Univerzitet u Kragujevcu

Prirodno-matematički fakultet
Institut za matematiku i informatiku

Naziv radionice: Gaming With Unity
Tema: Veverica - Unity 2D Game

Studenti:
Stefan Agatonović 41/2016
Stefan Damajnović 61/2016
Miloš Tripković 106/2016
Ivona Šović 111/2016

Mentor:
dr Ana Kaplarević-Mališić

Sadržaj

Šta je Unity?

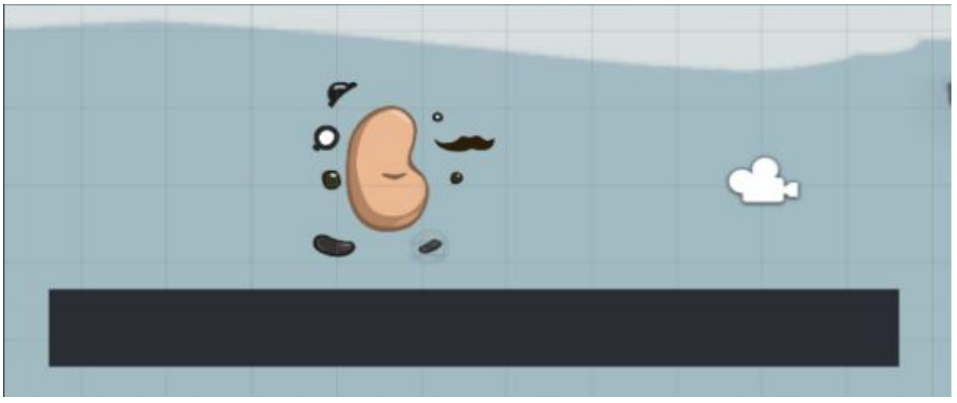
Unity je višeplatformski game engine. Predstavlja platformu i integrisano razvojno okruženje za ravoj 2D i 3D interaktivnih multimedijalnih aplikacija i igara za računare, konzole, mobilne uređaje i web sajtove, sa mogućnošću izvšavanja na preko 20 podržanih platformi. On brine o osnovnim zadacima kao što su iscrtavanje, detekcija i odgovor na koliziju.
Funkcionalnosti koje pruža game engine su:

- Zvuk
- Skripte - podrška za pisanje novih programskih elemenata u određenom programskom jeziku
- Animacije - 2D i 3D
- Iscrtavanje - 2D i 3D grafika
- Fizička enigma - detekcija kolizije, odgovor na koliziju
- Mrežni rad
- Upravljanje memorijom

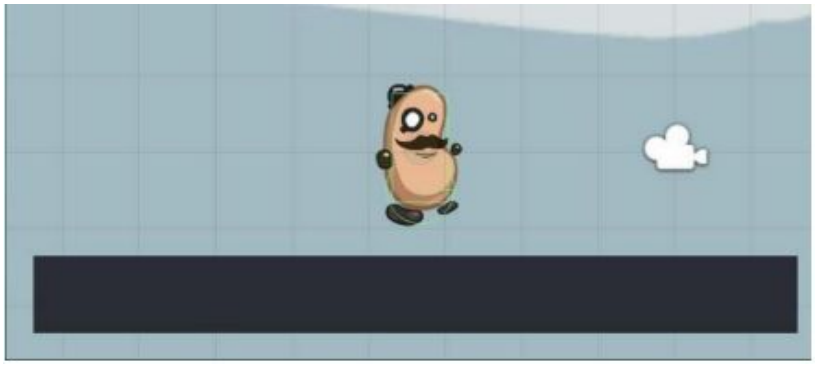
Jezgro Unity-a napisano je u C/C++ programskom jeziku, dok je Unity UI Editor napisan u jeziku C#. Za pristup najnižem sloju, odnosno jezgru i funkcijama Unity-a, dostupan je API za korišćenje u .NET Framework-u, i jezicima C#, Boo, ali i JavaScript. Za pisanje koda se standardno koristi Monodevelop, ali je moguće korišćenje bilo kog drugog okruženja, npr. Microsoft Visual Studio.

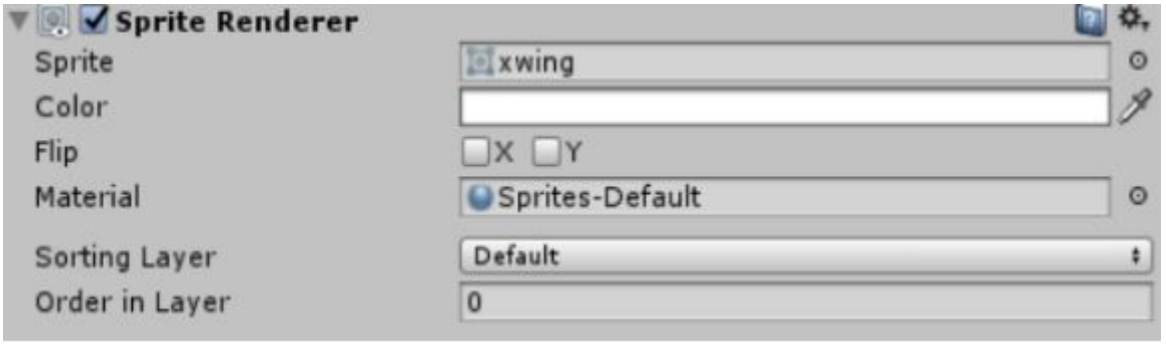
Sprites

U Unity-u 2D sprite-ovi su jednostavne slike koje uglavnom prikazuju jedan objekat. Nekoliko sprite-ova može činiti jedan objekat u pojedinačnim frejmovima kako bi bila moguća animacija lika. Na slici možemo videti objekat sačinjen od više sprite-ova.



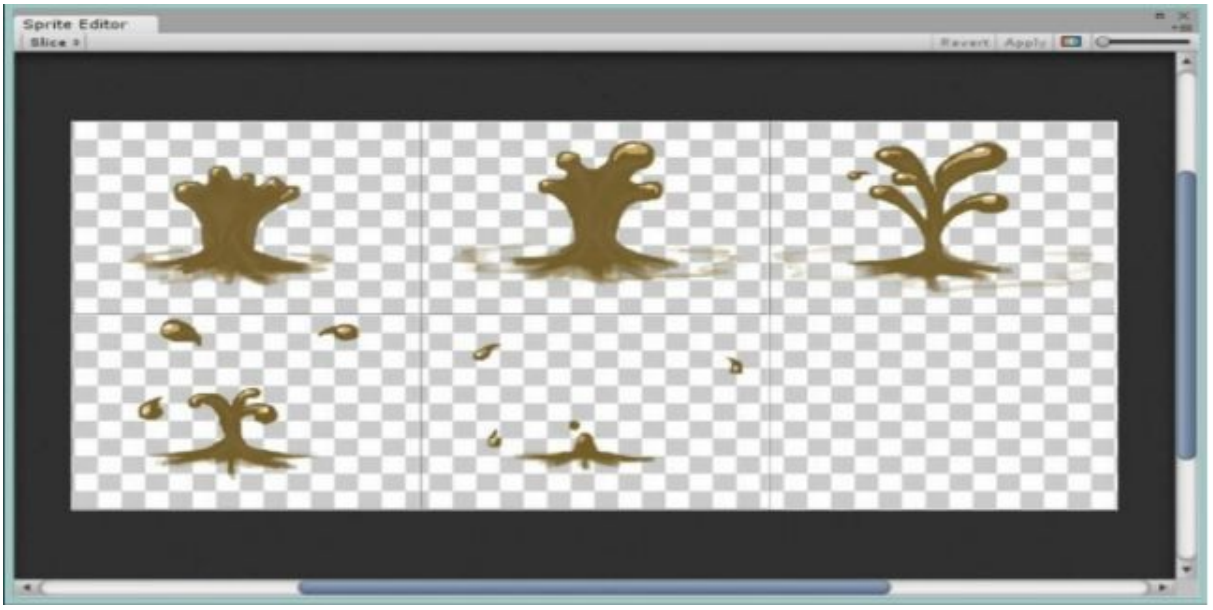
U jezgru 2D sistema se nalazi novi uvoznik sprite tekstura, koji uvozi vaše teksture i priprema ih kao sprite-ove u vašem folderu gde se nalazi projekat. Kada se prenesu na scenu automatski dobijaju Sprite Render objekat koji je spreman da se prikaže u igri.Sprite Render je novi 2D pokazivač koji iscrtava sprite-ove na ekran.





U okviru Sprite Render komponente možemo videti ime sprite-a, možemo menjati boju, okretati sprite po x i y osi, menjati material sprite-a, birati layer u kome se nalazi i određivati njegov raspored u layer-u.

Spritesheets je ključni deo bilo kog 2D sistema, posebno 2D animacije. Ujedinjavanje svih tekstura u jednu veću teksturu daje bolje performanse prilikom slanja sprite-ova na grafičku karticu, nego slanje puno manjih fajlova. Ovo se odnosi na sledeću sliku:



Asset Store

U okviru Scene and Game view panela mozemo naći i Asset Store gde možemo kupiti ili besplatno skinuti gotove modele, skripte, materijale, audio fajlove i još mnogo toga.

Okruženje Unity programa

Prilikom startovanja Unity programa otvara se prozor za kreiranje novog projekta, tu je potrebno uneti naziv projekta, lokaciju gde će projekat biti sačuvan i u ovom slučaju biramo

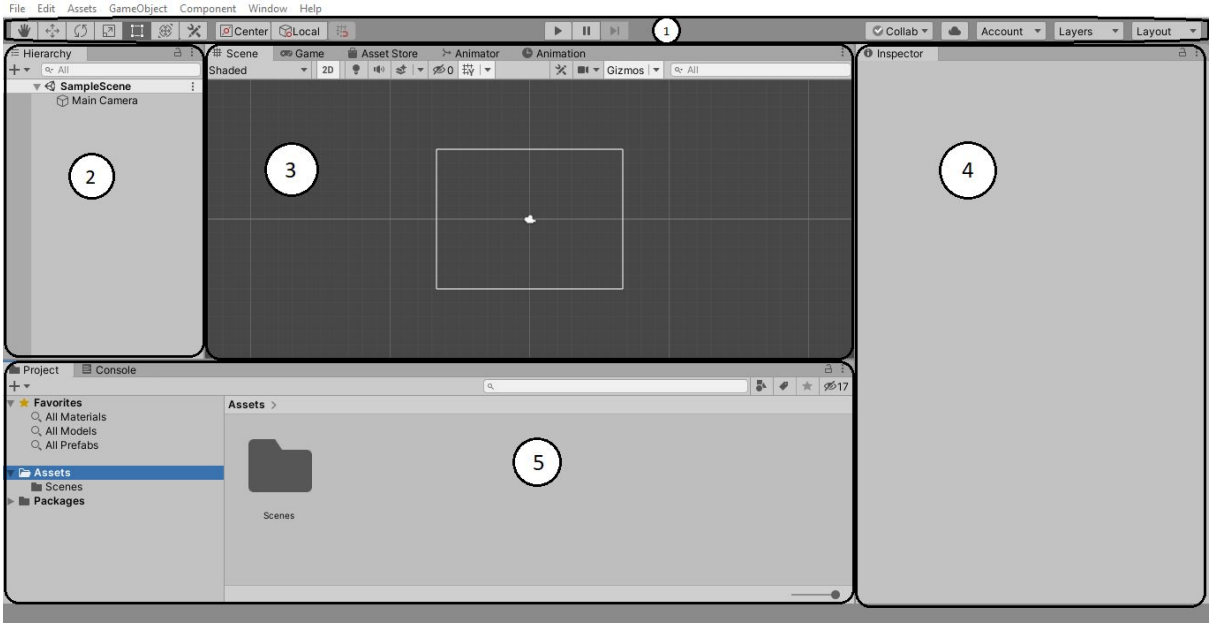
opciju 2D. Klikom na dugme *Create Project* otvara se okruženje u kome započinjemo sa kreiranjem igrice.

Unity u direktorijumu programa, prilikom kreiranja projekta, kreira različite fajlove i direktorijume.
Unity kreira četiri različita direktorijuma:

1. *Assets* - Primarni direktorijum za objekte igre, kao što su modeli, teksture, zvukovi i skripte. Preporuka je da se u ovom folderu drže fajlovi organizovani u direktorijume kako se ne bi mešali.
2. *Library* - Lokalni keš za importovana sredstva.
3. *ProjectSettings* - U ovom folderu se skladište sva podešavanja projekta kao što su fizika, oznake itd.
4. *Temp* - Folder za smeštanje privremenih fajlova generisanih tokom bildovanja projekta.

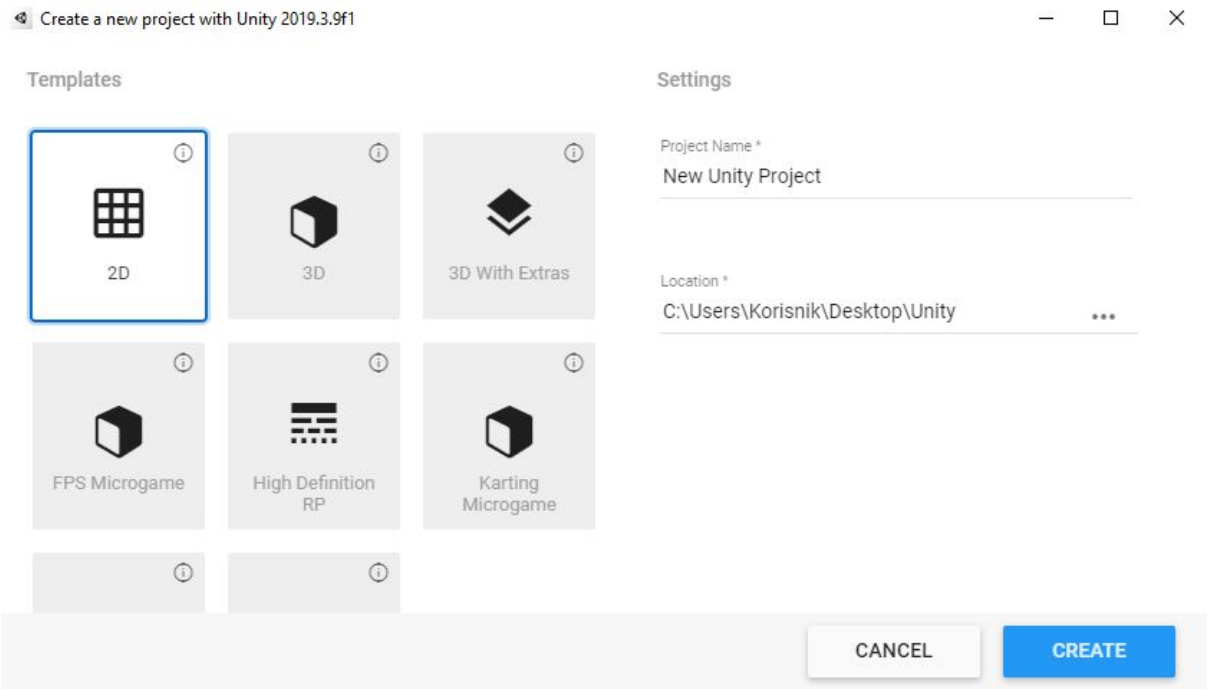
Razvojno okruzenje

Kada napravimo projekat, otvara se glavni editorski panel, koji je podeljen u 5 glavnih delova.



Pre nego što počnemo sa radom potrebno je napraviti projekat. Pokrenućemo *Unity*. Pošto je u pitanju **2D** projekat potrebno je izabrati *2D* u *Templates* delu, pa dodeliti naziv projektu i lokaciju gde će projekat biti sačuvan i klikom na **Create** možemo početi.

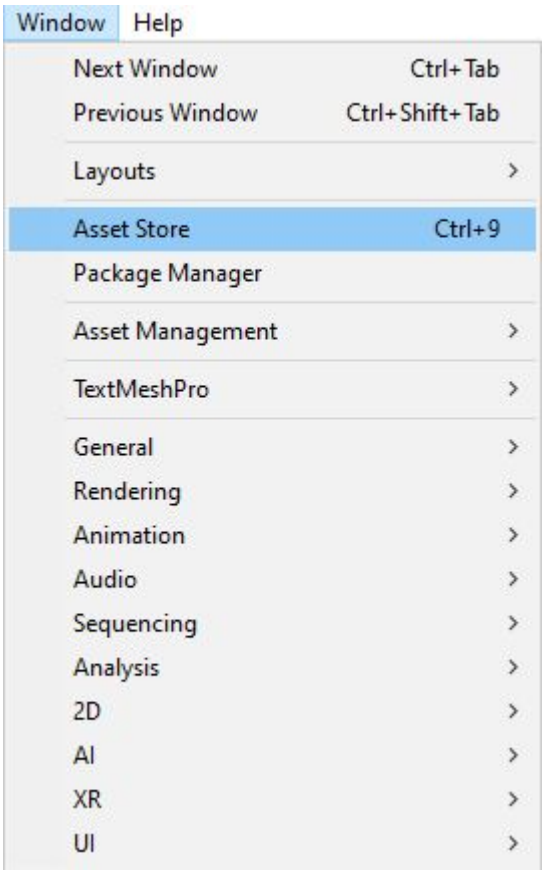
Izrada igrice



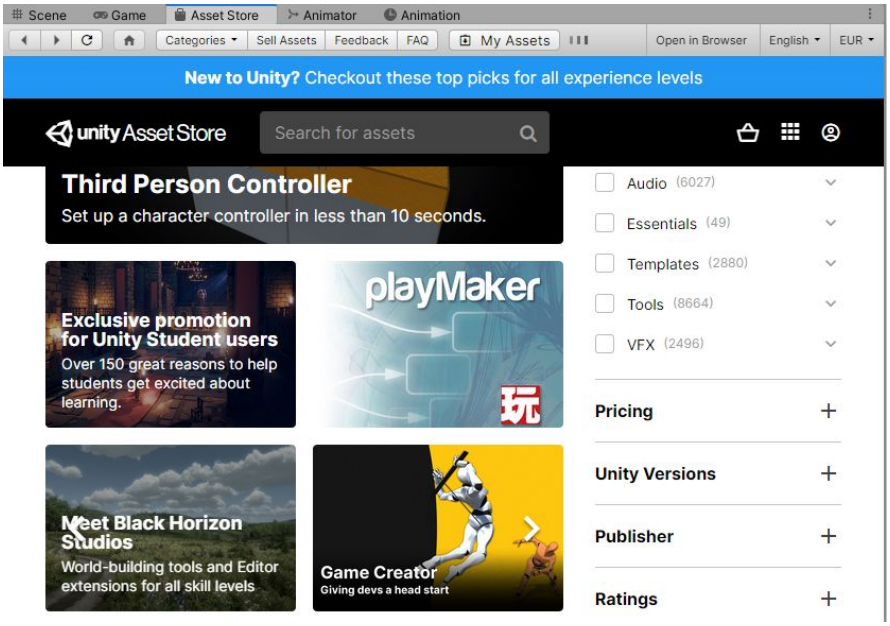
Ovo kreira novi projekat i pošto smo izabrali 2D template Unity je postavio posebna podešavanja na našu kameru i editor da bolje rade sa 2D.

