



**23. DANI OTVORENIH RAČUNARSKIH SUSTAVA
CROATIAN LINUX USERS' CONFERENCE 2016**

**Kako napraviti Space Invaders pomoću
Unity2D-a i C#-a**

Nino Nikola Stanušić

FER / KSET

Fakultet Elektrotehnike i Računarstva / Klub Studenata Elektrotehnike

Sadržaj

- 1. Uvod**
- 2. Potrebni materijali i programi**
- 3. Koncept Space Invaders-a**
- 4. Opis Unity sučelja**
- 5. Import paketa**
- 6. Generiranje objekata u scenu**
- 7. Skripta - Brod, gibanje**
- 8. Skripta - MetakPonašanje**
- 9. Skripta - Brod, dodavanje metka i kolizije**
- 10. Skripta za invadere - Pojedinačno ponašanje**
- 11. Skripta za invadere - Grupno ponašanje**
- 12. Rješavanje nepoželjnih kolizija**
- 13. Animacija**
- 14. Zaključak**

1. Uvod

Ovaj dokument je pisan radi pokazivanja osnova korištenja programa Unity prilikom izrade 2D igara te osnove pisanja skripta u programskom jeziku C#. Ovaj dokument je primarno namijenjen profesorima osnovnih i srednjih škola koji su bili na radionici izrade Space Invadersa na DORS/CLUC konferenciji u Zagrebu 2016. godine na FER-u.

Namjera i način pisanja je detaljan i napravljen tako da svatko sa malo znanja u programiranju može pratiti izradu korak po korak. Svaki korak je opisan te svaka linija koda objašnjena. Zbog toga preporučujem da se ovaj dokument čita dok se sa strane dio po dio rješava.

Zahvaljujem se organizatorima konferencije i svima koji su došli na radionicu.

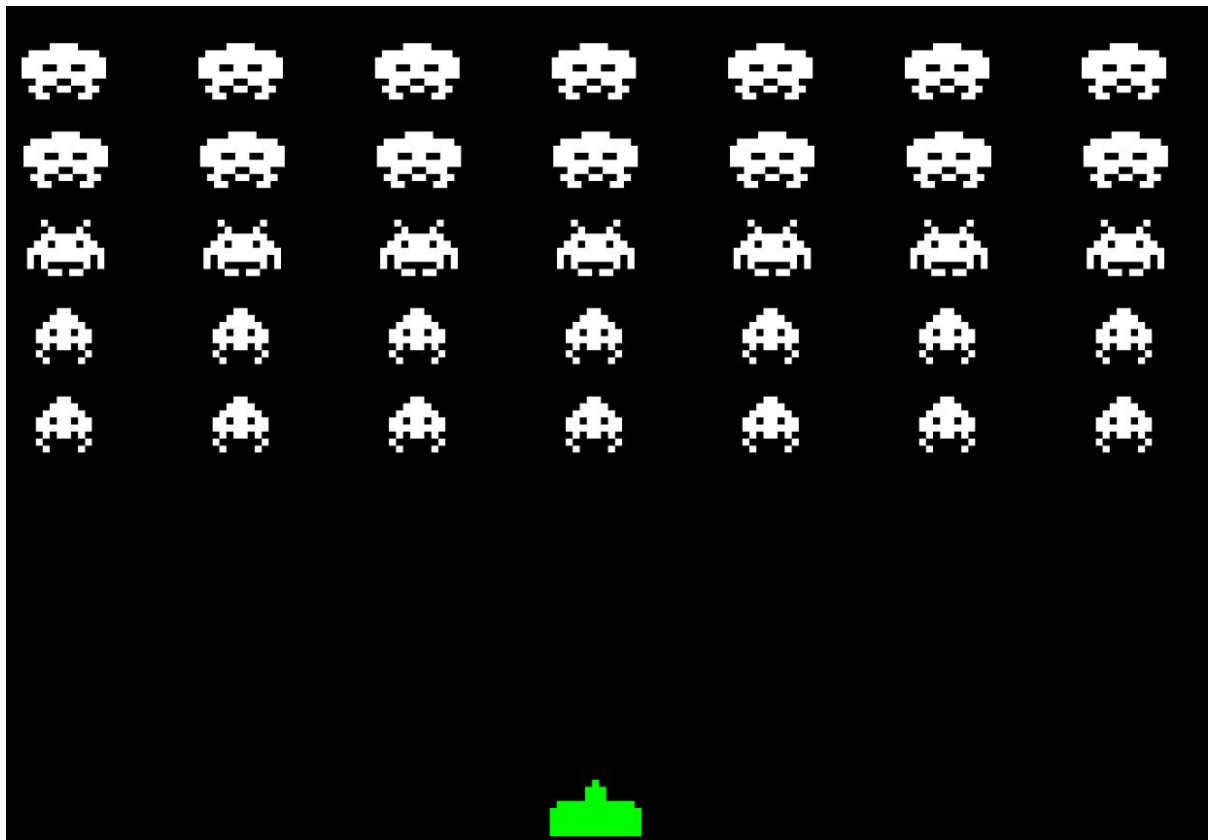
2. Potrebni materijali i programi

Potrebno je na računalu instalirati Unity minimalnu verziju 5.3.4 koja uz sebe dohvati Visual Studio 2015 za pisanje skripta u C#-u.

Pomoćni materijali nalaze se u github repozitoriju na adresi <http://github.com/nnstanusic/SpaceInvaders>

U repozitoriju se nalaze dva paketa jedan osnovni paket sa kojim krećemo raditi i drugi paket u kojem je spremljeno gotovo riješenje. Skinuti se može preko gumba "Download ZIP"

