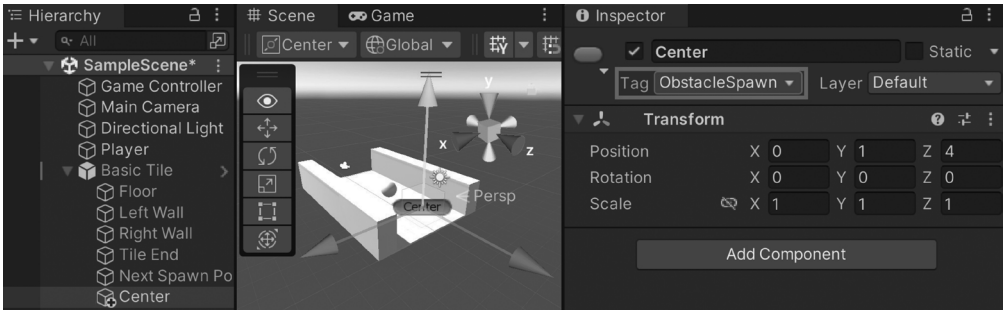


Slika 1.31 – Kreiranje Center markera

11. Želimo način da dobijemo sve potencijalne tačke postavljanja, koje ćemo želeti u slučaju da odlučimo da proširimo projekat u budućnosti, pa je potrebno da dodelite oznaku kao referencu da biste olakšali pronalaženje tih objekata. Da biste to uradili, na vrhu prozora **Inspector** kliknite na padajući meni **Tags** i izaberite opciju **Add Tag...**. Iz menija, koji će biti prikazan, pritisnite dugme **+**, a zatim joj dodelite naziv `ObstacleSpawn`:

Slika 1.32 – Kreiranje oznake ObstacleSpawn

12. Vratite se nazad i izaberite objekat **Center** i dodelite svojstvo **Tag** oznaci **ObstacleSpawn**:

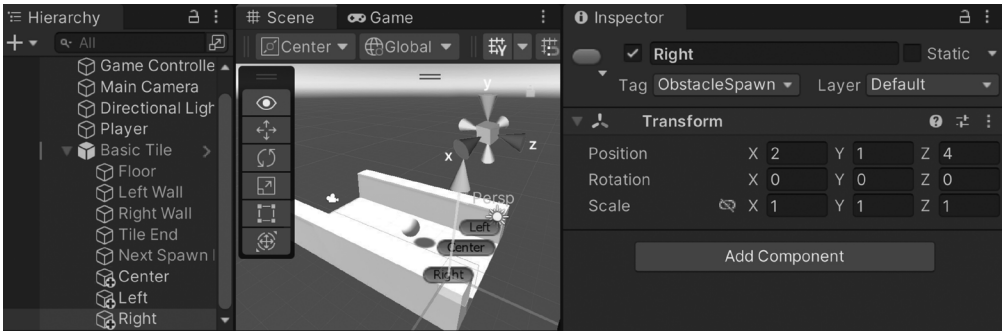


Slika 1.33 – Dodeljivanje oznake objektu Center

Napomena

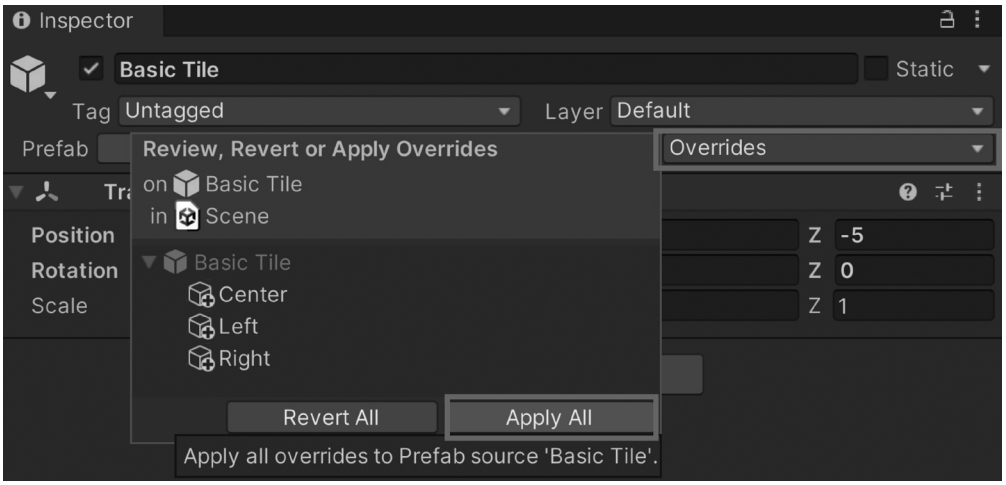
Za više informacija o oznakama i zašto bismo želeli da ih koristimo, posetite stranicu `https://docs.unity3d.com/Manual/Tags.html`.