Asset store

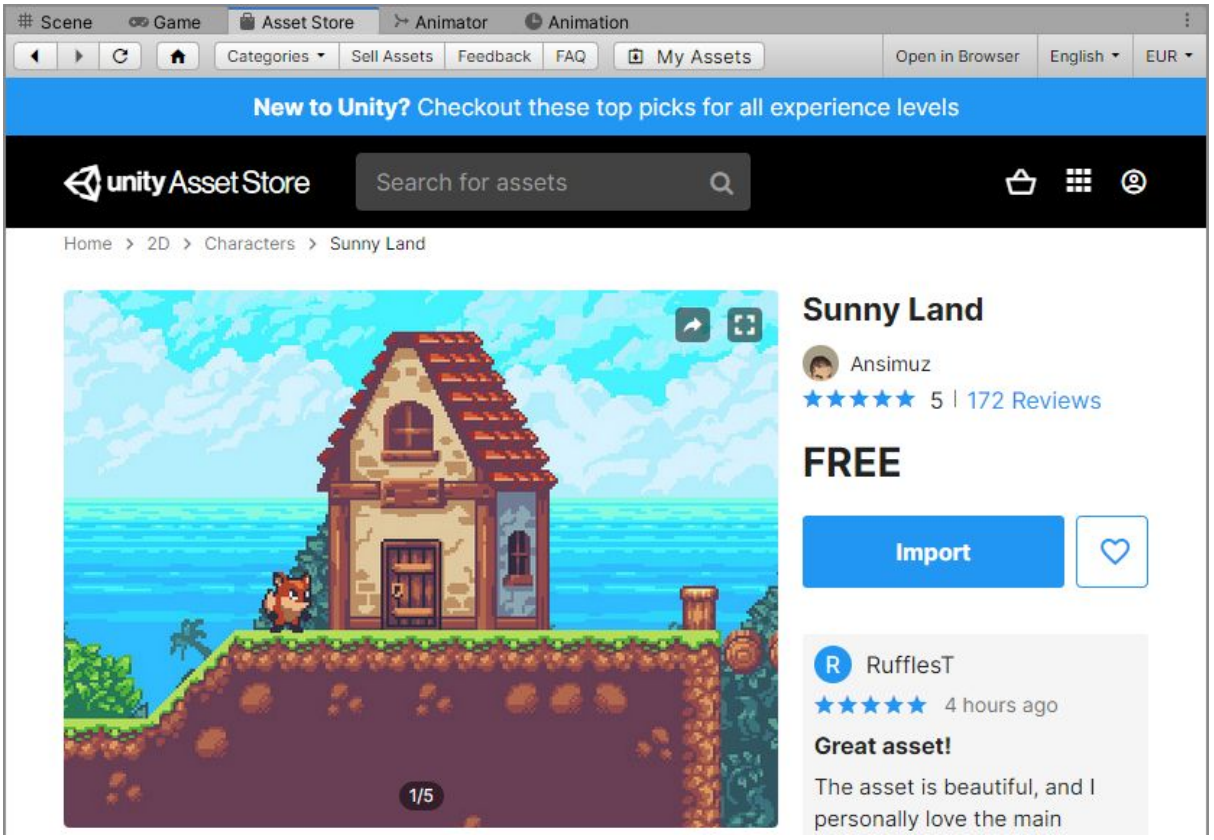
Na početku svakog projekta preporučljivo je koristiti *Unity asset store* da bi našli sve što nam je potrebno za početak sa radom (Sprite-ovi, zvučni efekti itd.) ,što je naravno moguće kasnije i izmeniti našim Sprite-ovima i animacijama.
Za ovaj projekat koristićemo *Sunny Land* asset pack koji je besplatan. Da bi u naš projekat ubacili ovaj asset prvo je potrebno otvoriti asset store klikom na Window opciju a zatim klikom na Asset store (Ctrl+9).



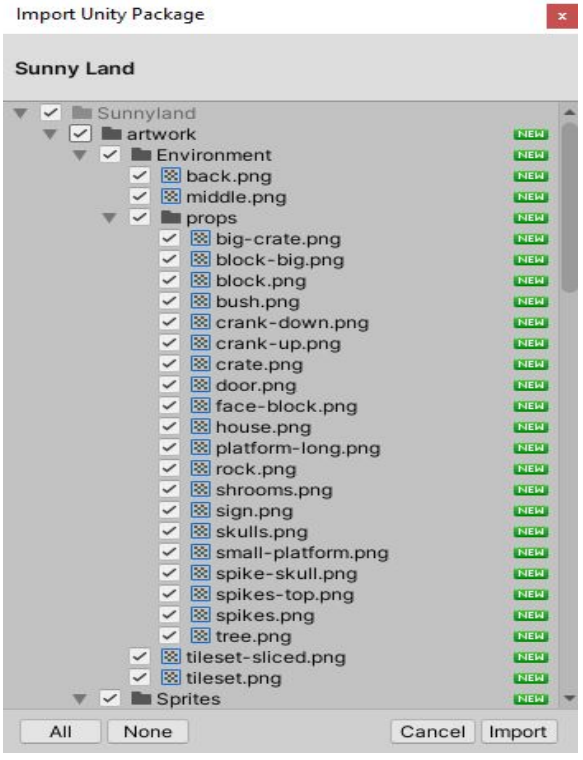
Ovo u delu editora gde je postavljena scena prikazuje asset store.



Potrebno je navigirati do paketa koji nama treba tako da pretražujmo Sunny Land.

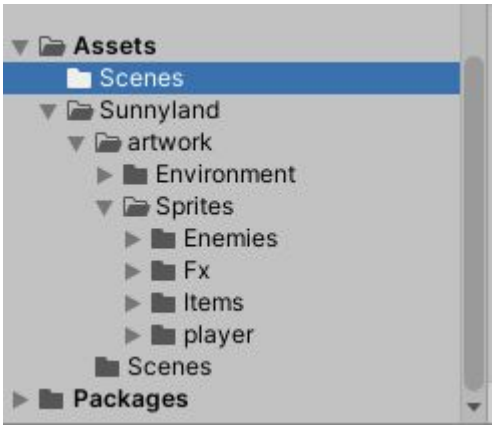


Kada ga pronađemo potrebno ga je instalirati i import-ovati u projekat.
Klikom na dugme import otvoriće se prozor koji nam prikazuje sve što taj asset paket sadrži i ako nam se nešto određeno ne svidi možemo ga ukloniti iz liste i neće se ubaciti u naš projekat.



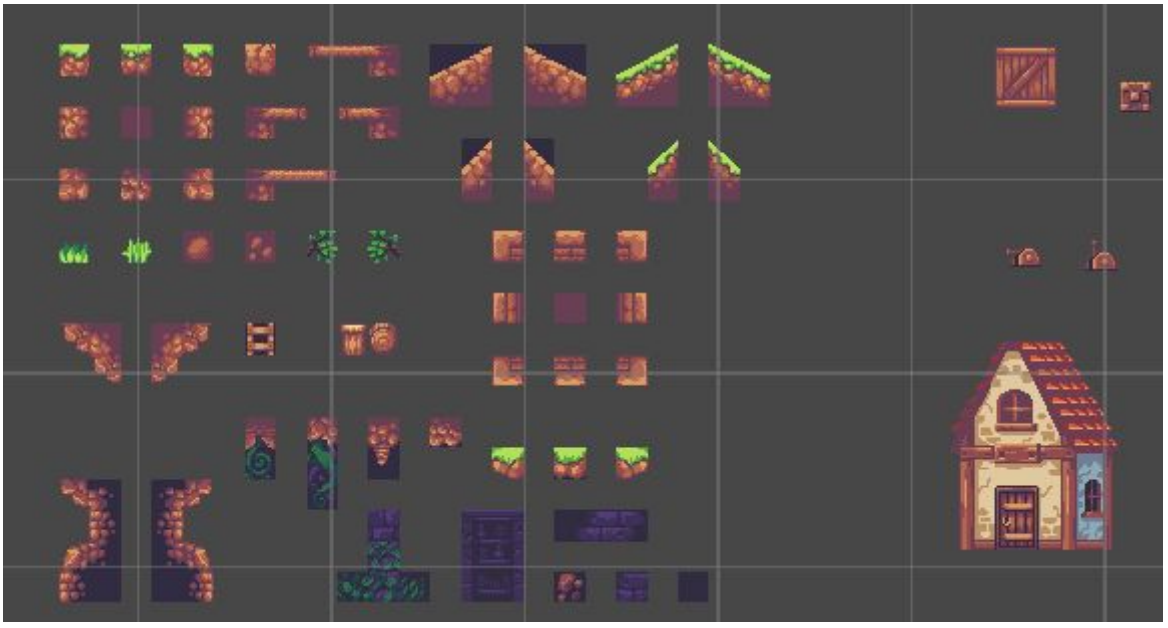
Za potrebe ovog projekta biće nam potreban ceo sadržaj ovog paketa tako da je potrebno pritisnuti samo Import dugme.

U Project delu editora pojaviće se paket koji smo instalirali sa sadržajem sortiranim po poddirektorijumima zbog lakše navigacije.



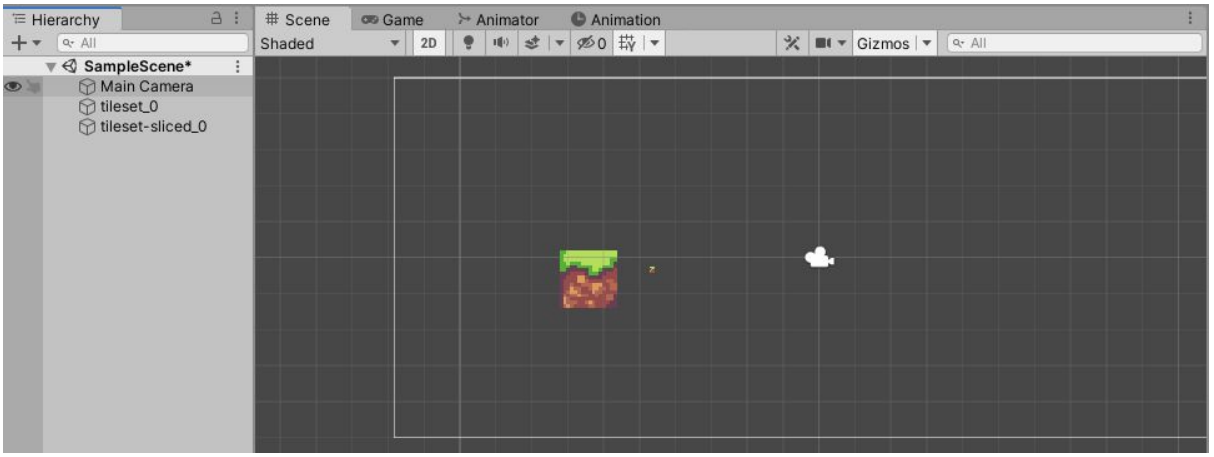
Tileset

Svaki asset paket uglavnom u sebi sadrži neku demo scenu koja prikazuje način na koji se instalirani paket moze iskoristiti. U poddirektorijumu Scenes nalazi se demo scena i klikom na nju otvara se na sredini editora.

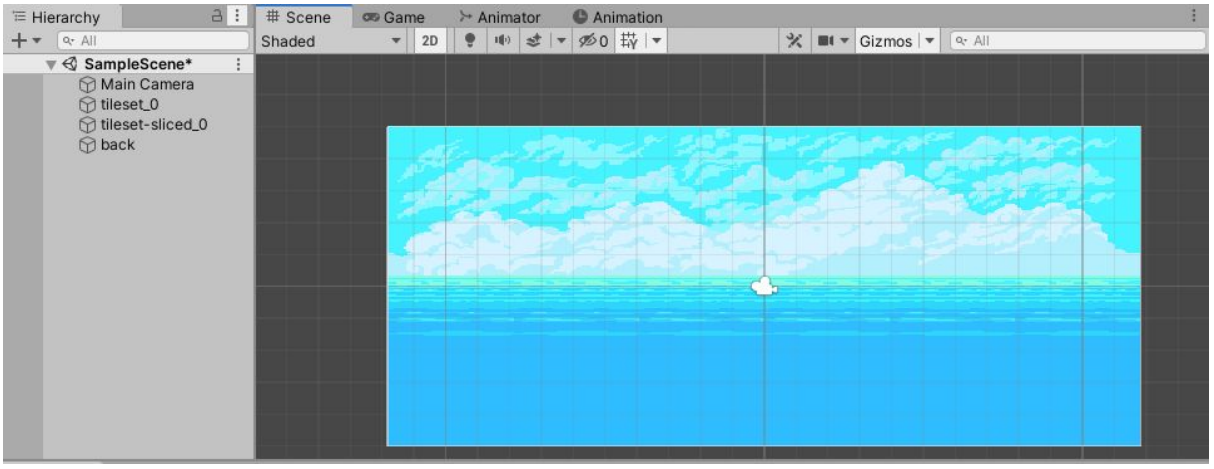


Ova demo scena sadrži samo sve tile-ove koje se nalaze u paketu uklopljene u celine da bi onaj ko radi sa njima znao kako se i gde uklapaju. Za dalji rad koristićemo Sample scenu koja je kreirana kada smo napravili projekat.
Sada možemo koristiti sadržaj paketa koji smo instalirali u sceni a način na koji to radimo zavisi od paketa koji smo instalirali.Ovaj paket sve sprite-ove dali u individualne slike. Ako uđemo u poddirektorijum artwork unutar njega se nalaze direktorijumo Environment koji sadrži sve slike koji će predstavljati pozadinu i props poddirektorijum koji u sebi sadrži sve sitne detalje koji se mogu dodati sceni kao i neke dodatke sa kojima igrač može da interaguje. Artwork direktorijum još u sebi sadrži i Sprites direktorijum u kome se nalaze slike

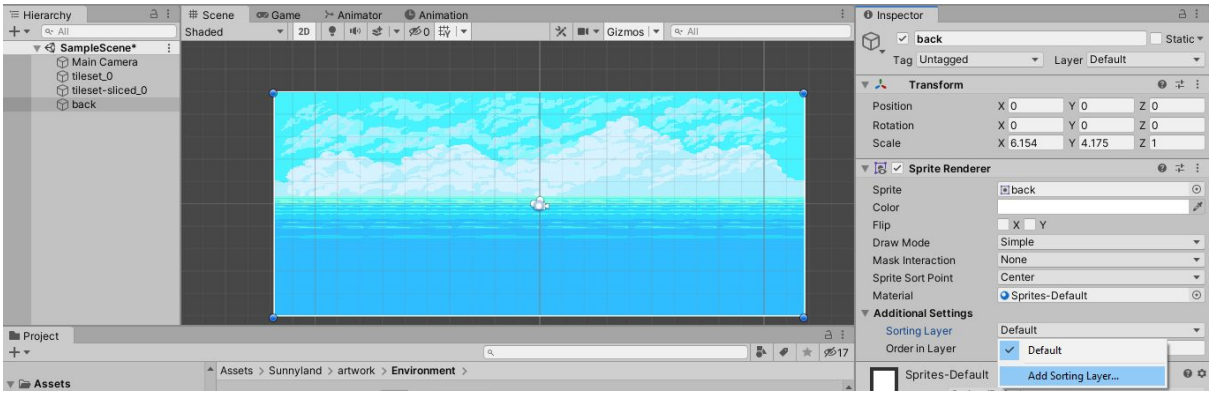
kojima će biti predstavljen glavni lik, neprijatelji, predmeti koje glavni lik može sakupljati kao i efekti koji se pojavljuju kada igrač interaguje samo neprijateljima ili skupi neki predmet.
Za kreiranje scene koristićemo sadržaj Environment poddirektorijuma. Možemo uzeti bilo koju sliku i prevući je na scenu i to će odmah raditi, slika će se pojaviti na sceni i tako možemo kreirati nivo kakav god mi želimo. Umesto ovakvog načina dizajniranja većina 2D paketa koristi *tileset,* koji predstavlja veoma veliku sliku sa gomilom sprajtova poređanih jedan do drugog i ako prevučemo bilo koji isečeni deo tog tileset-a slika će opet biti prikazana ali kao dosta manja.Ubacivanjem dva sprite-a scena izgleda ovako:



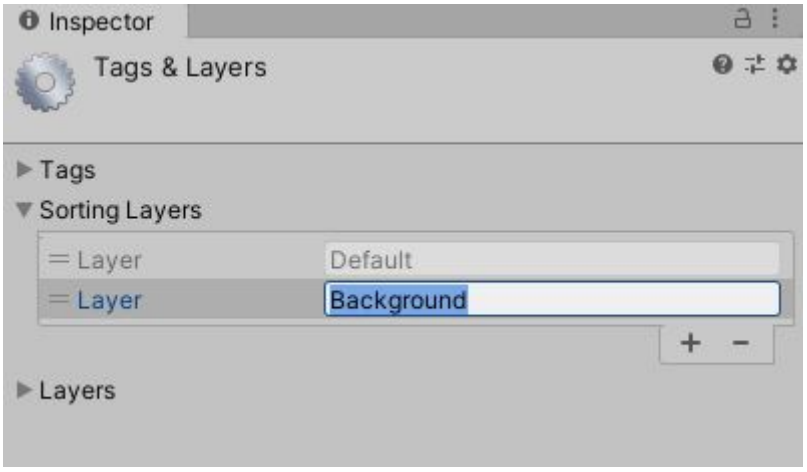
Ako bismo sada želeli da ubacimo pozadinu na ovu scenu, možemo to uraditi tako što ćemo opet iz direktorijuma Environment prevući na scenu sliku nazvanu back. Nakon ubacivanja scena će izgledati ovako:



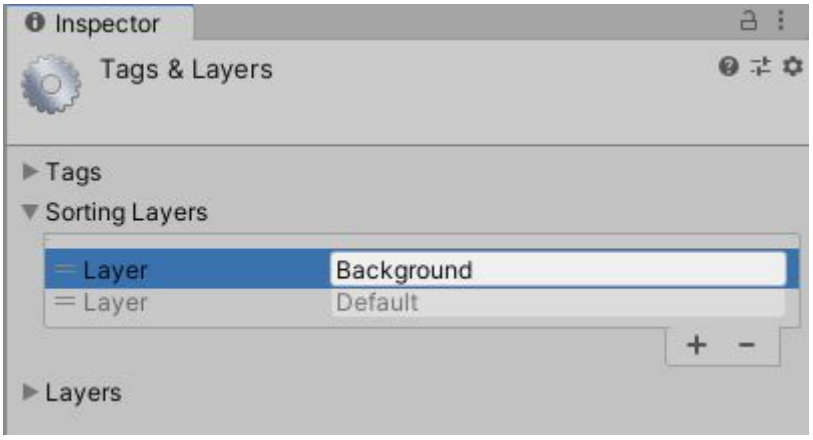
Primetićemo da pozadina lepo izgleda kada se ubaci na scenu ali problem je što je pozadina iscrtana preko spajtova koji smo ubacili ranije a idealno bi bilo da pozadina bude iscrtana ispod svega. Da bismo ovo ispravili koristimo nesto što se zove Sorting layer.
U Inspector prozoru u Sprite renderer delu nalazimo podešavanje koje se zove Sorting layer. Klikom na njega možemo dodati novi sloj za sortiranje.