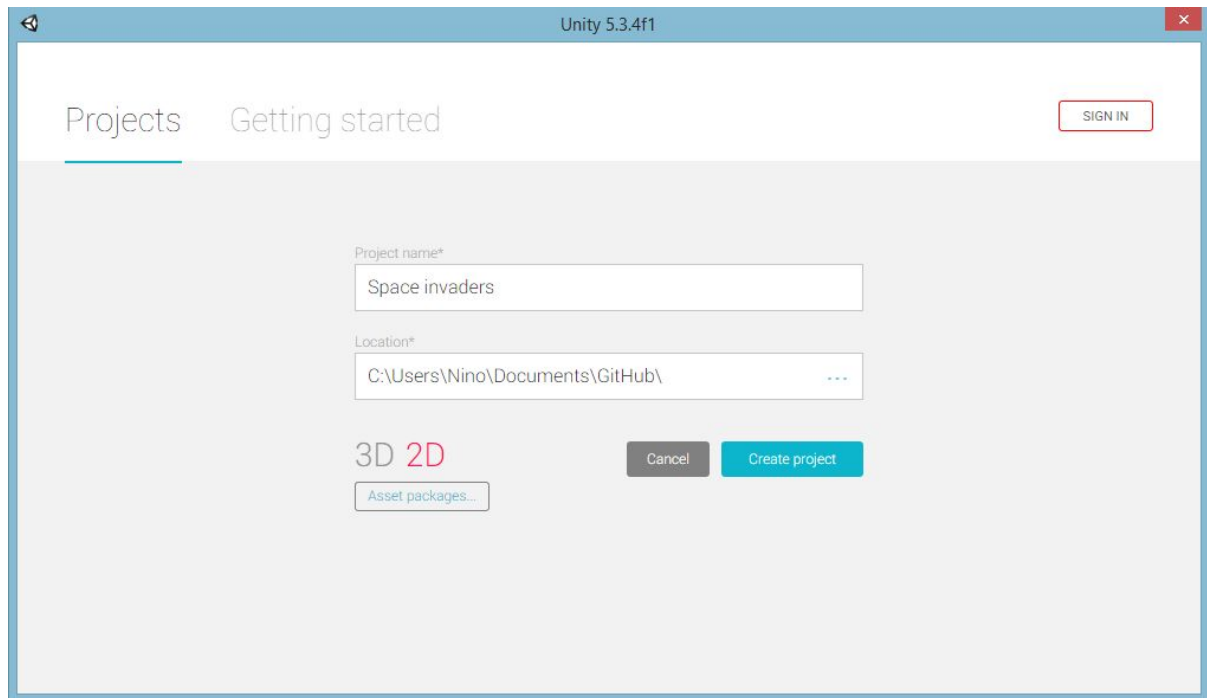
3. Koncept Space Invadersa



Invaderi polako se kreću prema desno, kad dođu do kraja ekrana naprave mali skok prema našem brodu i gibaju se prema lijevo do kraja ekrana i ponovno se spuštaju. Kroz cijelo to vrijeme oni ispaljuju metke prema nama koje moramo izbjegavati. Cilj je uništiti sve *invadere*.

4. Opis Unity sučelja

Prilikom pokretanja Unity-a susrećemo se sa početnim prozorom koji od nas traži nastavak na već postojećem projektu ili pokretanje novog projekta.

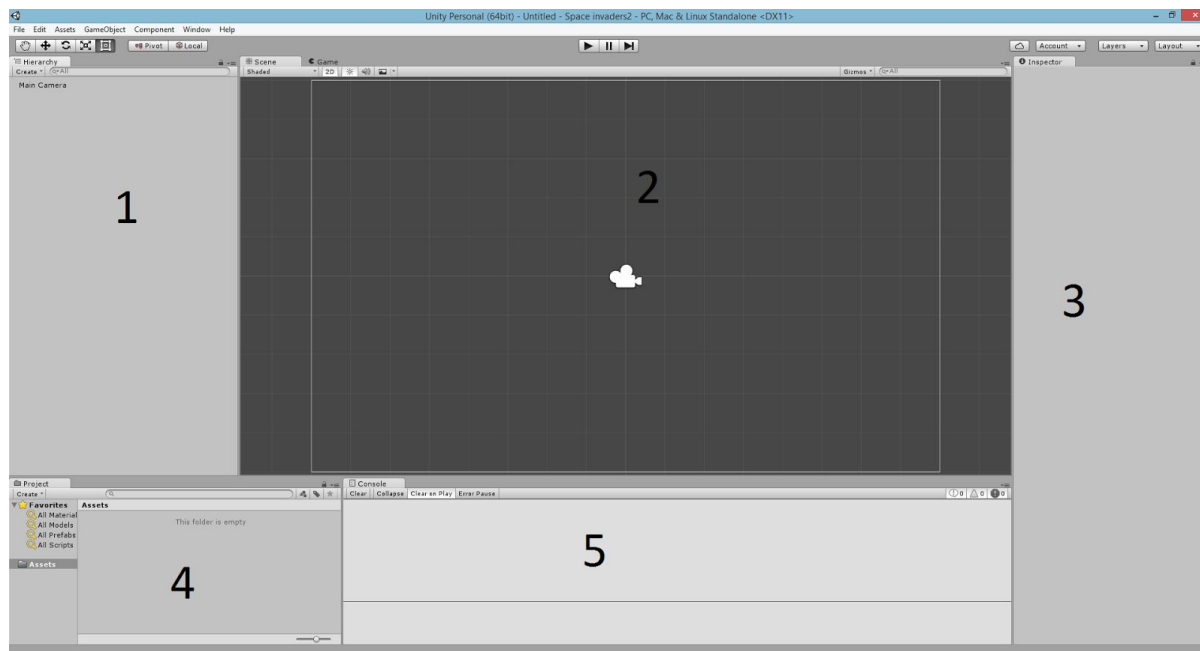


Zadamo ime projektu te lokaciju na kojoj će se nalaziti.

Obavezno stisnuti gumb da radimo 2D igru.

Za kraj stisnemo gumb *Create project* i krećemo sa izradom naše igre.

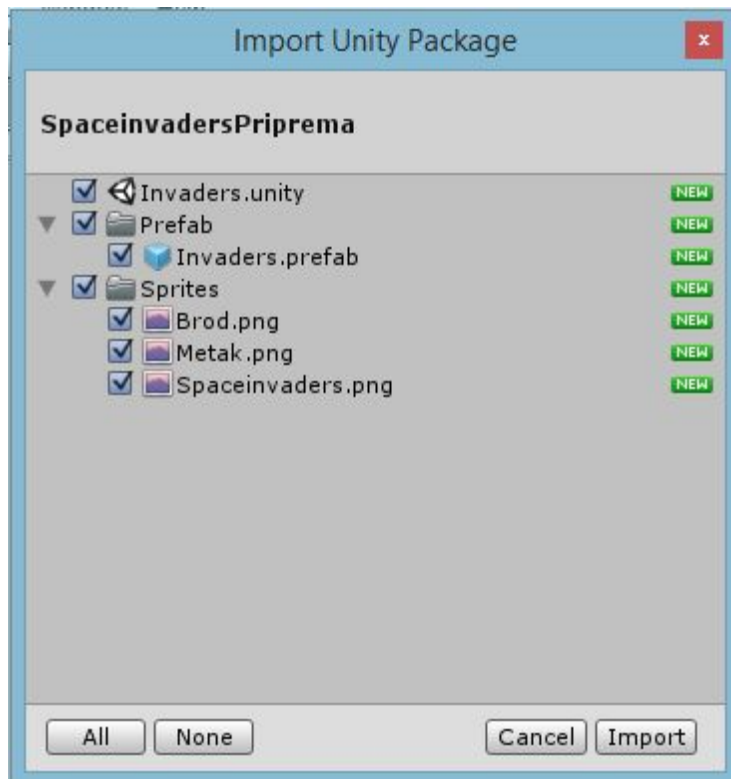
Nakon stvaranja projekta otvorio nam se *unity editor* u kojem je prozor raspodijeljen u 5 panela.