13. Duplirajte marker dva puta i dodelite im nazive `Left` i `Right`, pomerajući ih dve jedinice levo i desno od centra, tako da postanu druge moguće tačke prepreka:



Slika 1.34 – Kreiranje Left i Right markera

14. Imajte na umu da, podrazumevano, ove promene ne utiču na originalni prefab; zato su objekti trenutno crni tekst. Da biste to promenili, izaberite **Basic Tile**, a zatim, u prozoru **Inspector**, u odeljku **Prefab**, kliknite na **Overrides** i izaberite opciju **Apply All**:



Slika 1.35 – Primena promena na prefab

15. Sada kada je prefab ispravno podešen, možete da nastavite i da ga uklonite tako što ćete ga selektovati u **prozoru Hierarchy** i pritisnuti *Delete*.

16. Zatim, potrebno je da izvršite neke modifikacije u skriptu `GameManager`. Za početak, potrebno je da dodate nove promenljive:

```
/// <summary>
/// Manages the main gameplay of the game
/// </summary>
public class GameManager : MonoBehaviour
{
```

```
    [Tooltip("A reference to the tile we want to
        spawn")]
    public Transform tile;

    [Tooltip("A reference to the obstacle we want to
        spawn")]
    public Transform obstacle;

    [Tooltip("Where the first tile should be placed
        at")]
    public Vector3 startPoint = new Vector3(0, 0, -5);

    [Tooltip("How many tiles should we create in
        advance")]
    [Range(1, 15)]
    public int initSpawnNum = 10;

    [Tooltip("How many tiles to spawn with no
        obstacles")]
    public int initNoObstacles = 4;
```

Prva od ovih promenljivih je referenca za prepreku za koju ćemo kreirati kopije. Druga je parametar za broj pločica koje bi trebalo da budu dodate pre dodavanja prepreka. To je potrebno da bismo osigurali da igrač može da vidi prepreke pre nego što treba da ih izbegne.

17. Zatim, potrebno je da modifikujete funkciju `SpawnNextTile` tako da doda i prepreke:

```
/// <summary>
/// Will spawn a tile at a certain location and setup
/// the next position
/// </summary>
/// <param name="spawnObstacles">If we should spawn an
/// obstacle</param>
public void SpawnNextTile(bool spawnObstacles = true)
{
    var newTile = Instantiate(tile, nextTileLocation,
                              nextTileRotation);

    // Figure out where and at what rotation we should
    // spawn the next item
    var nextTile = newTile.Find("Next Spawn Point");
    nextTileLocation = nextTile.position;
    nextTileRotation = nextTile.rotation;
```

```
        if (spawnObstacles)
        {
            SpawnObstacle(newTile);
        }

    }
```

Vidite da smo modifikovali funkciju `SpawnNextTile` tako da sada ima podrazumevani parametar postavljen na `true`, koji će nam reći da li želimo da dodamo prepreke ili ne. Na početku igre, ne želimo da igrač mora odmah da izbegava prepreke, ali možemo da podesimo vrednost da bismo povećali ili smanjili broj koji koristimo. Pošto ima podrazumevanu vrednost `true`, originalna verzija poziva u funkciji `Start` će i dalje raditi bez greške, ali ćemo je kasnije modifikovati.

18. Ovde, postavljamo pitanje da li je vrednost `true` za pozivanje funkcije `SpawnObstacle`, ali ona još nije napisana. Sada ćete je dodati, ali prvo upotrebite klasu `List` da biste bili sigurni da kompajler zna na koju klasu `List` se pozivate, tako da je potrebno da dodate iskaz `using` na vrh fajla:

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

19. Sada možete da napišete funkciju `SpawnObstacle`. Dodajte sledeću funkciju u skript:

```
private void SpawnObstacle(Transform newTile)
{
    // Now we need to get all of the possible places
    // to spawn the obstacle
    var obstacleSpawnPoints = new List<GameObject>();

    // Go through each of the child game objects in
    // our tile
    foreach (Transform child in newTile)
    {
        // If it has the ObstacleSpawn tag
        if (child.CompareTag("ObstacleSpawn"))
        {
            // We add it as a possibility
            obstacleSpawnPoints.Add(child.gameObject);

        }

    }
```