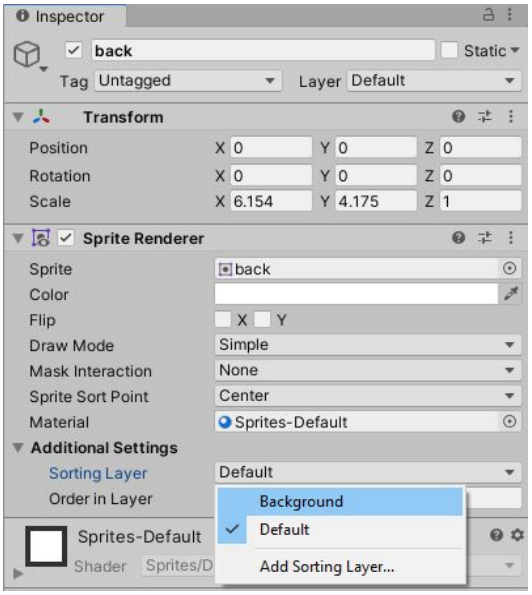
Klikom na Add Sorting Layer otvara nam se novi meni gde ako kliknemo na znak + sa desne strane možemo dodati novi sloj koji ćemo nazvati Background.



Ako želimo da se ovaj sloj nađe ispod Default sloja, moramo ga pomeriti iznad Default sloja jer ova lista Sorting layers predstavlja redosled kojim će se slojevi iscrtavati, a nama je potrebno da se pozadina iscrta pre svih ostalih slojeva.

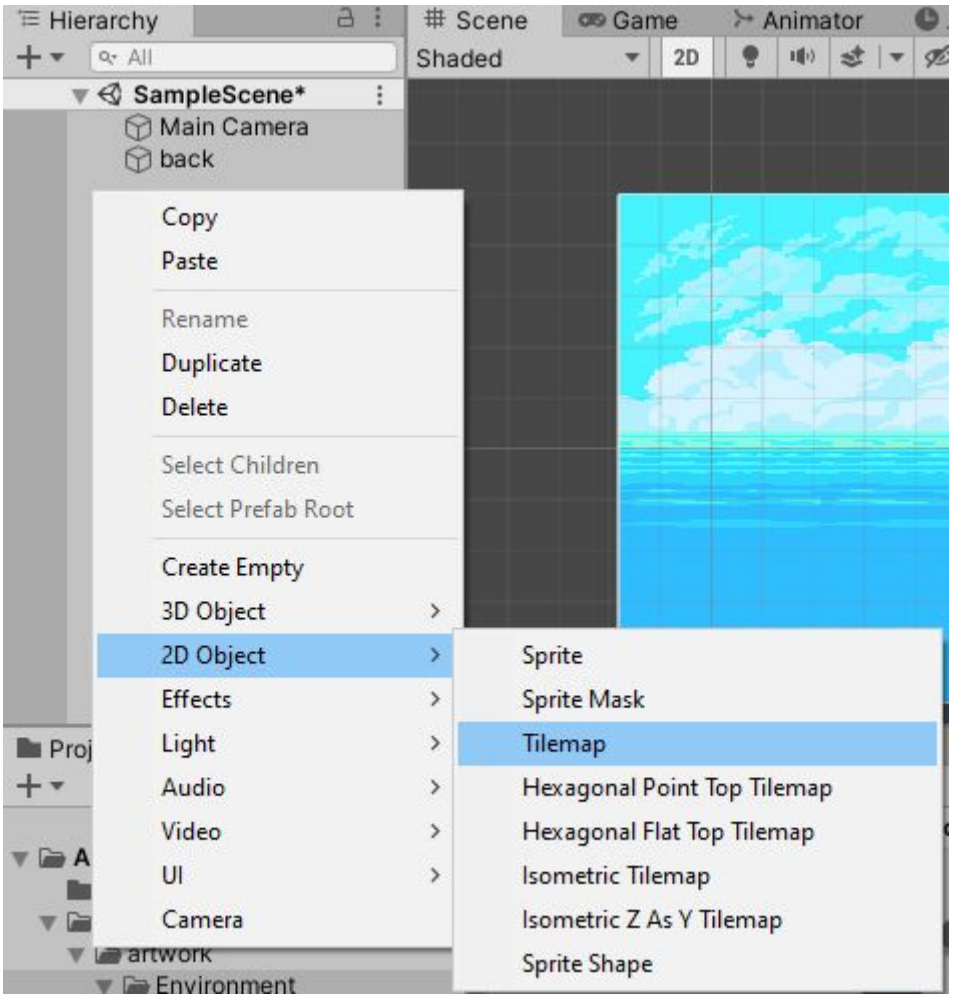


Sada možemo da kliknemo na pozadinu iscrtanu na glavnoj sceni i da promenimo Sorting layer pozadine na Background što će automatski pozadinu iscrtati ispod ostalih elemenata scene.

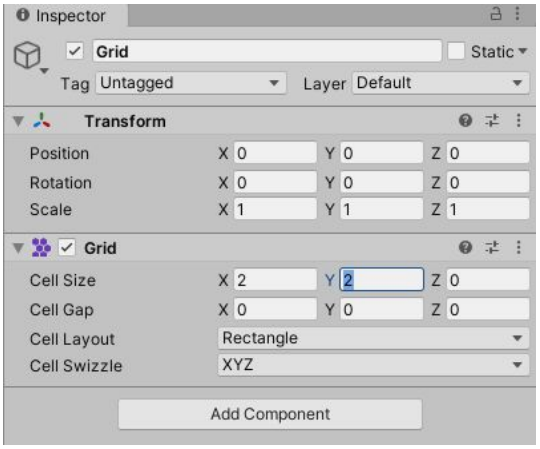


Tilemap

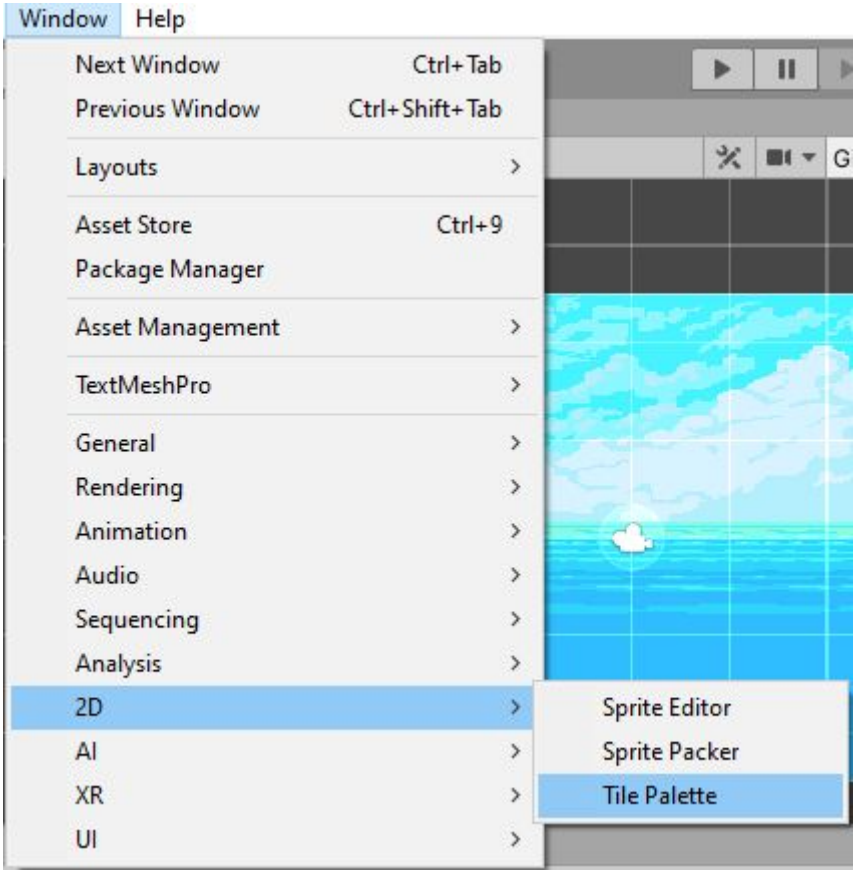
Ovakvim korišćenjem Unity-a koristimo *default sprite workflow*. Sprajtove koje smo ubacili možemo slobodno pomerati po sceni, možemo im menjati veličinu a da se oni ne vezuju za neki grid. Ovaj projekat ćemo napraviti koristeći tile based workflow tako da svi sprite-ovi koje ubacimo budu organizovani u grid.
Za početak ćemo obrisati sve objekte u sceni osim pozadine i desnim klikom na Hierarchy deo otvaraju nam se opcije gde biramo 2D object pa tilemap.



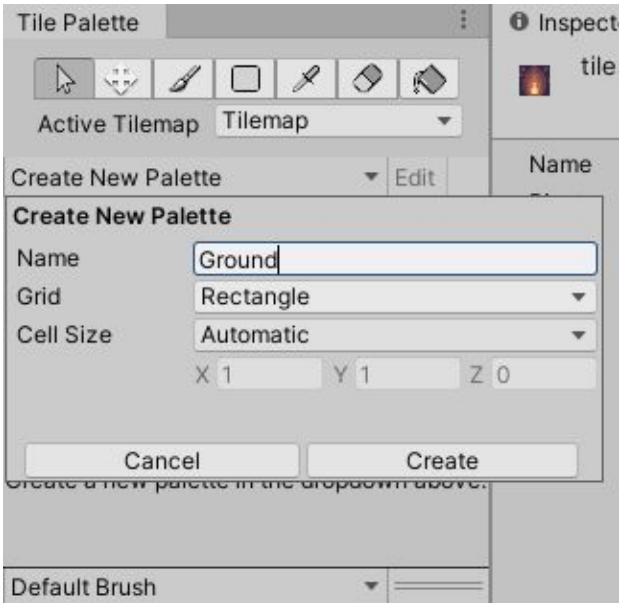
Ovo kreira objekat koji se zove Grid i koji preko cele scene stavlja mrežu.Širinu i dužinu polja te mreže ćemo uvećati kako bi polja bila preglednija tako što u Inspector prozoru podešavamo Cell Size ovog Grid objekta na 2.



Zatim ćemo otvoriti Tile Palette prozor tako što ćemo kliknuti Window opciju a zatim 2D i Tile Palette.

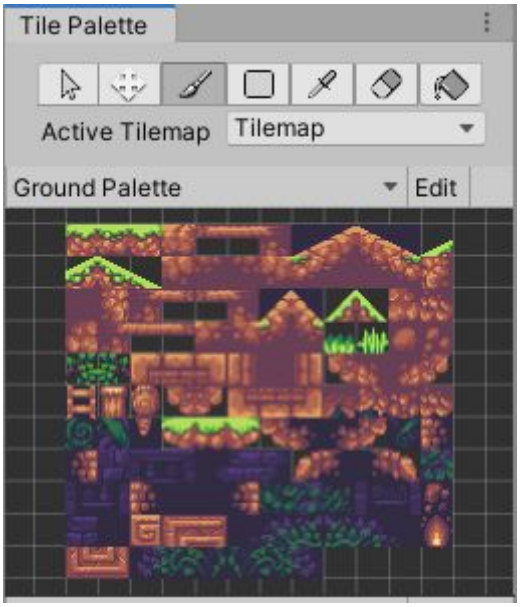


Otvoriće se Tile Palette prozor koji ćemo zakačiti za desne strane naseg Scene View-a. Tile Palette je prozor u koji ćemo ubacivati sve slike koje ćemo koristiti za dizajniranje nivoa tako da ih tu možemo izabrati i slikati njima po Grid-u koji smo napravili. Prvo ćemo napraviti paletu koju ćemo puniti slikama klikom na Create New Palette, nazvaćemo je Ground Palette, a zatim ćemo tu paletu sačuvati u Assets direktorijumu. Slike ćemo ubaciti u Tile Palette tako što ćemo izabrati sve slike iz Environment direktorijuma i prevući ih u ovaj prozor.

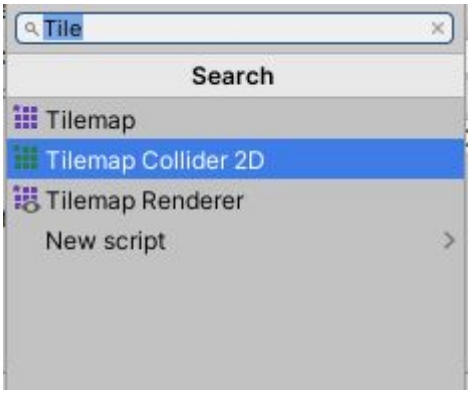


Kada prevučemo slike u ovaj prozor Unity će tražiti da odaberemo folder u kome ćemo te slike čuvati tako da ćemo napraviti direktorijum Tiles koji će čuvati slike te palete.

Sada nam se u Tile Pallete prozoru prikazuju sve slike koje smo ubacili i korišćenjem brush alata možemo slike naslikati direktno na grid.

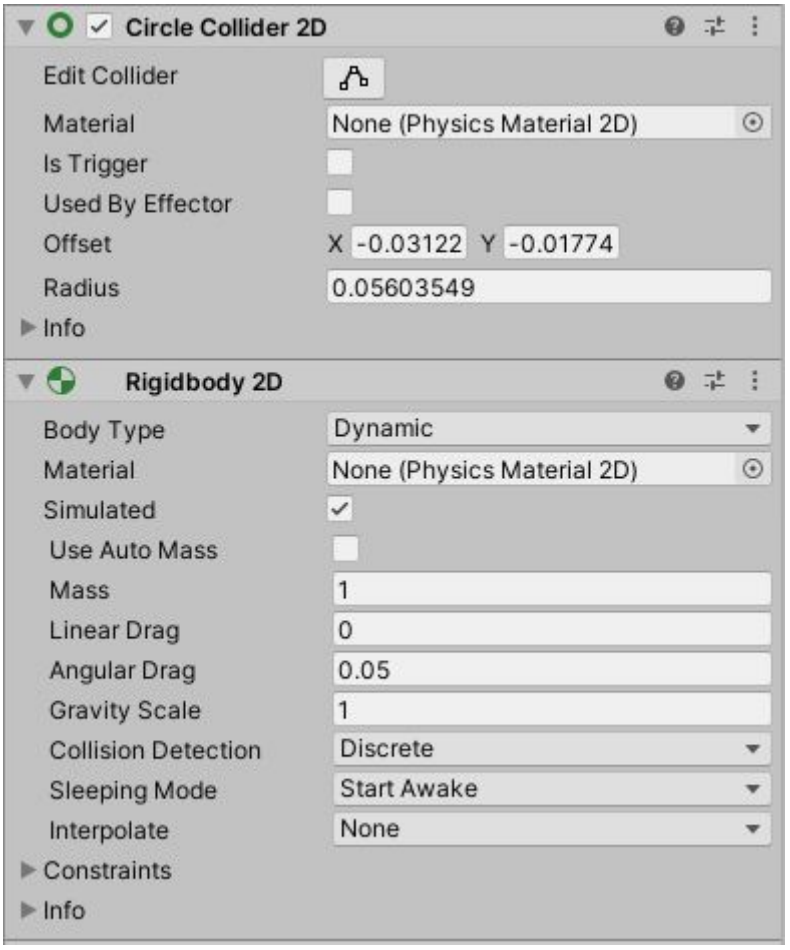


Da bi se teren koji smo postavili ponašao kao pravi teren po kome igrač i neprijatelji mogu hodati i na kome se mogu nalaziti predmeti sa kojima igrač interaguje potrebno je postaviti Collider-e na teren. To možemo uraditi tako što ćemo našem Tilemap-u priključiti komponentu koja se zove Tilemap Collider 2D. To možemo uraditi tako što prvo izaberemo Tilemap a zatim u Inspector prozoru editora pritisnemo na Add Component dugme i pretražimo Tilemap Collider. Ovo će omogućiti da objekti koji interaguju sa igračem kao i sam igrač ne propadnu kroz teren.



Collideri

Da bismo testirali da li Collider radi možemo ubaciti neki objekat u scenu kojem ćemo dodati neku Collider komponentu kao i komponentu koja će na taj objekat primeniti gravitaciju da bismo ispitali da li će se objekat zadržati na terenu. Iz Cherry poddirektorijuma koji se nalazi u Items direktorijumu na scenu prevlačimo bilo koji sprite. Zatim klikom na taj spirte u Inspector prozoru se prikazuju sve informacije o sprite-u, opet pritisnemo dugme Add Component i tražimo Circle Collider 2D, koji će nasem sprite-u pridružiti collider kružnog oblika, kao i RigidBody2D koji će na njega primeniti gravitaciju.

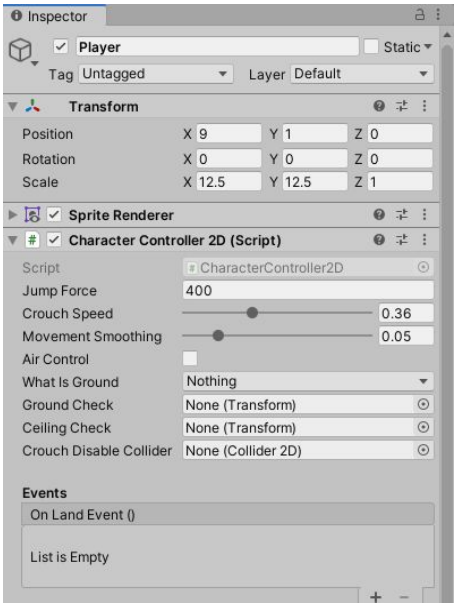


Potrebno je samo proveriti da li Circle Collider 2D pravilno oslikava konture našeg sprite-a, i ako ne veličinu i poziciju možemo promeniti klikom na Edit Collider. Naravno vrednost za Gravity Scale mora biti veća od 0 da bi objekat padao na dole.