1. *Hierarchy panel* - u ovom panelu imamo listu svih naših *GameObject*-a koji se trenutno nalaze u sceni. *Hierarchy* služi nama radi pregledljivosti i laganog pronalaska naših objekata u sceni. Ovaj panel prilikom prvog otvaranja sadrži samo kameru.
2. *Scena* - u sceni se nalaze svi objekti. Od Prepreka i zidova do glavnog lika, protivnika i projektila. U biti svi dijelovi naše igre.
3. *Inspector* - *Inspector* nam daje detaljne opise stvari na koje stisnemo mišem. *Inspector* nama opisuje svaki *GameObject* i pokazuje komponente kojima možemo mijenjati vrijednost.
4. *Project* - Project panel jednostavno rečeno je kako je naš fizički direktorij raspodijeljen. Gdje se nalazi koja slika, skripta ili zvuk.
5. *Console* - U konzoli se ispisuju sve naše greške prilikom kodiranja te za vrijeme izvođenja igre neke naše pomoćne poruke.

5. Import paketa i opis projekta

Paket koji smo skinuli sa githuba jednostavno mišem povučete u projekt panel i pojavi se ova slika:

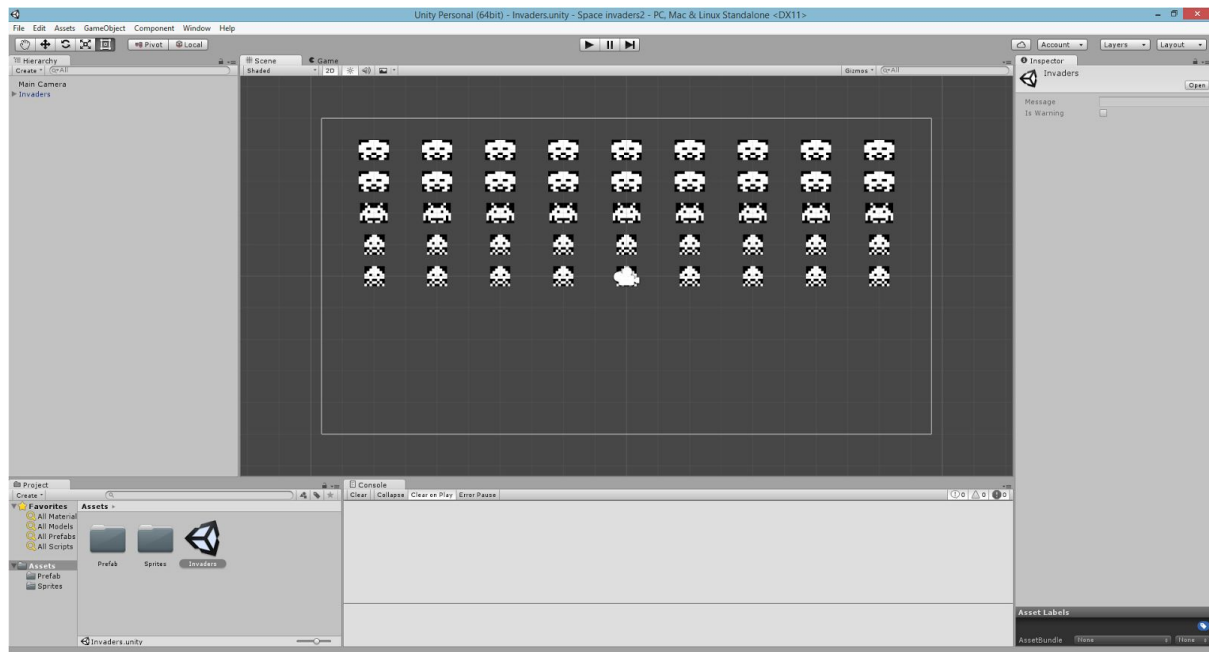


Kao što piše u paketu imamo već spremnu jednu scenu *Invaders.unity*. Direktorij *Prefab* koji sadrži *prefab Invaders.prefab* te direktorij *Sprites* u kojem se nalaze sve slike potrebne za ovu igru.

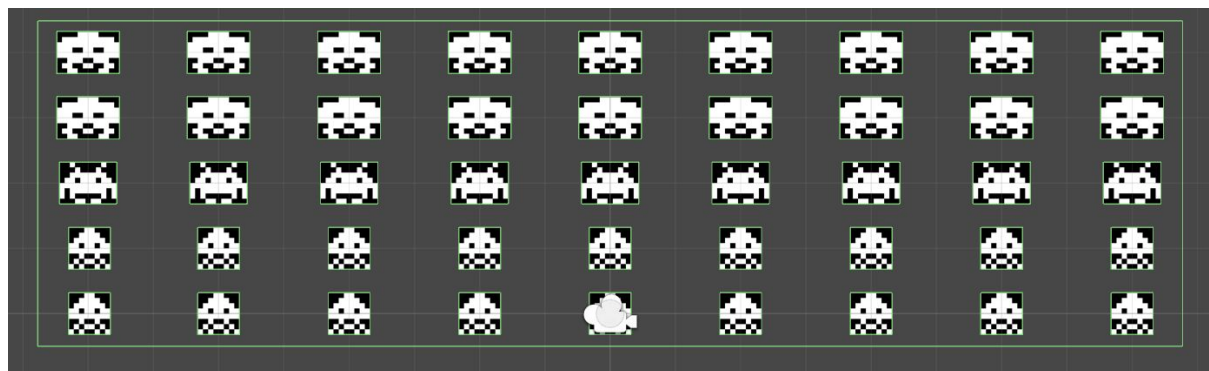
6. Generiranje objekata u sceni

U *project* panelu premjestimo se u *Assets* directorij i otvorimo našu scenu *Invaders.unity* dvostrukim pritiskom na miš.