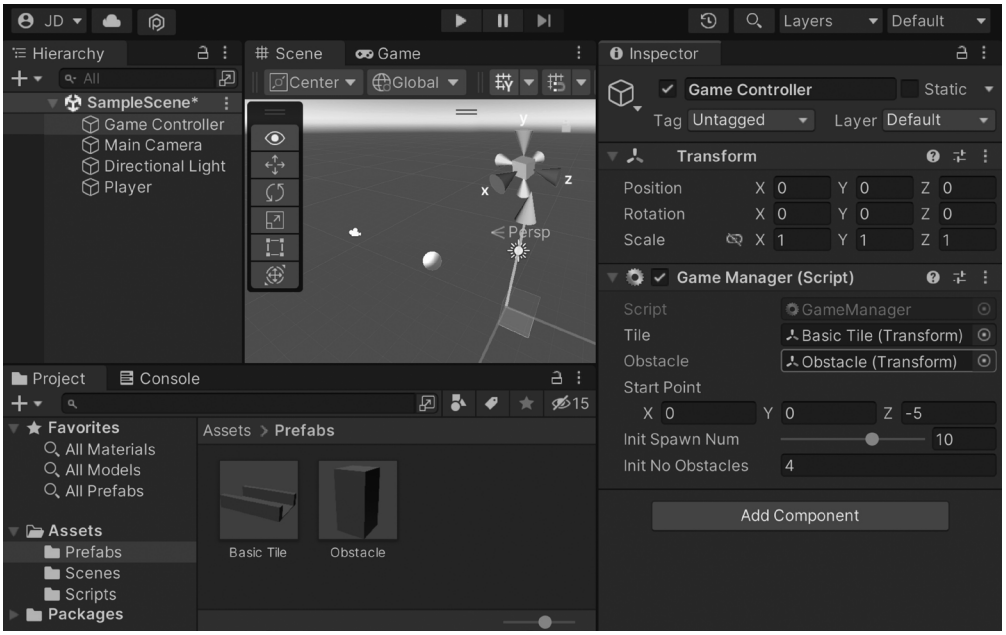
```
        // Make sure there is at least one
        if (obstacleSpawnPoints.Count > 0)
        {
            // Get a random spawn point from the ones we
            // have
            int index = Random.Range(0,
                obstacleSpawnPoints.Count);
            var spawnPoint = obstacleSpawnPoints[index];

            // Store its position for us to use
            var spawnPos = spawnPoint.transform.position;

            // Create our obstacle
            var newObstacle = Instantiate(obstacle,
                spawnPos, Quaternion.identity);

            // Have it parented to the tile
            newObstacle.SetParent(spawnPoint.transform);
        }
    }
```

20. Na kraju, ažurirajte funkciju `Start`:

```
    /// <summary>
    /// Start is called before the first frame update
    /// </summary>
    private void Start()
    {
        // Set our starting point
        nextTileLocation = startPoint;
        nextTileRotation = Quaternion.identity;

        for (int i = 0; i < initSpawnNum; ++i)
        {
            SpawnNextTile(i >= initNoObstacles);
        }
    }
```

Sada, dokle god je `i` manja od vrednosti `initNoObstacles`, neće biti dodata promenljiva, što nam efektivno daje bafer od četiri pločice, koje mogu da budu prilagođene promenom promenljive `initNoObstacles`.

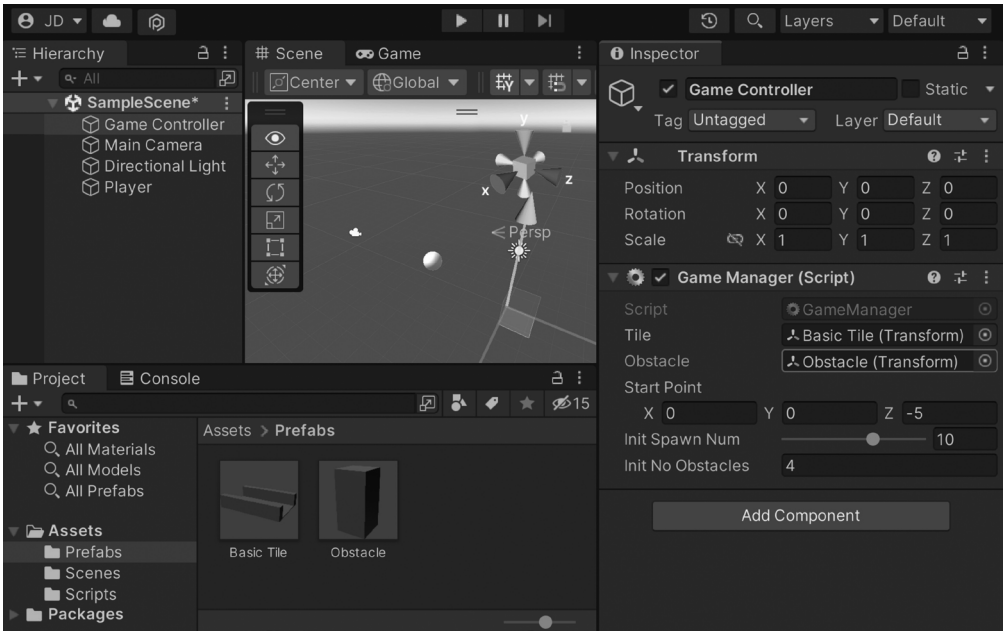
21. Sačuvajte skript i vratite se u Unity Editor. Zatim, u prozoru **Inspector** dodelite promenljivu `Obstacle` komponente **Game Manager (Script)** prefabu **Obstacle**, koji smo prethodno kreirali:



Slika 1.36 – Dodeljivanje svojstva Obstacle

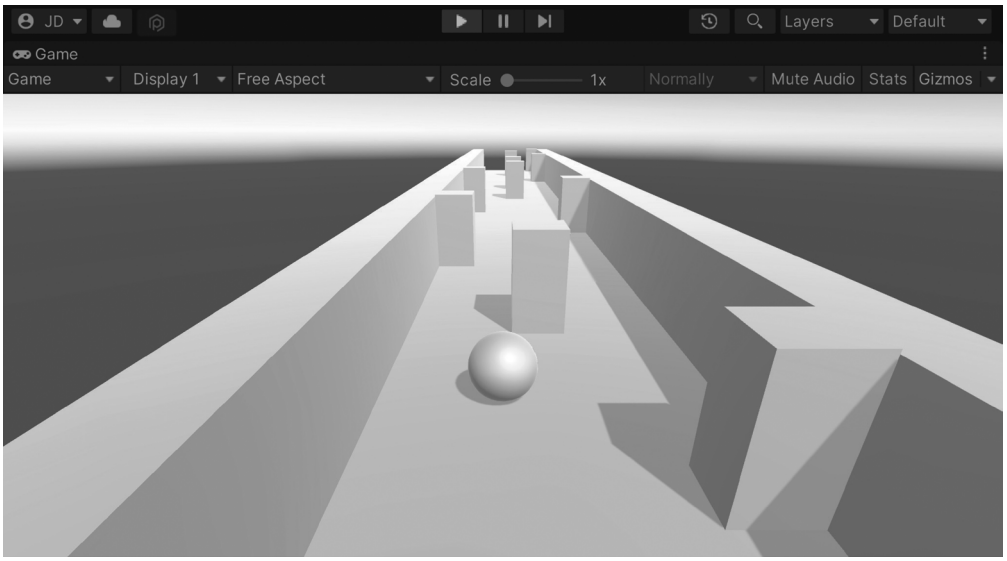
22. Trenutno je malo teško videti elemente zbog podrazumevanih podešavanja svetla, pa otvorite **prozor Hierarchy** i izaberite objekat **Directional Light**. Usmereno svetlo deluje slično kao što Sunce deluje na Zemlju, svetli sa određene pozicije.

23. Po podrazumevanim podešavanjima, svetlo je previše svetlo, a senke su previše tamne, tako da sam u prozoru **Inspector** promenio **Intensity** na `0.5`, a zatim i svojstvo **Realtime Shadows** | **Strength** na `0.5`:



Slika 1.37 – Podešavanje usmerenog svetla

24. Sačuvajte scenu i igrajte igru:



Slika 1.38 – Trenutno stanje igre

Kao što vidite na prethodnom snimku ekrana, sada imamo brojne prepreke koje naš igrač treba da izbegne!

Napomena

Za više informacija o usmerenim svetlima i drugim tipovima osvetljenja koje Unity ima, posetite stranicu `https://unity3d.com/learn/tutorials/topics/graphics/lighttypes?playlist=17102`.

Rezime

To je to! Čvrsta osnova – ali samo to, osnova. Međutim, opisali smo dosta sadržaja u ovom poglavlju. Govorili smo o tome kako da kreiramo novi projekat u Unity-u i izgradili smo igrača koji će se kretati kontinuirano i primati unose za horizontalno kretanje. Zatim smo govorili o tome kako možemo da koristimo Unity atribute i XML komentare da bismo poboljšali kvalitet koda i olakšali rad u timovima. Takođe, opisali smo kako da imamo pokretnu kameru. Kreirali smo sistem dizajna nivoa koji je zasnovan na pločicama, gde smo kreirali nove pločice dok se igra prikazuje, nasumično stvarajući prepreke koje igrač treba da izbegne.

U ovoj knjizi ćemo istražiti šta možemo da uradimo da poboljšamo ovaj projekat i da ga doteramo dok ga menjamo, kako bismo obezbedili najbolje moguće iskustvo na mobilnim platformama. Međutim, pre nego što dođemo do toga, potrebno je da smislimo kako da rasporedimo naše projekte, o čemu ćemo govoriti u sledećem poglavlju.