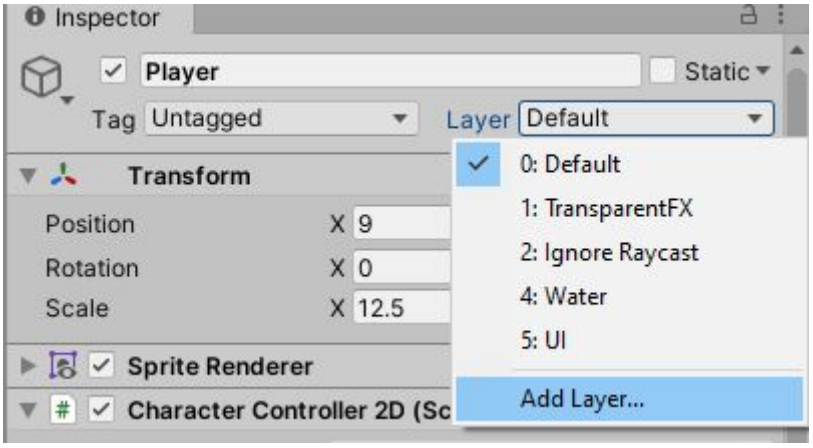
Ako sada pokrenemo igricu možemo primetiti da se objekat koji smo ubacili stvarno zaustavlja na površini terena.

Sada kada smo ubacili teren u našu igricu red je da ubacimo igrača koji će moći da se kreće po tom terenu. Jedan od najbitnijih delova pravljenja 2D igrice je kretanje igrača, iako kretanje u igrici uzimamo zdravo za gotovo, kreiranje kvalitetnog Character Controller-a koji bi upravljao kretanjem igrača može biti veoma zahtevno. Naravno, sve zavisi od toga šta želimo da nas igrač može da radi: da skače, da se pomera dok je u vazduhu, da čuči itd. Character Controller predstavlja skriptu koja kontroliše kretanje igrača i vrši složena izračuvananja koja se tiču fizike i omogućavaju našem igraču da se kreće.
Za početak ćemo u našu scenu ubaciti igrača tako što ćemo iz poddirektorijuma Player u Sprites direktorijumu izabrati bilo koji sprajt i samo ga vrevući na nasu scenu.

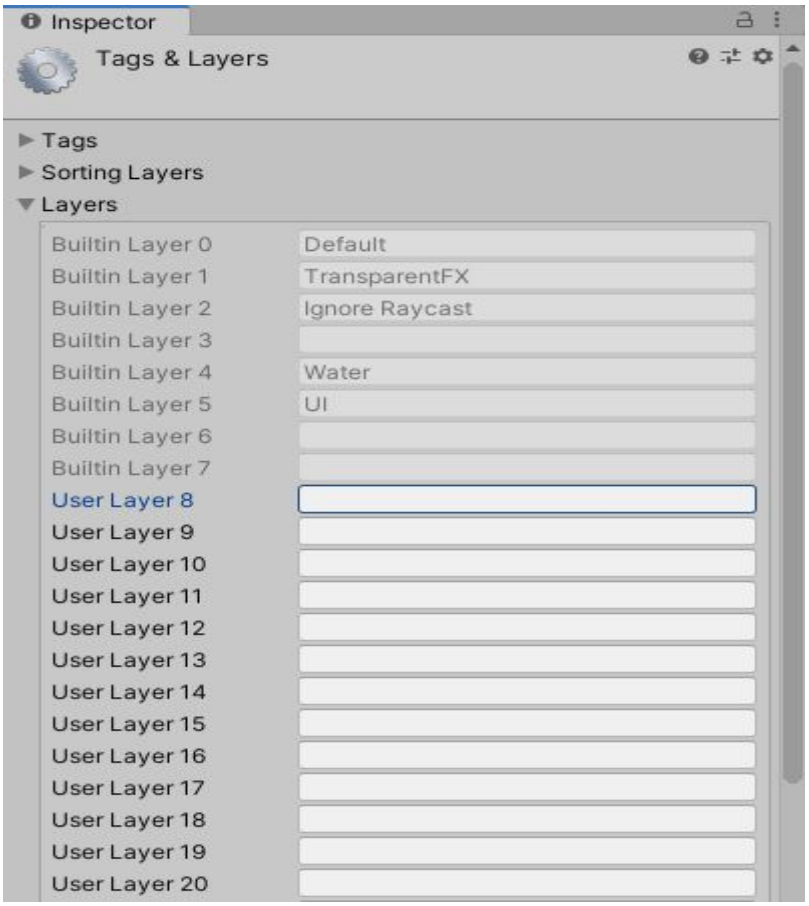
Naš igrač trenutno u sebi ima samo Sprite Renderer komponentu koja iscrtava igrača.Za ovaj projekat, kao osnovu našeg Character Controller-a koristićemo već gotovu skriptu koju ćemo kasnije menjati da bi se prolagodila našim potrebama. Potrebno je CharacterController2D.cs skriptu (koja je priložena uz ovu dokumentaciju) prevući u Unity u Assets direktorijum. A zatim je potrebno tu skriptu prevući unutar našeg igrača. Kada sada kliknemo na našeg igrača u Inspector delu prikazaće se skripta koju smo ubacili.



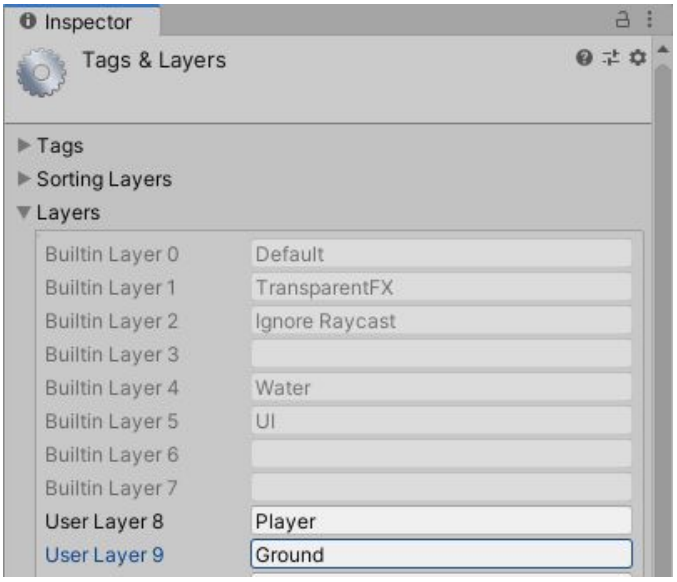
Unutar ovog dela možemo videti neke atribute koja ima skripta i koje možemo menjati da bi podesili kretanje igrača da bude optimalno za vrstu igrice koju pravimo. Od atributa vidim Jump Force i taj atribut predstavlja jačinu kojom će naš igrač skakati, trenutno ima vrednost 400 ali za potrebe naše igrice postavićemo vrednost na 700. Crouch Speed atribut predstavlja postotak brzine igrača koju će imati kada čuči i postavićemo je na 0.4. Air Control je boolean vrednost koji sktipti govori da li će igrač moći da se pomera dok je u fazi skoka i za našu igricu je potrebno da može tako da ćemo ovo postaviti na true klikom na kvadrat. Zatim je potrebno podesiti What Is Ground atribut koji će našoj sktipti naglasiti koje Layer-e može koristiti kao podlogu po kojoj može hodati. Pošto u našoj igrici trenutno imamo samo Default sloj na kojem se sve nalazi kreiraćemo nove slojeve. Prvo ćemo kliknuti na našeg igrača a zatim u Inspector delu kliknuti na Layer atribut koji se nalazi u samom vrhu Inspector-a.



Zatim kliknemo na Add Layer opciju koja otvara prozor u kojem možemo raditi sa slojevima.

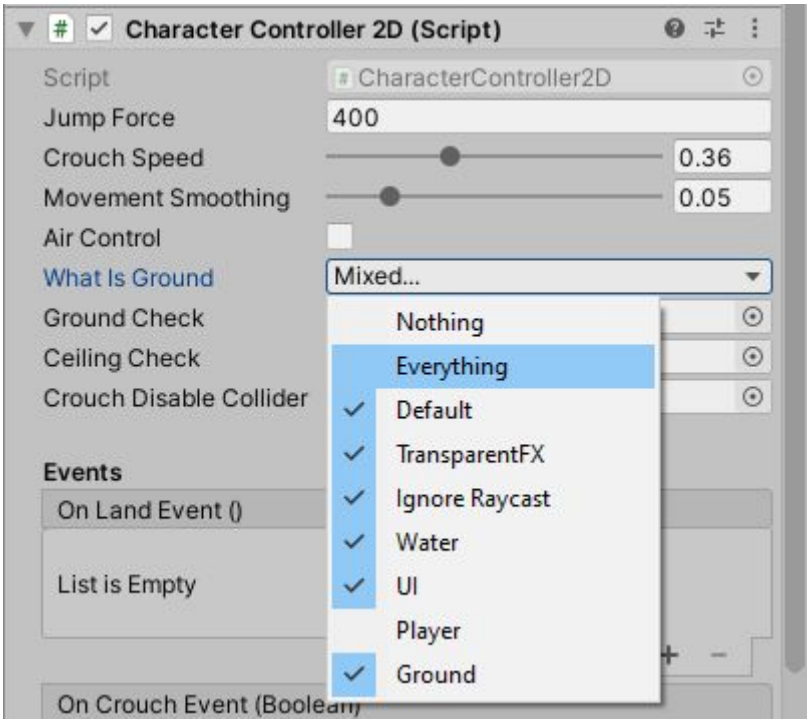


Za potrebe naše igrice dodaćemo Player sloj u kojem će se nalaziti samo igrač, i Ground sloj kojem će pripadati sve površine po kojima će naš igrač moći da hoda.



Sada možemo postaviti Layer atribut našeg igrača na Player sloj, i na sve površine po kojima igrač može hodati stavićemo da su u Ground Layeru kako bi igrač mogao da hoda po njima.

Samda možemo postaviti What Is Ground atribus naše skripte i izabraćemo sve slojeve osim samog Player sloja jer ne želimo da igrač može hodati sam po sebi.

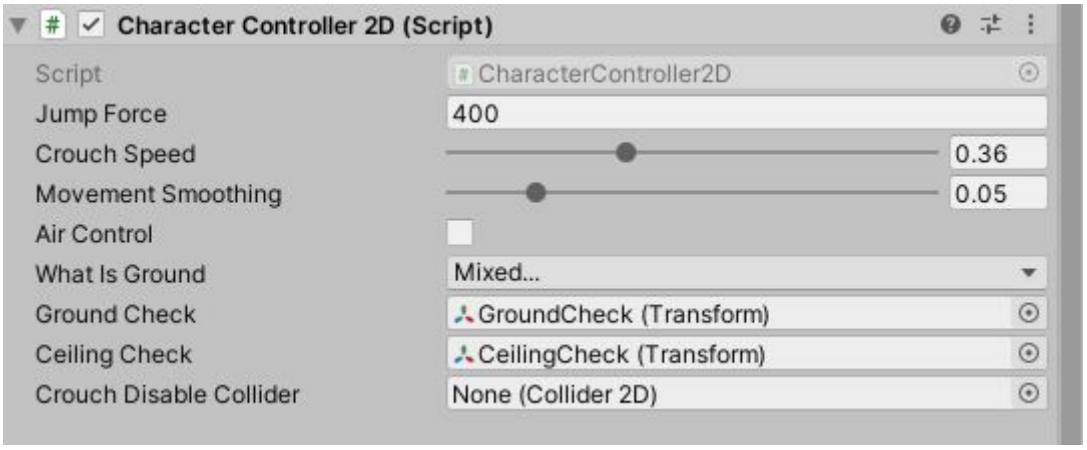


Dalje u skripti imamo 3 prazna objekta koji se zovu Check-ovi. Koristimo ih da odredimo da li je igrač trenutno na zemlji da bi odredili može li igrač skočiti, i dok igrač čuči želimo da znamo imamo li neki plafon iznad glave da bi znali može li se igrač ispraviti.Koristićemo ove prezne objekte da bi odrediti gde su te lokacije. Pritisnićemo desni klik na našeg igrača u Hierarchy prozoru i pritisnućemo Create Empty opciju. Kada sada kliknemo na Game Object objekat koji se nalazi ispod našeg igrača i ako izaberemo Move Tool iz Toolset dela prikazaće nam se gde se taj novi objekat nalazi.

Primetićemo da ovaj objekat stoji previse nisko da bi bio naš Ceiling Check tako da ćemo ga pomeriti malo na gore. Kliknemo na zelenu strelicu i povučemo ceo objekat na gore.

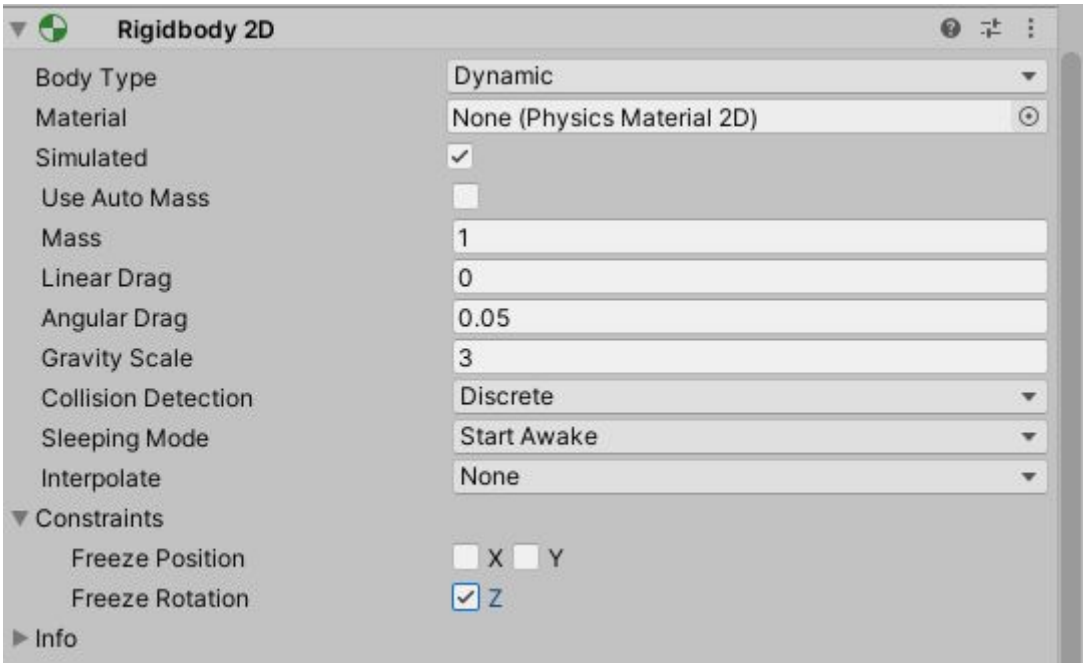
Sada ćemo kliknuti na Ctrl+D da bi napravili duplikat našeg game objekta i taj objekat ćemo pomeriti kod nogu našeg igrača da funkcioniše kao Ground Check.

Sada ćemo preimenovati prvi objekat na CeilingCheck a drugi na GroundCheck da bi ostali organizovani. Sada je potrebno da prevučemo ove objekte koje smo kreirali u atribute naše PlayerMovement.cs skripte tako da skripta sada izgleda ovako:

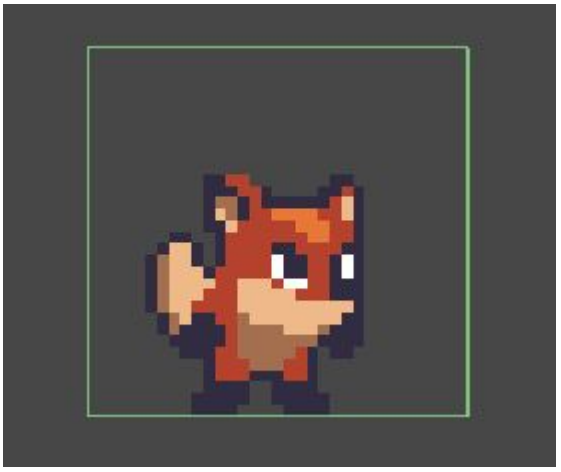


Da bi igrač mogao da interaguje sa okolonom prvo je potrebno dodati komponente u njega koje će omogućiti da na njega bude primenjena fizika kao i Collider-i da bi mogao da se sudara sa drugim objektima i terenom.

Pritisnemo na našeg igrača pa na Add Component dugme i tu dodajemo Rigidbody2D koji će na igrača primeniti gravitaciju itd. Vrednost gravitacije postavićemo na 3. Pošto ne želimo da se naš igrač rotira već samo da se kreće po 2D terenu uvek okrenut na gore potrebno je da odemo u Constraints deo i da boolean FreezeRotation za Z osu postavimo na true.



Sada je potrebno našem igralu pridružiti Collider komponentu jer ako bi sada pokrenuli igricu igrač bi samo propao kroz teren. Dobra je ideja da za ovo koristimo više Collider-a kako bi mogli da ispeglamo kolizije i učinimo kretanje realističnijim. Za našeg igrača koristićemo 2 Collidera, i to jedan CircleCollider i jedan BoxCollider. Prvo ćemo ubaciti BoxCollider tako što kliknemo na našeg igrača a zatim na Add Component i tu pronađemo BoxCollider2D i dodamo ga. Na kratko ćemo isključiti vidljivost pozadine kako bi se Collider-i bolje videli. BoxCollider koji smo ubacili prikazan je tankom zelenom linijom i primetićemo da baš i ne odgovara konturama našeg igrača.

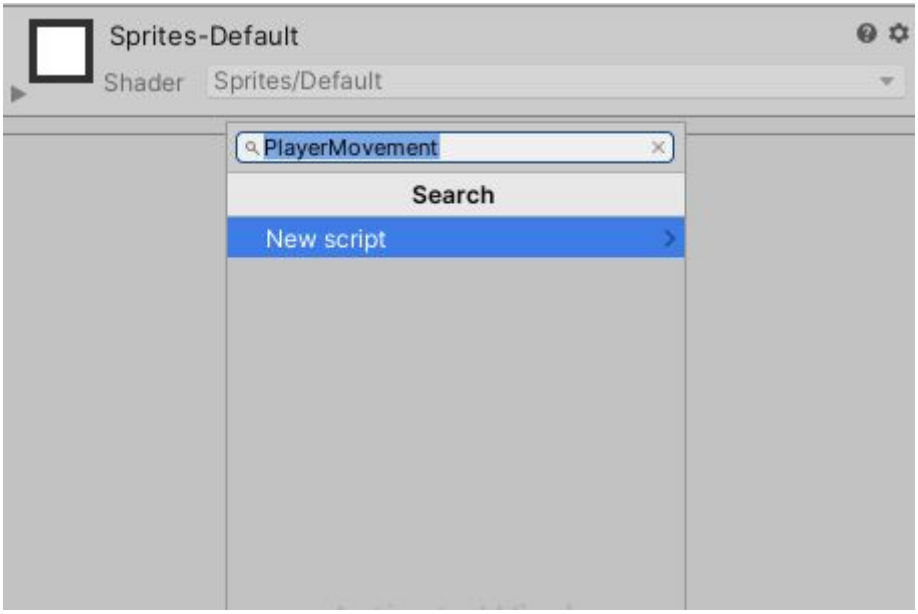


Moramo izmeniti naš Collider tako da bolje odgovara igraču. Prvo pritisnemo Edit Collider dugme a zatim menjamo dimenzije Collidera. Preporučljivo je držati Alt dugme dok menjamo visinu ili širinu Collidera kako bi se skalirao sa obe strane ose i ostao centritan.