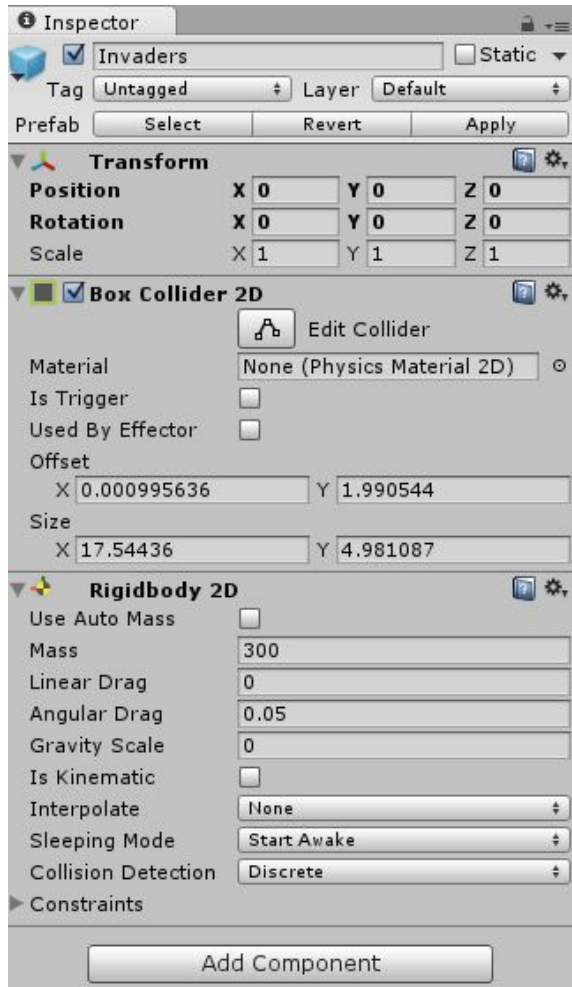
Naš editor bi sada ovako trebao izgledati:



Scena koju smo otvorili je popunjena je raznim invaderima. U *hierarchy* panelu se nalazi jedan novi *GameObject* zvan *invaders*. Klikom na njega oko svakog invadera se stvori jedan zeleni pravokutnik te oko svih njih također jedan pravokutnik.



Ako pogledamo u inspektoru *GameObject* “Invaders” ima 3 komponente:



Transform:

Komponenta koja sadrži koordinate, rotaciju i skalu našeg objekta.

BoxColider2D:

Zeleni pravokutnik kojeg vidimo. Komponenta označava da se objekt može sudarati sa drugim objektima.

Rigidbody2D:

Fizičko tijelo našeg objekta.

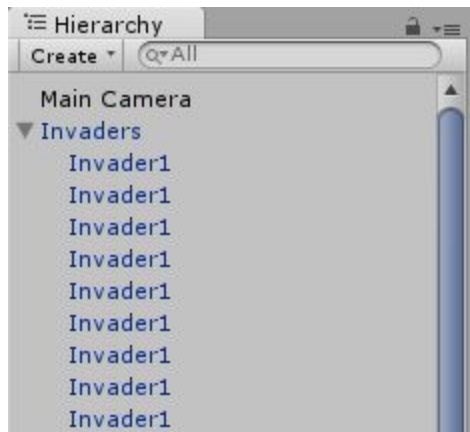
Ova komponenta definira fiziku nad našim objektom.

Koliko je teško?

Kolika gravitacija utječe na njega?

Postoji li trenje za vrijeme gibanja?

U hierarchy panelu uočimo da GameObject Invaders ima lijevo od sebe strelicu. Pritiskom na tu strelicu vidimo sve objekte sadržane unutar sebe.



Invaders je roditelj dok su Invader1-3 njegova djeca.

Invaderi 1 do 3 su također GameObjecti te svaki ima vlastite komponente.


Trenutno su te komponente samo *Transform*, *Sprite Renderer* i *Box Collider 2D*.

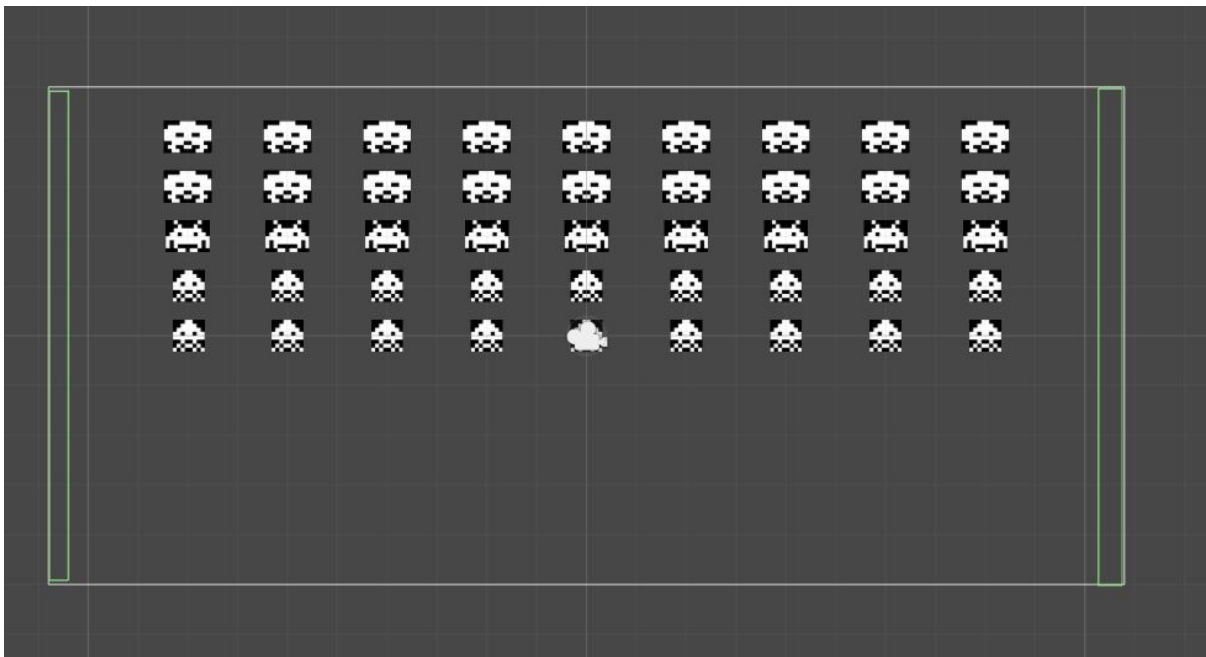
Jedini objekt koji nama još nedostaje u sceni je naš svemirski brod. Njegova slika se nalazi u direktoriju *Sprites*. Povučemo sliku na scenu i stvorimo automatski novi *GameObject* u sceni.

Uočimo da naš brod počinje sa komponentama: *Transform* i *Sprite Renderer*. Dakle koordinate i komponenta za prikaz slike.

Nedostaje nam *Box Collider 2D* i *Rigidbody2D* komponente da bi se naš brod mogao sudarati sa drugim objektima i da ima fizičko tijelo nad kojim možemo vršiti sile i postavljati gibanje.

Ako bi sada pokrenuli igru naš svemirski brod bi sa ekrana gibao se polako prema dole pošto ima svoje tijelo i gravitacija djeluje na njega. Jednostavno u našem brodu na komponenti *Rigidbody2D* i stavimo *Gravity Scale* na vrijednost nula.