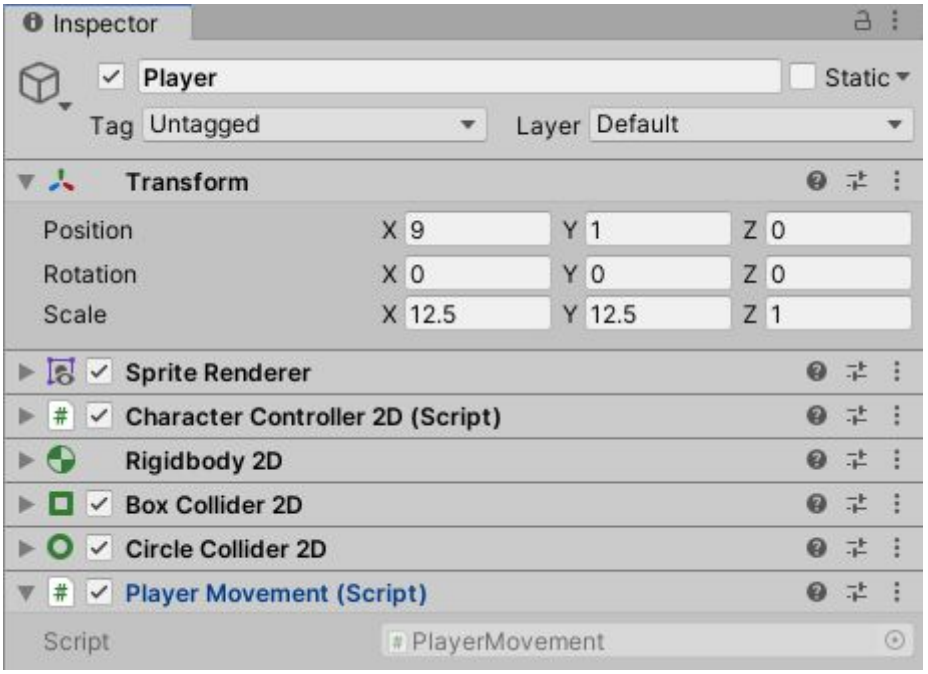
Ovaj BoxCollider smo podigli malo tako da ne ide preko nogu igrača zato što će preko nogu biti postavljem CircleCollider. Sada opet idemo na Add Component dugme i igrači dodajemo CircleCollider2D komponentu. Zatim opet idemo na Edit Collider i nameštamo ga tako da bude poravnat sa BoxCollider-om sa desne i leve strane i naravno mora da pokrije noge igrača.

Kretanje

Ako sada pokrenemo igricu primetićemo da igrač pada na teren koji smo postavili u Ground Layer i igrač je sada spreman da počne da se kreće. Izabraćemo našeg igrača i dodaćemo skriptu koja će kontrolisati kretanje igrača. Opet kliknemo Add Component dugme i pretražićemo PlayerMovement i i zatim New Script.



Zatim pritisnemo Create and Add dugme i u Inspector prozoru pojaviće se Player Movement(Script) komponenta i unutar nje Script atribut postavljen na skriptu koju smo upravo kreirali.



Sada je potrebno da skriptu otvorimo u nekom edioru da bismo mogli da je menjamo. Prva stvar koju moramo uraditi je proslediti referencu CharacterController2D skripte PlayerMovement skipti bismo onda mogli da nateramo CharacterController2D da pomera našeg igrača. To ćemo uraditi tako što ćemo u PlayerMovement skirpti na samom početku dodati referencu na objekat tipa CharacterController2D što je prikazano u sledećem kodu:

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

public class PlayerMovement : MonoBehaviour
{

    public CharacterController2D controller;

    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Sada će nam se u Player Movement komponenti u Inspector prozoru pojaviti jedan prazan slot naslovljen sa Controller i u taj prazan slot je potrebno prevući CharacterController komponentu.

I sada svaki put kada u PlayerMovement skripti napišemo controller referenciramo se na CharacterController skriptu.

Kada sada želimo da protumačimo korisnikov ulaz to možemo uraditi koristeći Input klasu i njenu metodu GetAxisRaw sa parametrom ‘’Horizontal’’ tako da vraća korisnikov ulaz koji se odnosi na horizontalnu osu. Ova metoda kao povratnu vrednost ima broj od -1 do 1 koje će se menjati u zavisnosti od ulaza korisnika. Ako pritisnemo na tastaturi levu strelicu ili slovo A GetAxisRaw će vratiti vrednost -1 dok će za desnu strelicu ili slovo D vratiti vrednost 1. Ovu metodu ćemo koristiti da odredimo gde se želimo pomerati po horizontalnoj osi.

```
    void Update()
    {
        Input.GetAxisRaw("Horizontal")
    }
}
```

Moguće je sada koristiti ovu vrednost da bi pomerali igrača unutar Update metode ali nije preporučljivo pomerati igrača ovde. Za pomeranje je preporučljiva druga metoda koja je posvećena fizici igre koja se zove FixedUpdate i radi na sličan način kao Update ali umesto da se poziva svaki put kada svaki put kada računar želi da iscrta frejm na slici poziva se fiksiran broj puta u sekundi. U nastavku ćemo kreirati FixedUpdate metodu i koristićemo je za pomeranje igrača, dok će u tumečenje korisnikovog ulaza biti rađeno u Update metodi.

Da bismo prosledili vrednost ulaza iz jedne metode u drugu biće nam potrebna promenljiva koju ćemo nazvati horizontalMove i biće tipa float sa podrazumevanom vrednosti 0, a onda ćemo u Update metodi napisati da horizonralMove preuzima vrednost Input.GetAxisRaw metode. Sada je ostalo samo da pomerimo našeg igrača i to ćemo uraditi tako što ćemo se referencirati na CharacterController2D i iskoristiti njenu Move metodu koja prima realan broj i pomera igrača za tu vrednost a pošto želimo da se pomeramo u skladu sa korisnikovim ulazom Move metodi ćemo proslediti horizontalMove. Move metoda pored ove vrednosti od

ulaza očekuje još dva parametra, jedan određuje da li igrač treba da čuči a drugi određuje da li igrač treba da skače i sada ćemo ove te vrednosti postaviti na false da bi testirali skriptu.

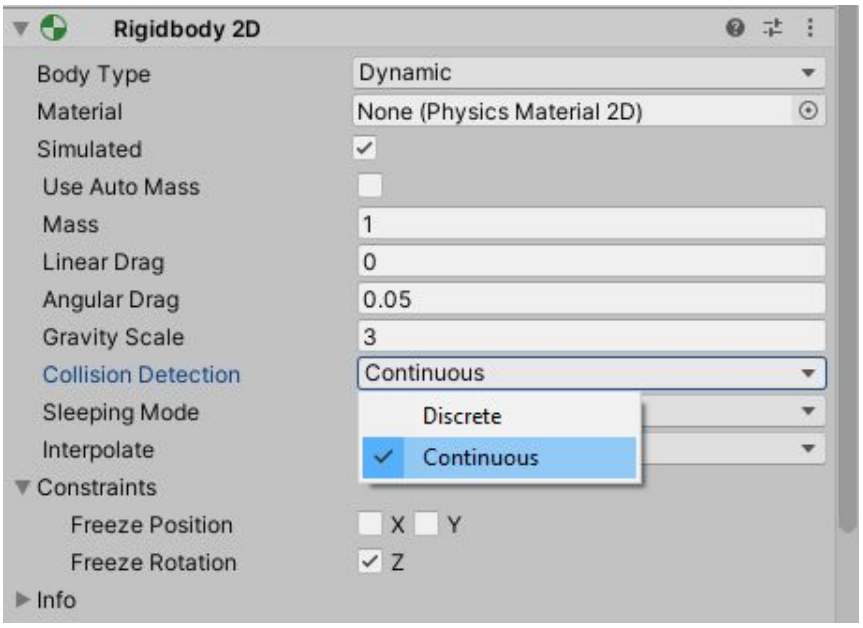
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{

    public CharacterController2D controller;
    float horizontalMove=0f;

    // Update is called once per frame
    void Update()
    {
        horizontalMove = Input.GetAxisRaw("Horizontal");
    }

    void FixedUpdate()
    {
        controller.Move(horizontalMove,false,false);
    }

}
```

Sada skripta može odrediti pravi smer u kojem treba pomeriti igrača i vrši pomeranje ali nema nikakvu kontrolu nad brzinom kojom će se igrač kretati. Da bi imali kontrolu i nad brzinom igrača napravićemo još jednu promenljivu koja će se zvati moveSpeed i biće postavljena na vrednost 40 tako da kada horizontalMove promenljiva preuzima vrednost korisnikovog ulaza, taj korisnički ulaz ćemo pomnoziti sa runSpeed tako da ako korisnik želi da se igrač kreće desno, vrednost runSpeed biće 40 dok ako želi da se kreće levo imaće vrednost -40. Poslednja stvar koju moramo uraditi da bi se igrač pomerao kako valja je da kad god kažemo igraču da se pomeri, vrednost za koju se pomera množimo sa Time.fixedDeltaTime koje nam vraća vreme proteklo od poslednjeg poziva metoda FixedUpdate. Ovo će osigurati to da se igrač pomeri svaki put za istu vrednost bez obzira na to koliko puta je FixedUpdate metoda pozvana u sekundi tako da će brzina našeg igrača biti ista na svim platformama i sistemima na kojima bude pokretana. PlayerMovement skripta će nakon izmena izgledati ovako:

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

public class PlayerMovement : MonoBehaviour
{

    public CharacterController2D controller;

    public float runSpeed=40f;

    float horizontalMove=0f;

    // Update is called once per frame
    void Update()
    {
        horizontalMove = Input.GetAxisRaw("Horizontal")*runSpeed;
    }

    void FixedUpdate()
    {
        controller.Move(horizontalMove * Time.fixedDeltaTime(),false,false);
    }

}
```

Sada će nam se u Inspector prozoru unutar Player Movement komponente pojaviti i atribut Run Speed čiju vrednost možemo menjati direktno iz Unity-a jer smo u skripti vidljivost ove promenljive postavili na public.

Kada pokrenemo igricu možemo sada pomerati našeg igrača po nivou koji smo napravili. Ako bi igrač pao sa jedne platforme na drugu desiće se da malo naizgled potone u platformu pa se vrati na površinu. Da bismo izbegli ovaj problem u igračevoj RigidBody2D komponenti ćemo Collision Detection atribut da promenimo sa Descrete na Continuous što bi trebalo da spreči takve greške.

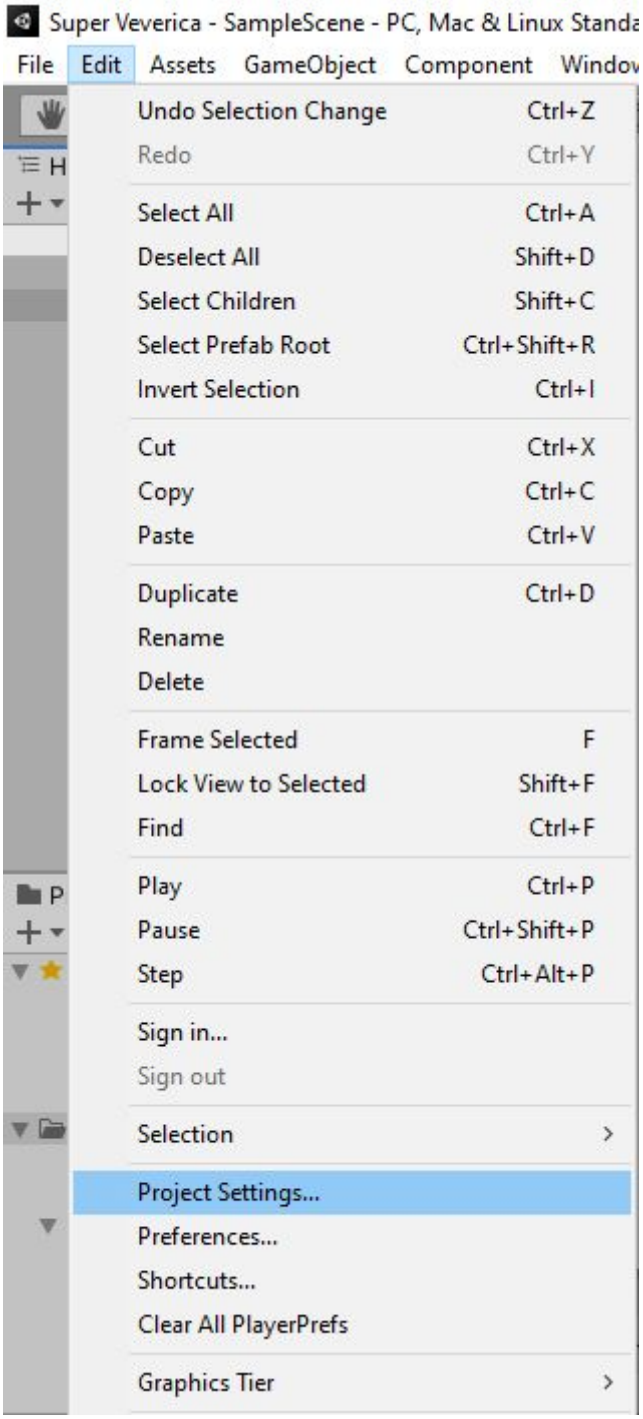


Sada je potrebno da nateramo igrača da skače što je dosta lakse uraditi od horizontalnog kretanja. Sve što moramo uraditi je da u Update metodi PlayerMovement skripte proverimo da li je dugme za skok pritisnuto i to ćemo proveriti koristeći Input klasu i njenu metodu GetButtonDown koja proverava da li je neki taster pritisnut a pošto u našem slučaju želimo da proverimo dugme za skakanje, metodi ćemo proslediti ‘’Jump’’ i ako je stisnut taster za skok metoda će vratiti boolean vrednost true i tada je potrebno naterati igrača da skoči. Naravno, želimo skok da obavimo u FixedUpdate metodi tako da ćemo kreirati novu promenljivu boolean tipa i nazvaćemo je jump sa podrazumevanom vrednosti false i kada u Update metodi odredimo da je taster za skok stisnut potrebno je ovu promenljivu postaviti na true. U FixedUpdate metodi sada pri pozivu Move metode CharacterController-a umesto da vrednost za skok uvek bude false postavićemo je na jumo tako da igrač skače kada god korisnik to želi i posle toga naravno vrednost jumo promenljive moramo postaviti na false da igrač ne bi zauvek skakao

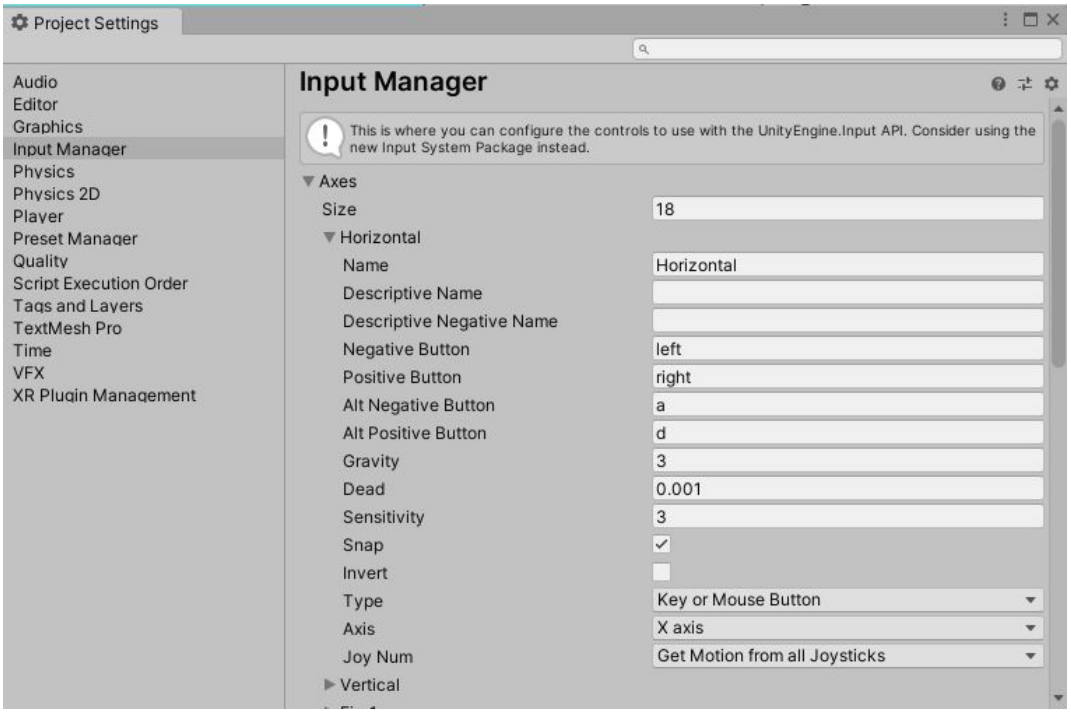
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{

    public CharacterController2D controller;

    public float runSpeed=40f;

    float horizontalMove=0f;
    bool jump=false;

    // Update is called once per frame
    void Update()
    {
        horizontalMove = Input.GetAxisRaw("Horizontal")*runSpeed;

        if(Input.GetButtonDown("Jump")){
            jump=true;
        }
    }

    void FixedUpdate()
    {
        controller.Move(horizontalMove * Time.fixedDeltaTime ,false,jump);
        jump=false;
    }

}
```

Ako sada pokrenemo igricu možemo videti da igrač skače kada pritisnemo taster Space i moguće ga je kontrolisati dok je u vazduhu.

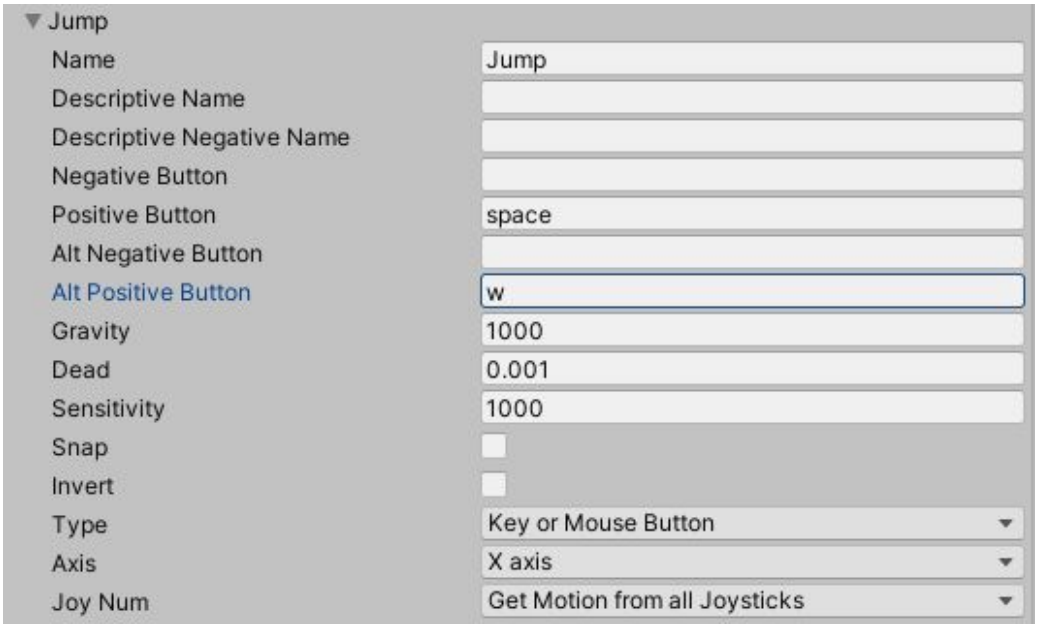
Sada se postavlja pitanje kako je to Unity povezao vrednost ‘’Jump’’ sa tasterom Space i vrednost ’’Horizontal’’ povezao sa tasterom A i levom strelicom odnosno tasterom D i desnom strelicom. To je nešto što vizuelno možemo konfigurisati. Ako u Unity-u idemo na Edit pa Project Settings otvara nam se Input Menager prozor.



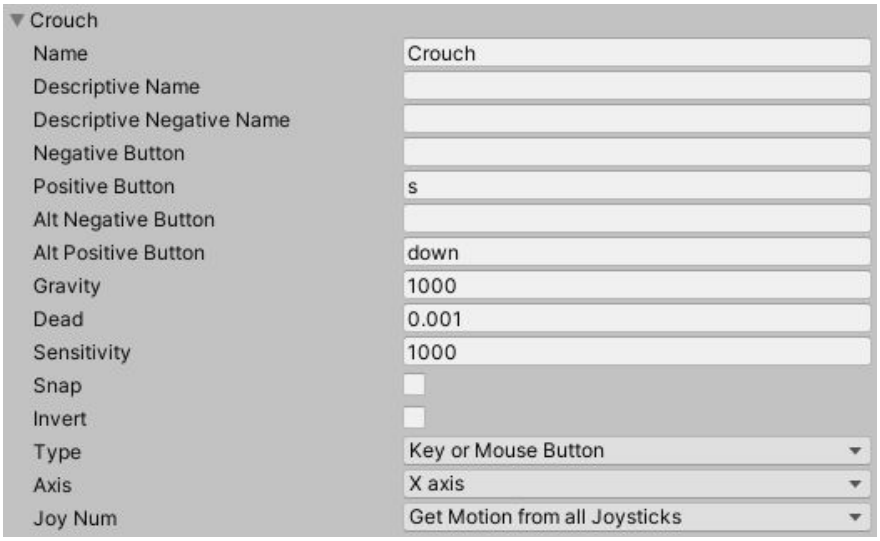
Ako u Input Menager prozoru pogledamo Horizontal sekciju tu možemo videti da je tu Horizontalno kretanje nazvano samo Horizontal i tako ga skripta prepoznaje. Takođe možemo videti sve tastere koji označavaju kretanje unutar naše igrice. Te tastere tu naravno možemo promeniti.



Ako pogledamo dole videćemo i da postoji sekcija koja se zove Jump i kada je otvorimo vidi se da je samo Space taster postavljem kao taster za skok. Sada možemo dodati taster W kao alternativni taster za skakanje.



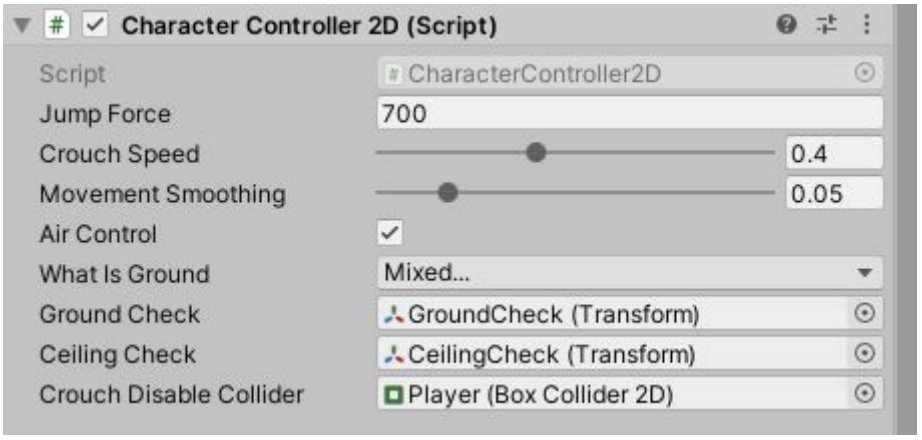
U nastavku ćemo postaviti i dugme za čučanje. Najpre ćemo napraviti duplikat Jumo sekcije desnim klikom na Jump i zatim biramo Duplicate opciju i dobijamo novu sekciju koja se takođe zove Jump i ima sve opcije iste. Ovu sekciju ćemo preimenovati u Crouch i umesto Space tastera i w tastera napravićemo tako da igrač može čučnuti na S taster ili strelicu na dole.



Sada kada smo podesili sve korisničke ulaze vreme je da omogućimo igraču da čučne. Ovo ćemo uraditi na sličan način kao i za skakanje. Napravićemo novu boolean promenljivu crouch koja će imati podrazumevanu vrednost false i u Update metodi ćemo, na isti način kao i za skok, pitati da li je pritisnut taster za čučanje i ako jeste postavićemo promenljivu crouch na true. U FixedUpdate metodi ćemo zatim u pozivu metode Move proslediti i crouch promenljivu umesto vrednosti false. Čučanje ne funkcioniše na isti način kao skakanje tako da kada igrač čučne ne treba se odmah ispraviti već tek kada taster za čučanje bude pušten. Srećom možemo registrovati i kada korisnik pusti dugme tako da ćemo u Update metodi ispod dela skripte koji proverava da li je stisnut taster za čučanje dodaćemo kod koji će proveravati da li je taster za čučanje pušten koristeći metodu GetButtonUp klase Input i prosledićemo joj vrednost ‘’Crouch’’, i ako jeste pušten taster za čučanje vrednost promenljive crouch postavljamo na false i igrač se tada ispravi.

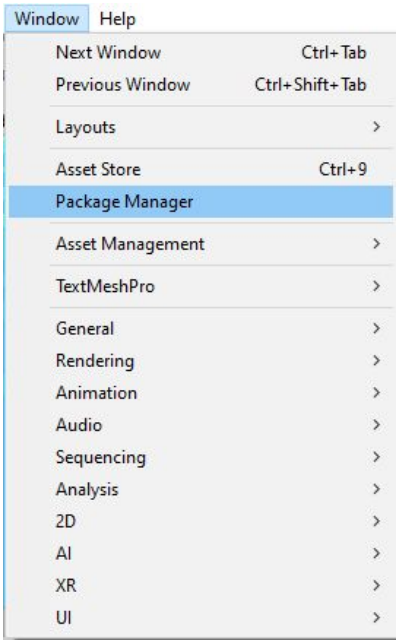
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{

    public CharacterController2D controller;

    public float runSpeed=40f;

    float horizontalMove=0f;
    bool jump=false;
    bool crouch=false;
    // Update is called once per frame
    void Update()
    {
        horizontalMove = Input.GetAxisRaw("Horizontal")*runSpeed;

        if(Input.GetButtonDown("Jump")){
            jump=true;
        }

        if(Input.GetButtonDown("Crouch")){

            crouch=true;

        }else if(Input.GetButtonUp("Crouch")){

            crouch=false;
        }
    }

    void FixedUpdate()
    {
        controller.Move(horizontalMove * Time.fixedDeltaTime ,crouch,jump);
        jump=false;
    }

}
```

Kada sada pokrenemo igricu sve će izgledati kao i ranije jos kretanje igrača još uvek nije animirano ali ako držimo taster S i pomeramo igrača levo-desno možemo primetiti da se igrač kreće sporije dok je taster pritisnut što znači da skripta radi. Još jedna stvar koja se mora desiti kada igraš čučne je to da sada može da prođe ispod nekih prepreka ali za to mu smeta njegov BoxCollider koji se i dalje sudara sa terenom. Pošto samo BoxCollider ometa igrača da se provuče ispod terena možemo privremeno onesposobiti BoxCollider tako da je igrač trenutno predstavljen samo CircleCollider-om. To nam je omogućeno u CharacterController2D skripti koja u sebi ima atribut koji se zove CrouchDisableCollider sa praznim slotom. Ako u taj prazan slot prevučemo BoxCollider igrača onda će on biti zanemaren kada igrač čuči i igrač će moći onda da se provlači ispod terena.

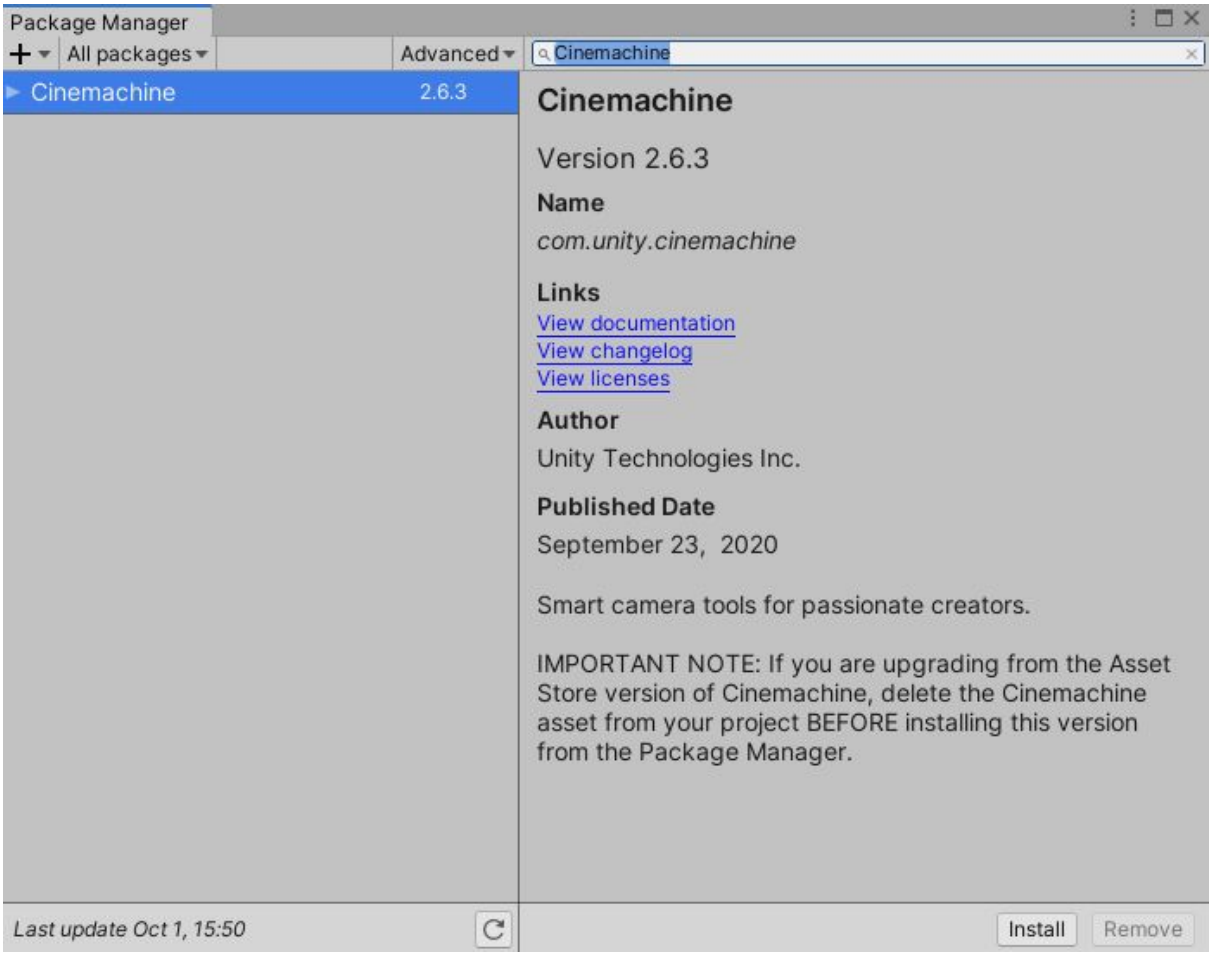


Kamera

Pošto sada naš igrač može da se pomera neophodno je namestiti kameru igrice da prati igrača gde god on ide, tako da ćemo unastavku podesiti kameru kako bi igranje igre izgledalo lepše i dinamičnije. Srećom, Unity je učinio proces implementacije dinamičke kamere mnogo lakšim dodavanjem jako dobrog seta mogučnosti koji se zove Cinemachine i koji ćemo iskoristiti u našoj igrici. Cinemachine ne dolazi već instaliran u Unity-ju već ga je potrebno ubaciti. To možemo lako uraditi klikom na Window opciju iz toolbara i izabiranjem Package Menager opcije.

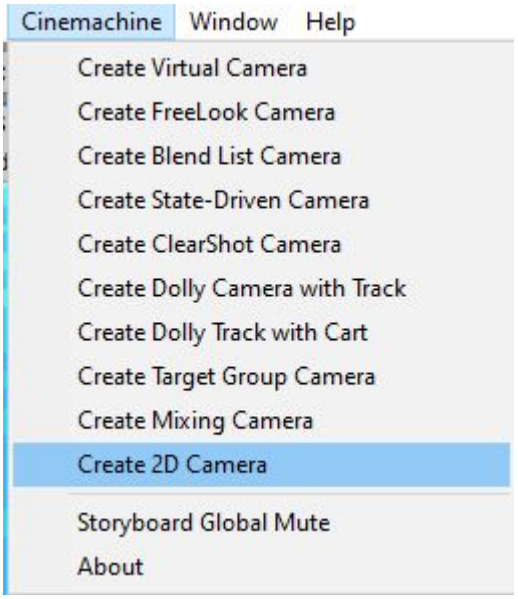


Ova opcija otvara Package Menager prozor gde možemo videti sve različite pakete koji su trenutno ubačeni u našu igricu. Sa desne strane možemo pretražiti Package Menager kucanjem Cinemachine u deo za pretragu i klikom na dugme Install možemo ubaciti Cinemachine u našu igricu.

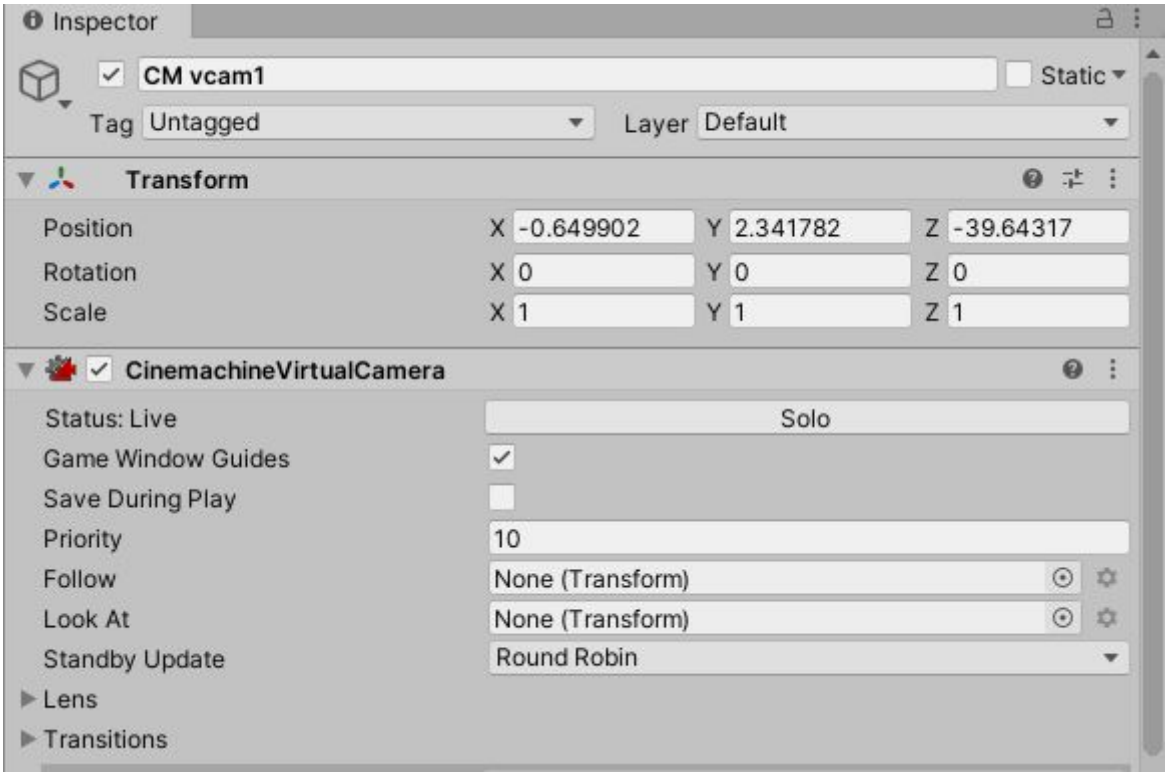


Sada kada u listi instaliranih paketa stoji Cinemachine znamo da smo uspešno instalirali Cinemachine paket i možemo početi da ga koristimo. U Toolbar delu Unity prozora pojaviće

se Cinemachine opcija u kojoj možemo izabrati vrstu kamere koju želimo da ubacimo u naš projekat. Pošto mi pravimo 2D igricu potrebna nam je 2D kamera tako da ćemo nju izabrati i tako je ubaciti u projekat.