Da bi smo spriječili gibanje izvan kamere napraviti ćemo još dva zida. U *Hierarchy* panelu stisnemo desnu tipku miša i koristimo *Create empty*. U hijerarhiji se pojavio novi *GameObject* zvan "GameObject". U *Inspector* panelu mu promijenimo ime u "Zid". Nas objekt za sada ima samo *Transform* komponentu pa ćemo mu dodati dvije *Box Collider 2D* komponente. Manipulaciju veličine pravokutnika možemo točno zadati pišući visinu i širinu te koordinate središta ili koristeći gumb u komponenti . Preko gumba se na zelenom pravokutniku pojave 4 referentne točke kojima možemo mišem povlačiti i time mijenjati veličinu pravokutnika. Naša dva zida bi na kraju trebala otprilike izgledati ovako:



I spremni smo krenuti pisati našu skriptu.

7. Skripta - Brod, Gibanje

U project panelu u *Assets* direktoriju ćemo novi direktorij pod imenom *Scripts* napraviti. Desni klik na prazno i odabiremo *Create -> Folder*. U direktoriju *Scripts* ćemo napraviti novu Skriptu. Stisnemo desnu tipku miša i sada *Create -> C# Script*. Skriptu dodamo na brod isto kao i sve ostale komponente.

Dvostrukim pritiskom otvaramo Visual Studio ili MonoDevelop ovisno o postavkama i krećemo sa pisanjem skripti.

Primarno skriptu izgleda ovako:

```
using UnityEngine;
using System.Collections;

public class Brod : MonoBehaviour
{
    // Use this for initialization
    void Start()
    {

    }

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

    }
}
```

Imamo razred *Brod* koja nasljeđuje razred *MonoBehaviour*.

Monobehaviour sadrži skup praznih metoda koje možemo dopuniti te tako definirati ponašanje našeg broda.

Prva metoda je *Start*. U njoj definiramo stvari koje na samom početku stvaranja objekta trebamo izvršiti.

Druga metoda je *Update* koja se poziva svaki put nakon što se slika osvježi. U updateu rješavamo unos informacije sa tipkovnice i miša te animaciju.

Prvo ćemo definirati početne varijable.

Našem brodu treba *smjer* u kojem će se gibati te njegova *brzina*. Dalje trebamo znati jeli naš brod živ ili *uništen*. I za kraj će nam trebati Metak kojeg ćemo pucati.

```
public GameObject Metak;  
public float Brzina;  
  
private float smjer;  
private bool unisten;  
  
// Use this for initialization  
void Start()
```

Public varijable možemo mijenjati u unity editoru i drugim skriptama dok *private* varijable mijenjamo samo unutar vlastite skripte.

Unosom u tipkovnicu trebamo mijenjati smjer gibanja našeg broda. Rekli smo da unos tipkovnice obavljamo u *Update* metodi.

```
void Update()  
{  
    //U kojem ce se smjeru nas brod gibati?  
    smjer = Input.GetAxis("Horizontal");  
}
```

Input.GetAxis("Horizontal") nama za tipke 'A' i 'D' te ljevu i desnu strelicu vraća vrijednosti od -1 do 1 ovisno o tome koliko dugo koji gumb držimo.

Sad kada smo smjer riješili idemo fiziku rješavati. To se obavlja u *FixedUpdate* metodi.

```
void FixedUpdate()  
{  
    //Horizontalno gibanje broda  
    GetComponent<Rigidbody2D>().AddForce(new Vector2(smjer, 0) * Brzina);  
}
```

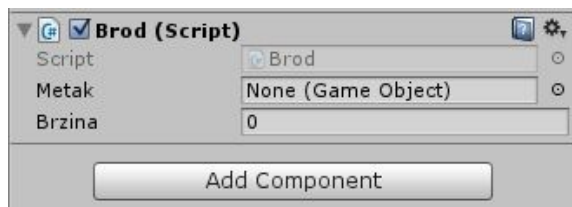
Metoda *GetComponent<Rigidbody2D>()* nama vraća našu komponentu *Rigidbody2D* od broda te nju odmah koristimo i dodamo silu na tijelo.

Sila je usmjerena prema *čemu*? po x osi i ovisi o smjeru. A sa brzinom mijenjamo veličinu vektora. Iz matematike se sjećamo:


$$\text{Vector}(x,y) * a = \text{Vector}(a*x, a*y)$$

FixedUpdate naspram *Update* metode poziva se u stalnim intervalima i time omogućuje konstantno dodavanje sile u realnom vremenu. Česta pojava je da se *Update* pozove više puta za vrijeme jednog izvršavanja *FixedUpdate*-a. Pa ako bi stavili dodavanje sile u *Update*-e naše tijelo bi dobilo ogromno ubrzanje.

Ako se vratimo u Unity Editor i pritisnemo brod da ga u *Inspector panelu* pogledamo uočimo da skripta ima nova polja.



To su varijable koje smo definirali kao *public*.

Pa podesimo brzu na 70¹ i pokrenemo igru gore preko play gumba  gore u sredini editora.

Uočimo da kada se krene gibati ne staje. Što znači da na naše tijelo ne djeluje nikakvo trenje. To promijenimo u *Rigidbody2D* komponenti. Varijablu *Linear Drag* postavimo vrijednost na 10¹.

¹Odobrane su visoki brojevi radi sto znaci veće sile djeluju na tijela. Što za uzrok ima da tijelo se brže ubrzava i usporava, čime dobijemo tečnije gibanje tj. nema osjećaja tromosti.

8. Skripta - MetakPonašanje

Krecemo istim postupkom kao i sa brodom. Uzmemo sliku metka iz *Sprites* direktorija i stavimo u scenu.

Opet jedine komponente koje naš metak trenutno ima su *Transform* i *Sprite Renderer*.

Naš metak se mora moći sudariti s nečim i mora se gibati u jednom smjeru. Što znači da mu dodajemo komponentu za koliziju *Box Collider 2D* i komponentu za gibanje i svu fiziku *Rigidbody2D*. Te napravimo novu skriptu *MetakPonasanje* u *Scripts* direktoriju.