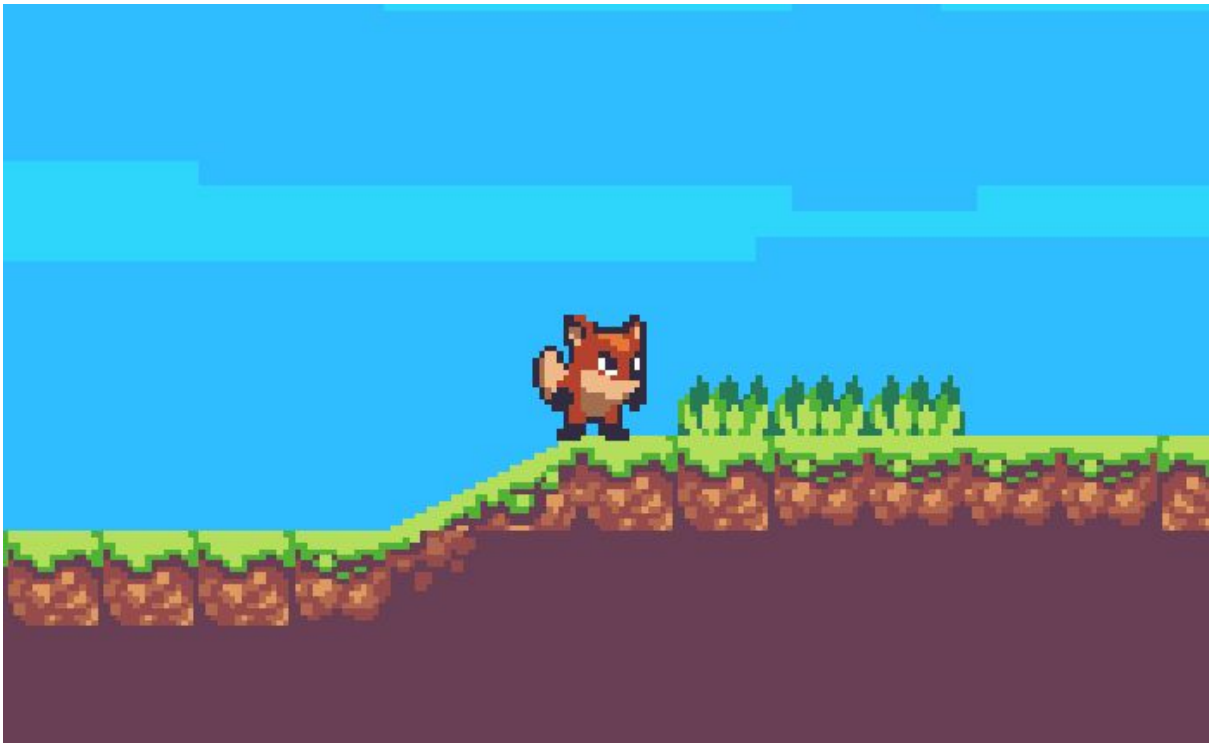


Ovo što smo sada kreirali se naziva virtuela kamera koja je praktično prazan objekat koji nije vidljiv u našoj sceni i jedino sto ima je pozicija, rotacija i još neke dodatne osobine kamere. Cinemachine je odličan zato što može svoje osobine virtuelne kamere primeniti na našu glavnu kameru.



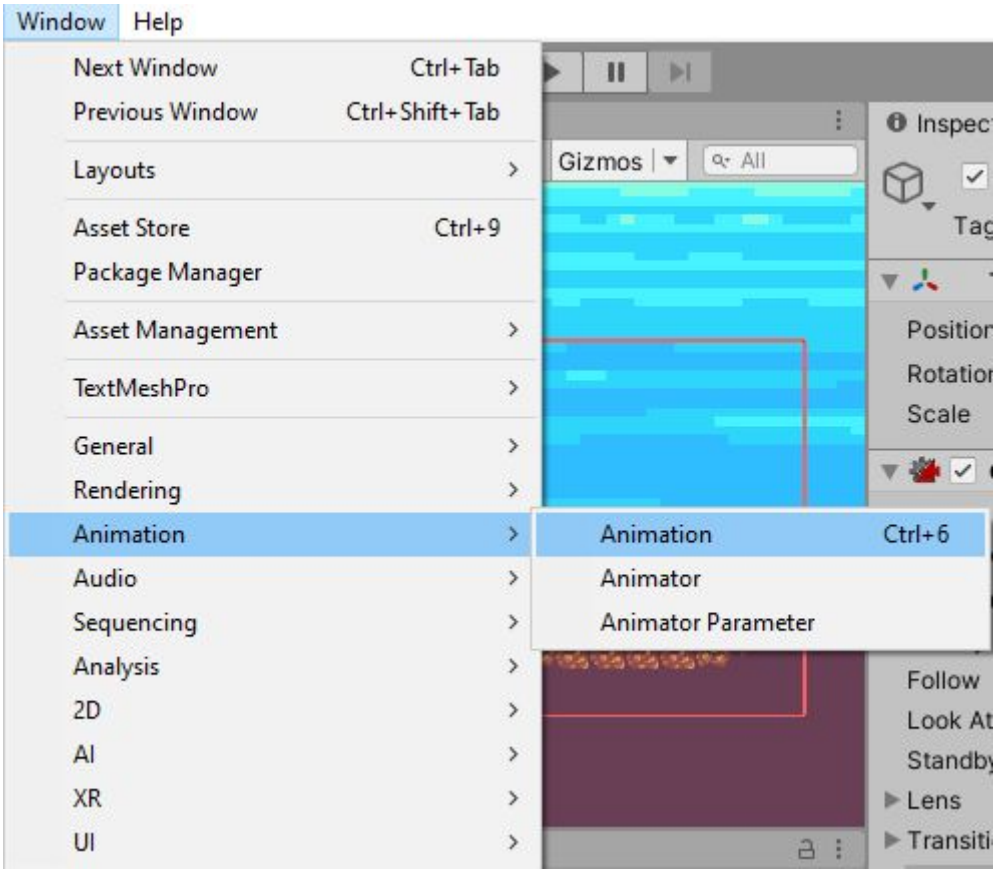
U Inspector delu pod CinemachineVirtualCamera komponentom možemo videti mnoštvo opcija koje se mogu primeniti na kameru i većinu nećemo menjati, ali kao bitan je ovaj atribut Follow koji trenutno ima prazan slot i u koji možemo ubaciti referencu na neki objekat koji želimo da naša kamera prati. Naravno mi želimo da kamera prati našeg igrača tako da ćemo

izabrati objekat koji predstavlja igrača i prevući ga u ovaj prazan slot. Odmah u SceneView-u možemo primetiti da se kamera iste sekunde vezala ta našeg igrača i ako sada pokrenemo igricu možemo videti da dinimačka kamera već funkcioniše i da prati igrača tako da uvek stoji na sredini ekrana.



Animacije

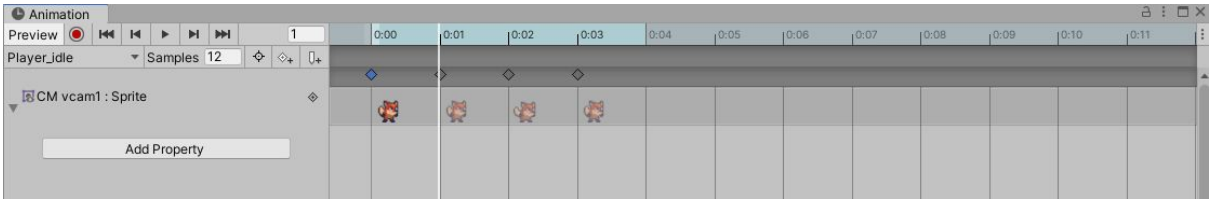
Sada kada imamo igrača koji može da se kreće i da se sudara sa terenom sa kamerom koja ga stalno prati potrebno je animirati njegovo kretanje. Postoje dva načina na koji se ovo može uraditi. Prvi način i verovatno najčešće korišćeni je Sprite Sheet Animation, i predstavlja crtanje različitih ali jako sličnih slika u nekom kratkom vremenu jednu po jednu. Ocu tehniku ćemo koristiti u našoj igrici jer je u Sunnyland Asset-u za našeg protagonistu već ima Sprite Sheet Animation postavljen baš na ovaj način. Druga tehnika je Skeletal Animation i funkcioniše tako što ‘’kosti’’ vezujemo za naš Sprite i animiranjem tih kostiju menjaće se i izgled Sprite-a. Ovaj način je koristan jer nam omogućava da samo jednom iscrtamo sliku toga što želimo da animiramo i da onda pomeranjem kostiju menjamo tu sliku koja se prikazuje koristniku. Mi ćemo naravno koristiti Sprite Sheet Animation tehniku.
Trenutno u našoj igrici kretanje nije animirano tako da možemo početi sa ubacivanjem Animation klipova.
Prvo ćemo otvoriti Animation prozor koji se nalazi u Window>Animation>Animation.



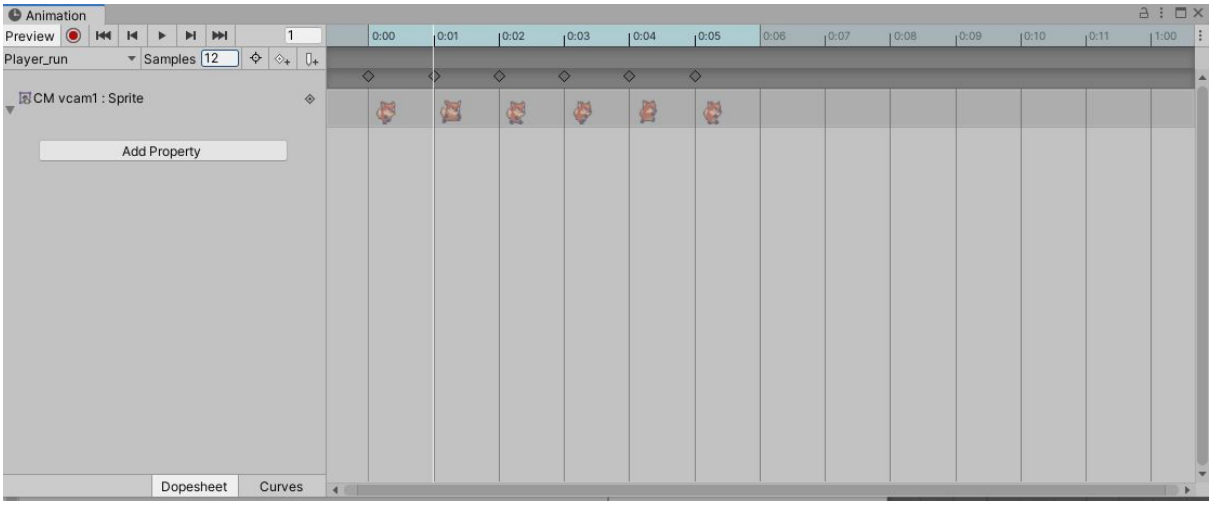
Ovo otvara Animation prozor u kojem piše da prvo moramo napraviti Animatora i Animation Clip ako želimo da animiramo našeg igrača. Sada pritisnemo dugme Create koje će od tražiti da se izabere lokacija na kojoj će animacija biti sačuvana, na šta ćemo mi krairati Animation direktorijum unutar Assets direktorijuma i Animation klip koji kreiramo ćemo nazvati Player_idle koji će predstavljati animaciju koja se korisniku prikazuje kada igraš miruje. Sada nam se u Animation prozoru prikazuje prazan Timeline u koje je potrebno prevući Sprite-ove koje želimo da animiramo.

U Assets>Sunnyland>artwork>Sprites>Player direktorijumu možemo pronaći sve slike koje su potrebne za animiranje grupisane u direktorijume.

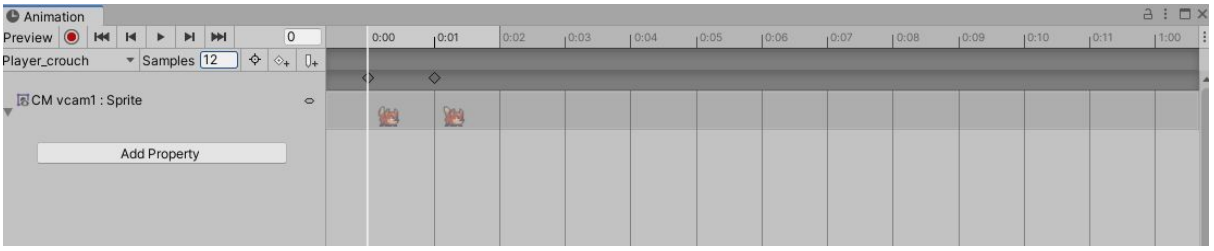
Kada uđemo u idle direktorijum nalazimo 4 slike koje predstavljaju celu idle animaciju. Sada možemo izabrati sve četiri slike i prevući ih u naš Timeline u Animator prozoru. Ovo će nam prikazati četiri Key frejma koji predstavljaju četiri slike koje će se smenjivati jedna za drugom. Ako bismo sada pokrenuli animaciju videli bismo da je animacija jako brza i da bi to popravili moramo promeniti broj slika koje će biti prikazivane u sekundi. To možemo promeniti tako što u Animator prozoru pronađemo Samples vrednost koja kontroliše broj prikazanih slika i promenićemo tu vrednost na 12 da bi usporili animaciju i ako sada pokrenemo animaciju možemo primetiti da izgleda dosta bolje.



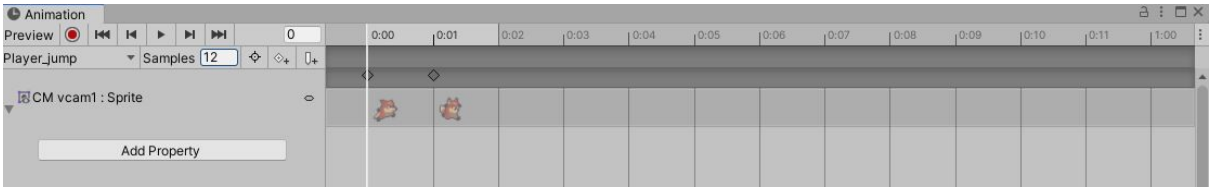
Dalje pritiskom na Player_idle biramo opciju Create New Clip i kreiranmo animaciju koja će se zvati Player_run i predstavljaće animaciju kada igrač trči. Ako sada iz run direktorijuma izaberemo sve Sprite-ove i ubacimo ih u naš Timeline i opet smanjimo vrednost Samples atributa na 12, uspešno smo napravili animaciju za trčanje igrača.



Slično ćemo odraditi i za čučanje igrača tako da Izgled animacije u Animatoru izgleda ovako:



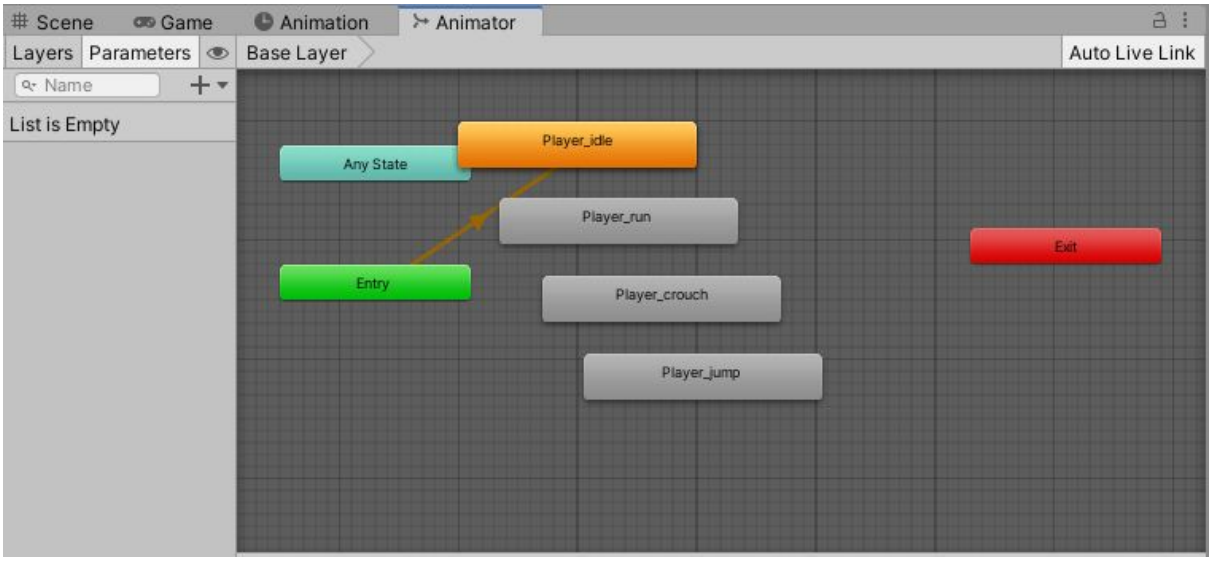
Ako isto uradimo i za animaciju za skakanje dobićemo ovo u animatoru:



Ako bismo pustili animaciju skoka primetili bismo da izgleda jako čudno zato što je naša animacija za skok sačinjena od samo 2 Key frejma: jednog kada igrač ide gore i jednog kada igrač pada. Jedno rešenje je da se ova animacija podeli na 2 dela: jedan deo bi bio animacija kada igraš skoči i ide na gore a druga animacija bi bila deo kada igraš dođe do vrhunca skoka i onda krene da se vraća ali lakše rešenje bi bilo da Samples vrednost postavimo na manju vrednost tako da se Kay frame promeni otprilike u trenutku kada dostignemo vrhunac skoka i trenutno ćemo je postaviti na vrednost 2.

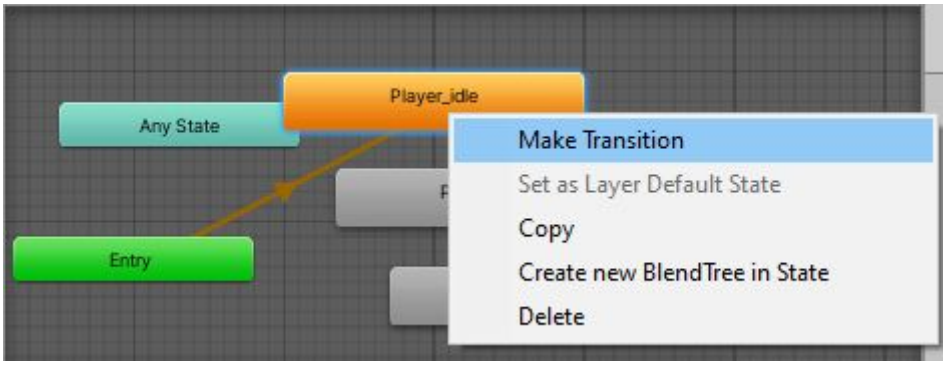
Primetićemo da se u Animations direktorijumu pored animacija koje smo napravili (idle, run, crouch, jump) postoji još jedan fajl koji se zove samo Player i predstavlja Animation Controller jer kontroliše koje će se animacije prikazivati u bilo kom trenutku i kada pokušamo

da otvorimo ovaj fajl otvoriće ga u Animator prozoru gde možemo videti da su prikazani svi Animation klipovi koje smo napravili.