Prvo zadamo varijable kao u Brod skripti. Imamo *brzinu* i *smjer*. Smjer jer ćemo odrediti ide li metak prema gore ili dolje.

```
public float brzina;  
public int smjer;
```

U *Update* metodi vršimo provjeru da kad metak izađe iz kamere da ga uništimo. Pošto imamo fiksnu kameru. Kamera ide od -6 do 6 po vertikali (odmjereno otprilike).

```
void Update()  
{  
    //Ako je Metak izvan ekrana unistiti metak, odokativne koordinate  
    if (transform.position.y > 6 || transform.position.y < -6)  
    {  
        Destroy(this.gameObject);  
    }  
}
```

Bez ove provjere bi naš metak beskonačno prema gore ili dolje putovao. Gibanje je metka je konstantno od početka stvaranja metka do njegovog uništenja. Pa to u *FixedUpdate*-u izvodimo na ovaj način:


```
void FixedUpdate()
{
    //Postavit brzinu metka na konstantan broj
    GetComponent<Rigidbody2D>().velocity = new Vector2(0, smjer * brzina);
}
```

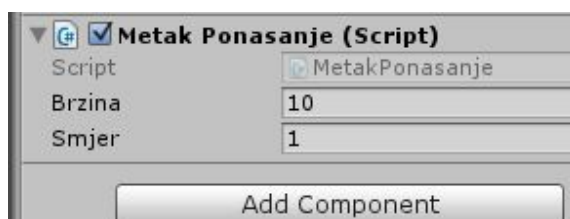
Direktno postavljamo brzinu tijela na konstantan vektor, pošto smo u 2D okruženju koristimo *Vector2*.

Za kraj metak sam se mora uništiti prilikom sudara. Pozivamo novu metodu iz nasljeđenog *MonoBehaviour-a*, *OnCollisionEnter2D*:

```
public void OnCollisionEnter2D(Collision2D collision)
{
    //Ako se sudari sa necime unistiti metak
    Destroy(this.gameObject);
}
```

OnCollisionEnter2D kao što ime kaže se poziva svaki put kada se ulazi u sudar.

Povratkom u *Unity Editor* u *Inspector* panelu pogledajmo naš gotovi metak. I definiramo *public* varijable.



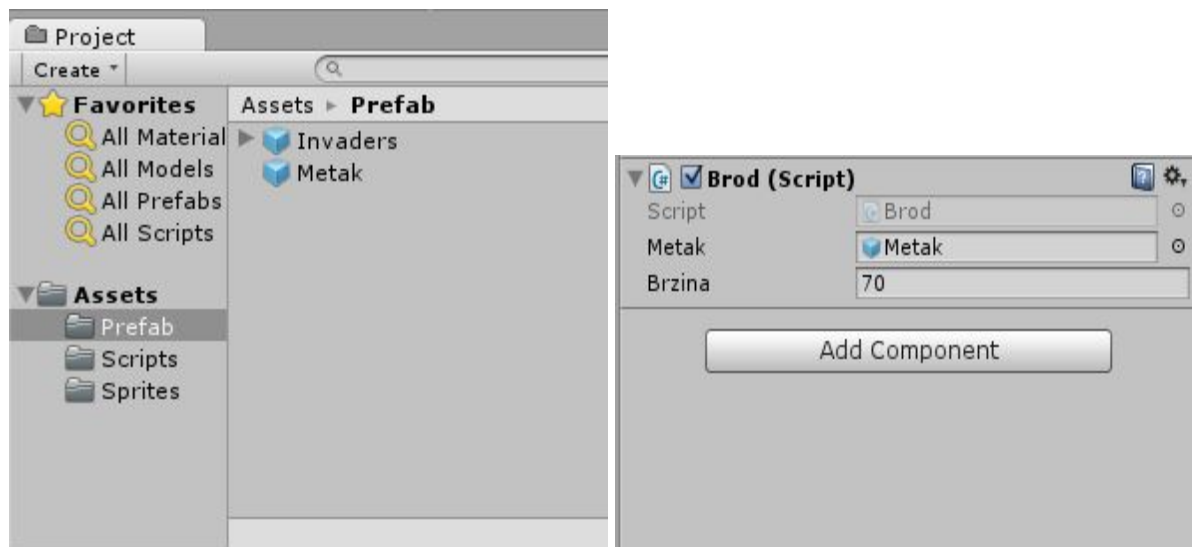
Pokretanjem igre naš metak bi se trebao gibati prema gore i prilikom sudara sa invaderima nestati.

9. Skripta - Brod, dodavanje metka i kolizije

Iako je naš metak gotov ne možemo ga držati u sceni na nasumičnom mjestu i da prilikom pokretanja izvodi stvari. Želimo da se pojavi kasnije u igri kada mi stisnemo tipku za pucanje.

Pa idemo spremiti *Prefab* (Predložak). Naš metak bi se trebao nalaziti u *Hierarchy* panelu pod imenom "Metak".

Povlačimo *Metak* iz *Hierarchy* panela u *Project* panel, točnije u *Prefab* direktorij. Sad imamo Predložak spreman za korištenje u drugim skriptama.



Dodamo prefab metka u našu *Brod* komponentu na brodu. Prefabi su tipa *GameObject*.

Vraćamo se u skriptu da dodamo kod za pucanje i stvaranje metka. U *Update* metodi rješavamo unos podataka sa tipkovnice pa ćemo dodati da izvodi pucanje za tipku 'Space'.

```
// Update is called once per frame
void Update()
{
    //U kojem ce se smjeru nas brod gibati?
    smjer = Input.GetAxis("Horizontal");

    //Ako je stisnuta tipka Space i nepostoji Metak na igralistu i brod nije unisten
    if (Input.GetKeyDown(KeyCode.Space) && GameObject.Find("Metak(Clone)") == null && !unisten)
        pucaj();
}
```

Objekt *Input* uz već gotova rješenja za implementaciju unosa preko *GetAxis* metoda sadrži i druge metode kojima možemo riješavati stvari. Tri dodatne metode su *GetKey*, *GetKeyDown* i *GetKeyUp*.

GetKey - vraća istinito dok je tipka stisnuta

GetKeyDown - vraća istinito u prvom trenutku kada je stisnuta tipka

GetKeyUp - vraća istinito kada pustimo tipku

Prvim uvjetom pitamo jeli se u ovom frame-u(okvir) stisnuo space.

Drugim uvjetom pitamo postoji li već metak imena "Metak(Clone)" jer ako postoji ne smijemo moći pucati. S time sprečavamo stvaranje ogromne količine metaka konstantnim stiskanjem space gumba.

Trećim uvjetom sprečavamo da pucamo kada smo pogođeni metkom.

Ako su sva tri uvjeta istinita koristimo metodu *pucaj*.

```
//stvari metak ispred sebe
void pucaj()
{
    Instantiate(Metak, transform.position + Vector3.up, Quaternion.identity);
}
```

Naredba za stvaranje objekta je *Instantiate*.

Argumenti su tipa *GameObject* koji želimo stvoriti, zatim *Vector3* koordinate gdje ga stvorimo i argument tipa *Quaternion*. *Quaternion* se koriste za reprezentaciju rotacije *GameObject-a*. Uzmite *Quaternion.identity* zdravo za gotovo za sada. Za ovaj tutorial nećemo ništa raditi sa rotacijom.

Za kraj nam ostaje riješiti koliziju našeg broda. Opet pozivamo metodu *OnCollisionEnter2D*:

```

public void OnCollisionEnter2D(Collision2D collision)
{
    //Ako se sudari sa bilocime osim Zida unisti brod
    if (collision.collider.name != "Zid" && !unisten)
    {
        Invoke("resetirajIgru", 5);
        Destroy(GetComponent());
        unisten = true;
    }
}

```

Prilikom sudara jedino sa čime se smije naš brod sudariti je Zid koji sprečava izlazak iz vidnog polja kamere.

Ako se sudarimo sa tuđim metkom ili invaderom događa se slijedeće: prvo pozivamo metodu *Invoke* koja nakon 5 sekundi poziva metodu *resetirajIgru* koju ćemo kasnije napisati.

Umjesto da uništavamo cijeli objekt ovdje samo uništavamo komponentu broda koja je zaslužna za crtanje slike. Brod nestaje sa scene gledajući sliku. Ali u pozadini je još uvijek tu. Ako bi cijeli *GameObject* uništili, metoda *resetirajIgru* koja bi se trebala nakon 5 sekundi pokrenuti nebi se izvršila.

I na kraju postavljamo zastavicu da je naš brod uništen.

Resetiranje igre prikazano je u ovom isječku koda:

```

//Resetiraj igru, Poziva se u slucaju da brod umre
void resetirajIgru()
{
    Application.LoadLevel(Application.loadedLevel);
}

```

Ispočetka pozivamo istu scenu.

10. Skripta za invadere - Pojedinačno ponašanje

Kreirajmo novu skriptu *InvaderPonašanje*.

U *Hierarchy* panelu otvorimo invaders *GameObject* i označimo sve Invadere. I svima odjedanput u inspektoru dodajemo komponentu *InvaderPonašanje*.

Jedino što invaderi rade je ako su pogođeni da sami sebe unište. S time da jedino se ne unište ako se sudare sa brodom.

```
void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.collider.name != "Brod")
        Destroy(this.gameObject);
}
```

11. Skripta za invadere - Grupno ponašanje

Roditelj svih invadera, *GameObject* "invaders", obavlja gibanje i pucanje. Za koliko se roditelj pomiče u prostoru toliko se i sva njegova djeca također pomiču.

Napravimo novu skriptu *GrupnoPonasanje*.

Prvo definirajmo varijable unutar razreda. Imamo opet metak kojeg treba ispucati. Brzinu koju zadajemo i smjer u kojem se gibamo.

```
public GameObject Metak;  
public float Brzina;  
private int smjer = 1;
```

Primijetimo da ovaj put smjer je definiran kao *private* što znači da ga samo unutar skripte možemo mijenjati. Ideja je da se svi kolektivno gibaju prema zidu. Kad se sudare sa zidom okrenemo smjer i spustimo invadere prema dolje za jedan mali skok. Zato imamo jedan veliki *BoxCollider2D* oko svih invadera u roditelju.

Fiziku gibanja opet riješavamo u fixed update-u gdje direktno postavljamo brzine.

```
void FixedUpdate()  
{  
    //Postavi brzinu invadera  
    GetComponent<Rigidbody2D>().velocity = new Vector2(smjer * Brzina, 0);  
}
```

Ostaje nam da mijenjamo smjer naših invadera i spuštamo ih postepeno.

```

public void OnCollisionEnter2D(Collision2D collision)
{
    //Ako se sudari sa zidom mjenjaj smjer i spusti Invadere
    if (collision.collider.name == "Zid")
    {
        smjer *= -1;
        transform.position += Vector3.down * 0.2f;
    }
}

```

Tijekom kolizije ako se objekt *Invaders* sudari isključivo sa *Zid* objektom promijenimo smjer i spustimo za mali korak.

Za stvaranje metaka riješavali smo u updateu.

```

// Update is called once per frame
void Update()
{
    //U 2% slucaja obavi pucanje metka
    if (Random.Range(0, 100) < 2)
    {
        //Saznaj koordinate nasumicno odabranog Invadera
        Transform invader = transform.GetChild(Random.Range(0, transform.childCount));

        //Stvori metak na toj poziciji
        Instantiate(Metak, invader.position, Quaternion.identity);
    }

    //Ako smo ubili sve Invadere resetiraj igru
    if (transform.childCount == 0)
    {
        Application.LoadLevel(Application.loadedLevel);
    }
}

```

U prvom if-u tražimo broj od 0 do 100 i želimo da bude vjerojatnost 2% da uspije. Pošto pišemo u update-u koji može varirati od 20 do preko 150 puta u sekundi, kod unutra će se rjeđe pozivati. Kod unutra pronalazi *transform* komponentu nasumičnog dijeteta i na toj lokaciji stvaramo metak.

I za kraj ispitujemo uvjet dali smo sve invadere uništili i ako je to istinito vratimo igru u početno stanje.

Za metak kojeg koriste invaderi stvorimo novi prefab istim postupkom kao za brod, samo što u *Inspector* panelu označimo smjer da je -1. Novi prefab ćemo nazvati "InvaderMetak".

12. Rješavanje nepoželjnih kolizija

Ako bi sada pokrenuli igru svi naši metci bi se sudarali u veliki *Box Collider 2D* koji okružuje sve invadere. Isto tako svi metci koje invaderi ispaljuju se odmah sudare sa invaderima pa se oni zapravo čine kao samoubojstvo.

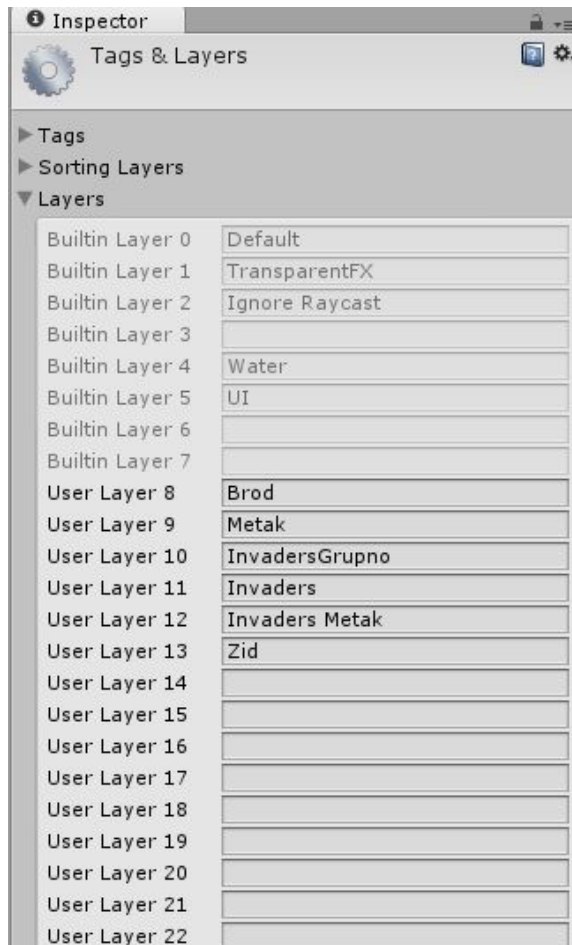
Trebamo nekako filtrirati što se sa čime može sudarati. To radimo preko *Layer-a*.