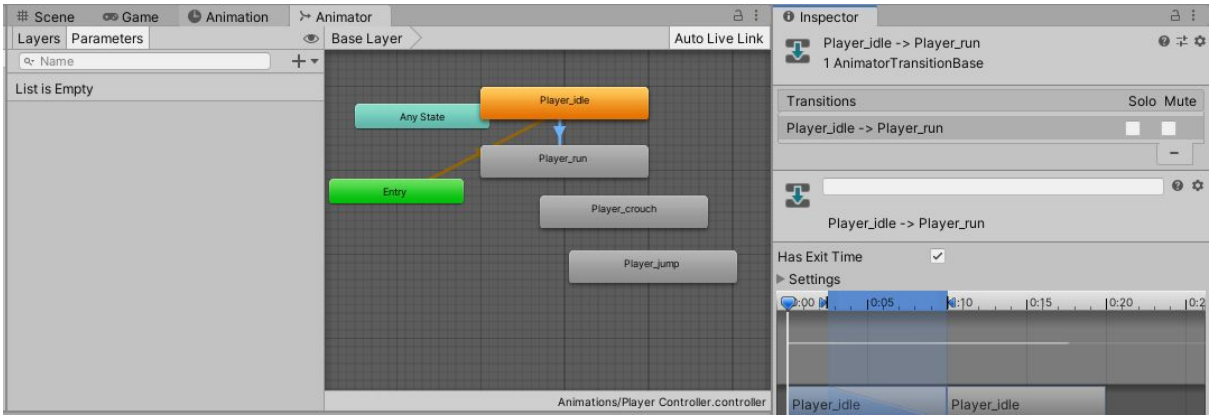


Odmah možemo primetiti da je Player_idle animacija obojena narandžastom bojom i da je strelicom povezana sa Entry objektom koji je zelene boje. Ovo znači da je Player_idle naša default animacija koja će biti prikazana korisniku odmah čim se pokrene igrica. Ako bi sada pokrenuli igricu videli bi da traje Idle animacija iako igrač može da trči i skače i dalje će biti prikazana idle animaicija. Sada možemo kreirati tranzicije od naše idle animacije do drugih klipova.

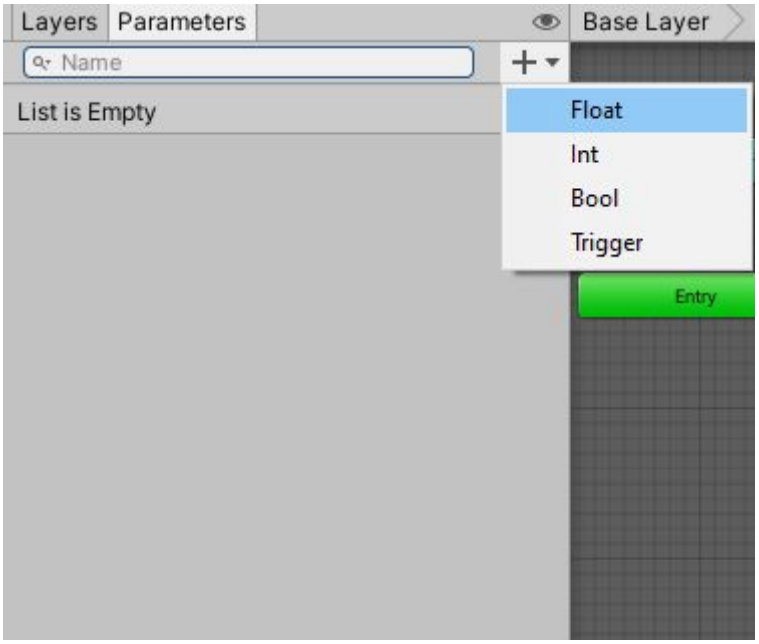
Kada god igrač počne da trči želimo da animaciju proemnimo iz idle animacije u run animaciju. To možemo uraditi preko nečega što se zove Transition. Ako pritisnemo desni klik na našu idle animaciju unutar Animator-a možemo izabrati opciju Make Transition .



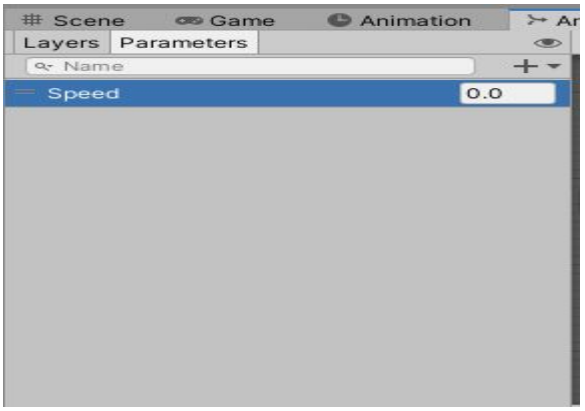
Dobićemo strelicu koja ide iz idle animacije i koju možemo povezati sa bilo kojom drugom animaciom. U ovom slučaju povezaćemo je sa run animacijom.



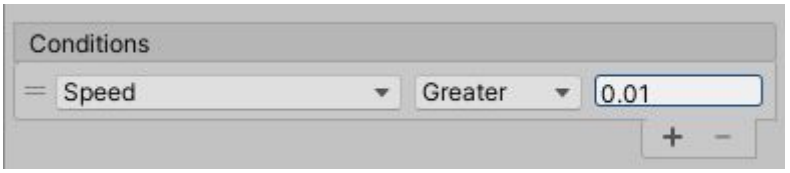
Ovako smo kreirali tranziciju između idle i run animacije i ako je izaberemo možemo menjati kako i kada će se da tranzicija dogoditi.Prvo što moramo postaviti je uslov jer ne zelimo da se ova tranzicija odmah dogodi, već da se dogodi kada igrač miruje i onda počne da trči. Trenutno naš Animator ne za da li se igrač uopšte kreće. Da bismo to omogućili koristićemo parametre. Sa leve strane možemo videti dva tab-a: Layers i Parameters. U Parametars tab-u klikom na znak plus možemo kreirati novi parametar.



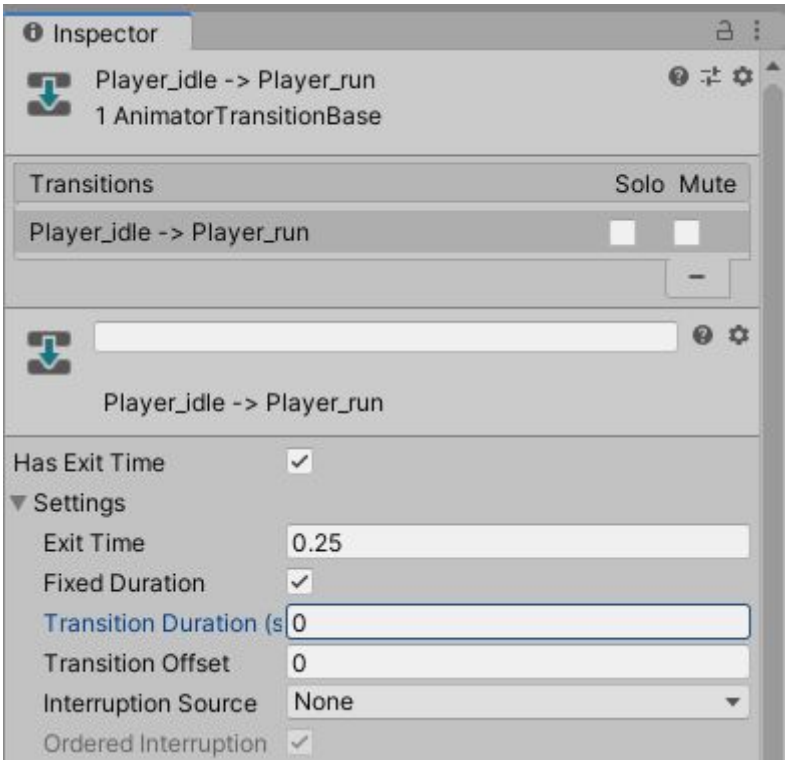
Prametar će biti tipa float, nazvaćemo ga Speed i postavićemo njegovu default vrednost na 0.



Sada možemo postaviti uslov u Conditon delu naše tranzicije. Napravićemo novi uslov klikom na znak plus. Uslov će biti da brzina igrača bude veća od neke male vrednosti, npr.0.01.



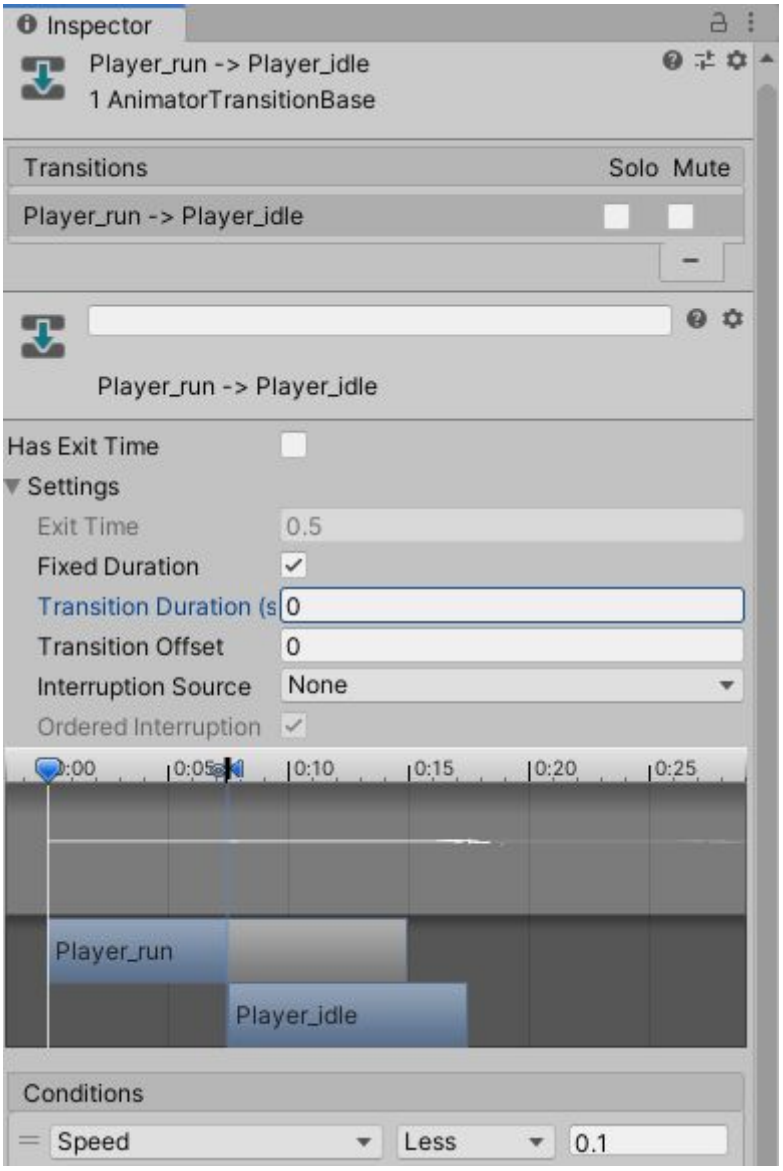
Drugim rečima, ako počnemo da se krećemo, Animator će početi da prikazuje animaciju trčanja. Moramo se još pobrinuti i da tranzicija između animacija bude trenutna, odnosno da nema čekanja između promene animacije već da se promena desi odmah. To možemo uraditi ako što ćemo Has Exit Time atribut postaviti na false i u Settings delu ćemo Transition Duration postaviti na 0.



Takođe želimo i da se vratimo nazad iz run animacije u idle animaciju kada igrač prestane da trči. U ovom trenutku kada igrač počne da trči animacija će biti promenjena iz idle u run animaciju ali kada igrač stane i dalje će biti prikazana run animacija.

Za ovo će nam biti potrebna jos jedna tranzicija koja ovaj put ide od run animacije do idle animacije. U ovo tranziciji dodaćemo novi uslov, i to da tranzicija nastane samo kada je

Speed parametar manji od 0.01, i naravno Has Exit Time i Transition Duration postavljamo isto kao u prethodnom slučaju.



Jedino što još ostaje je da Speed parametar menjamo koristeći našu skriptu. Potrebno je otvoriti PlayerMovement skriptu i prvo što želimo da uradimo je da unutar ove skripte imamo pristup Animatoru. Srećom kada smo počeli da kreiramo animacije za igrača Unity je sam napravio Animator komponentu i pridružio je igraču. Animator komponentu nećemo menjati ali možemo primetiti da komponenta referencira Animator Controller.

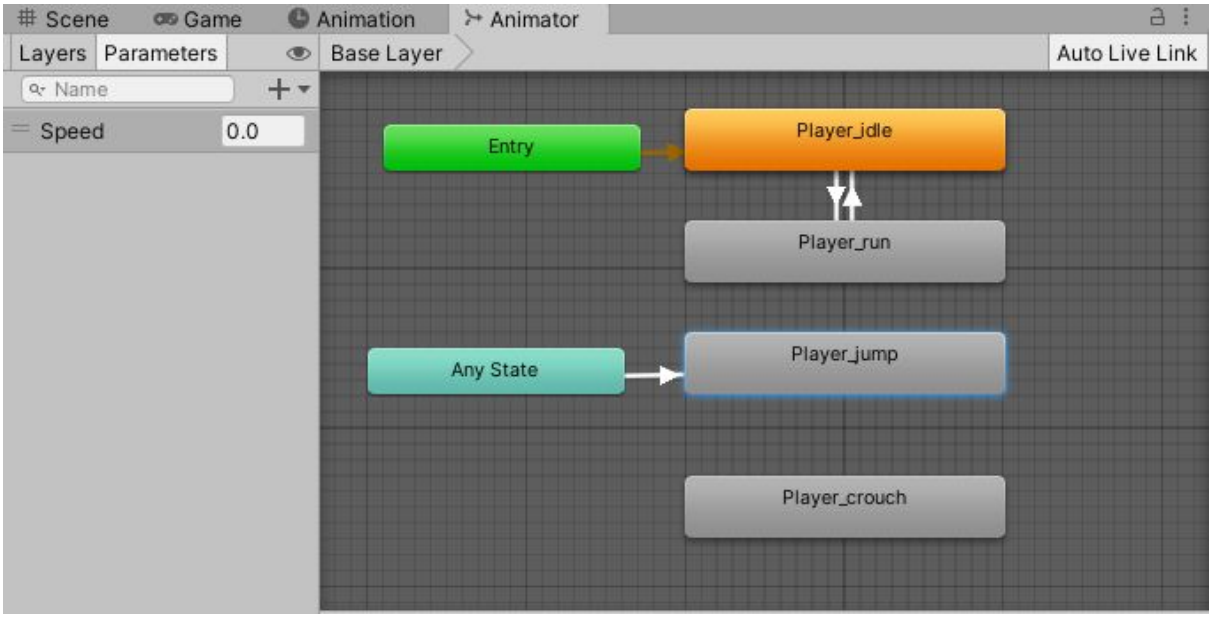
U PlayerMovement skripti ćemo napraviti referencu na naš Animator Controller tako što ćemo napraviti Animator promenljivu koju ćemo nazvati animator i koja ima vidljivost public.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerMovement : MonoBehaviour
6  {
7
8      public CharacterController2D controller;
9
10     public Animator animator;
```

Sada kada otvorimo Unity i kliknemo na Player objekat, u Inspector delu u PlayerMovement komponenti možemo videti atribut Animator sa praznim slotom iu taj prazan slot je sada potrebno prevući Animator komponentu. Sada kada god u našoj skripti napišemo animator, to će biti referenca na Animator Controller igrača.
Sada unutar PlayerMovement skipte možemo napisati animator.SetFloat što će pozvati metodu koja će postaviti vrednost za parametar čije joj ime prosledimo. Pošto želimo da promenimo vrednost Speed parametra, poziv metode će izgledati ovako:

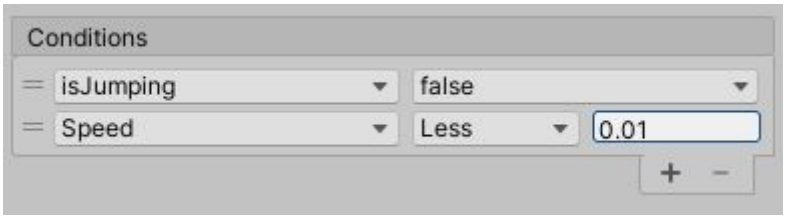
```
18      void Update()
19      {
20          horizontalMove = Input.GetAxisRaw("Horizontal")*runSpeed;
21
22          animator.SetFloat("Speed",Matf.Abs(horizontalMove));
```

Primetićemo da SetFloat metodi prosleđujemo horizonalMove koji predstavlja vrednost za koju se treba pomeriti igrač ali pošto horizontalMove ima negativnu vrednost kada se igrač kreće u levo potrebno je proslediti njegovu apsolutnu vrednost.
Ako sada pokrenemo aplikaciju čim igrač počne da se kreće on prelazi iz idle u run animaciju a kada stane opet se vrati u idle animaciju.
Dalje želimo da imamo mogućnost da uđemo u jump animaciju iz bilo koje druge animacije, nebitno da li igrač samo stoji u mestu,trči ili čuči. To možemo postići tako što ćemo napraviti tranziciju iz bilo koje animacije u jump animaciju ali sa velikim brojem animacija na raspolaganju ovo bi vrlo brzo postalo veoma zamorno tako da je bolje za to koristiti Any State blok koji nam dozvoljava da promenimo animaciju iz bilo koje baš u jump animaciju. Napravićemo sada tranziciju iz Any State u Player_jump.



Sada je potrebno postaviti uslove pod kojima će se ova tranzicija nastati. Za ovu tranziciju biće nam potreban novi parametar koji će biti tipa bool i nazvaćemo ga isJumping i postaviti podrazumevanu vrednost na false.
Sada možemo kreirati novi uslov u Condition delu pod kojim će nastati ova tranzicija. Uslo ćemo postaviti tako da isJumping parametar mora imati vrednost true.

Naravno potrebno je i kreirati tranziciju koja će iz jump animacije ući u idle animaciju ili u run animaciju u zavisnosti od toga da li se igrač u tom trenutku kreće. Prvo kreiramo tranziciju iz jump animacije u idle animaciju. Uslov za tranziciju će biti da je isJumping parametar false i da Speed parametar ima vrednost manju od 0.01.



Zatim dodajemo tranziciju iz jump animacije u run animaciju na sličan način kao ranije. Prvo dodajemo tranziciju i postavljamo uslov ta tranziciju samo kada isJumping parametar ima vrednost false i kada Speed parametarima vrednost veću od 0.01.

Naravno, za sve ove tranzicije potrebno je postaviti atribute HasExitTime na false i Transition Duration na 0.

Ostaje još samo da isJumping parametar kontrolišemo koristeći PlayerMovement skriptu što ćemo uraditi na isti način kao za Speed parametar. U PlayerMovement u delu koji registruje da je igrač skočio pozvaćemo animator.SetBool metodu kojoj ćemo proslediti da za isJumping parametar vrednost postavi na true odnosno false.