Idemo prvo dodati nekoliko *Layer-a*. Gledamo što se čime može sve sudariti. Imamo *layer-e* za brod, za metke, metke od invadera, invadere, veliki *Box Collider 2D* u invaders objektu i naš zid.

Gore u desnom kutu iznad *Inspector* panela imamo gumb layers i u njemu edit layers gumb.



Pritiskom na edit layers daje nam opciju dodavanja novih layera. Pa idemo sve gore navedene layer-e dodati.



Ostaje nam još svaki objekt u svoj layer svrstati. Idemo kroz Hierarchy panel i svaki objekt stavimo u svoj layer.

U inspektoru za svaki objekt imamo skroz gore iznad transformacije da možemo staviti koji layer želimo da naš objekt bude.

I za kraj nam ostaje filter napraviti. Filter se nalazi pod *Edit->Project Settings-> Physics2D*

Na dnu u inspektoru imamo jednu matricu u kojoj možemo označavati što sa čime se smije sudarati. U gore navedenom slučaju sve dozvoljene kolizije su:

	Zid	Invaders Metak	Invaders	InvadersGrupno	Metak	Brod	UI	Water	Ignore Raycast	TransparentFX	Default
Default	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
TransparentFX	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Ignore Raycast	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Water	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
UI	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Brod	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Metak	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
InvadersGrupno	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Invaders	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Invaders Metak	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Zid	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I igra je skoro gotva. Ostaje još animacija i gotovi smo.

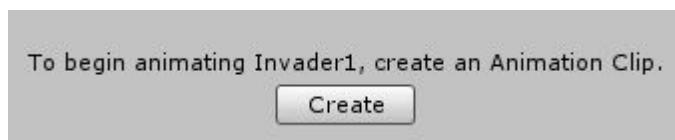
13. Animacija

Za space invaderse imamo dosta jednostavnu animaciju. Za pocetak dodajemo svim invaderima od 1-3 *Animator* komponentu.

Sad otvaramo novi panel *Animator* i *Animation*. Nalaze se pod *Window* gore na alatnoj traci.

Dok smo u *Animation* panelu oznacimo jednog Invadera1 u Hierarchy panelu.

U *Animation* panelu nam onda nudi opciju kreiranja novog *Animation Clipa*.

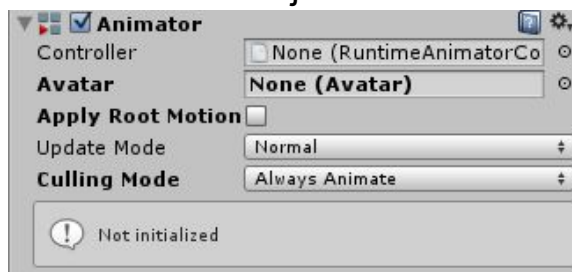


Stisnemo Create i uočimo što se sve promijenilo.

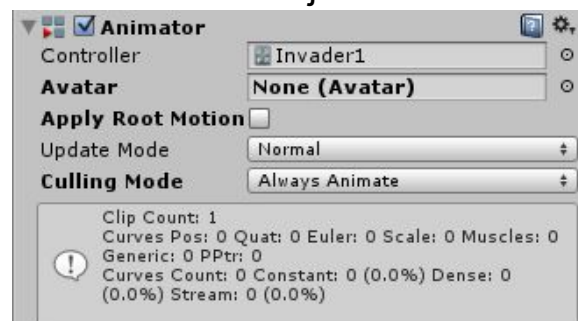
Prvo trebamo clip negdje spremiti. Pa napravimo direktorij *Animation* u *Assets* direktoriju. I unutra spremimo prvu animaciju pod imenom *Invaders1.anim*.

U komponenti *Animator* od *Invader1* se stvorio novi *Controller Invader1*.

Prije:



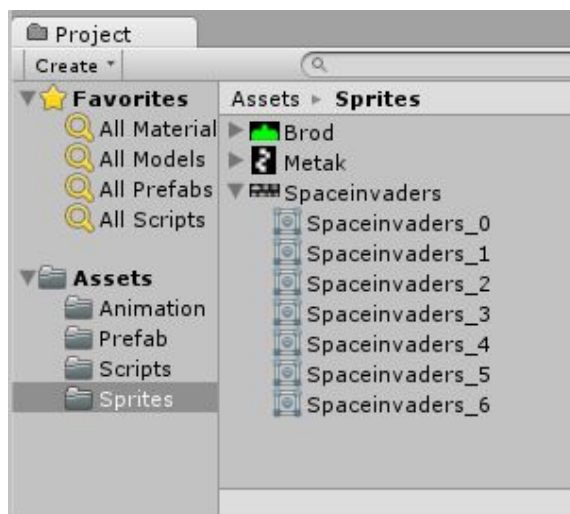
Poslje:



Isto tako *Animation* panel se odjednom promijenio.

Imamo vremensku traku na desnoj strani a na lijevoj strani neki gumb *Add Property*.

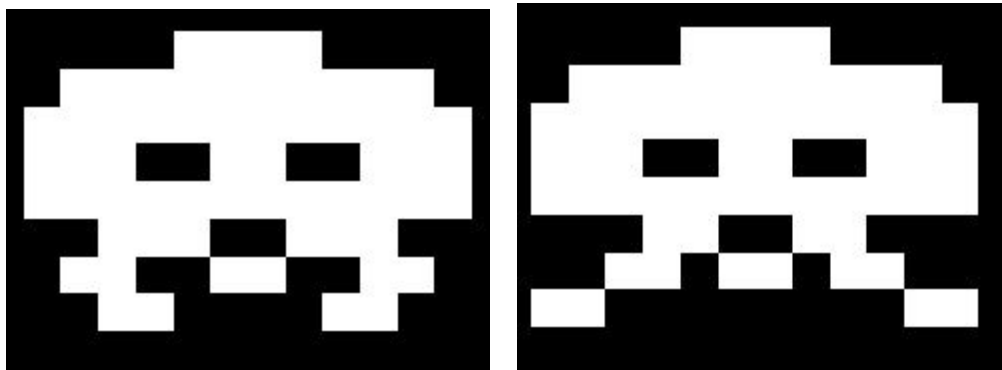
Mi samo želimo mijenjati sliku ovisno o vremenu. Pa odemo u sprites direktorij i otvorimo Spaceinvaders sprite.



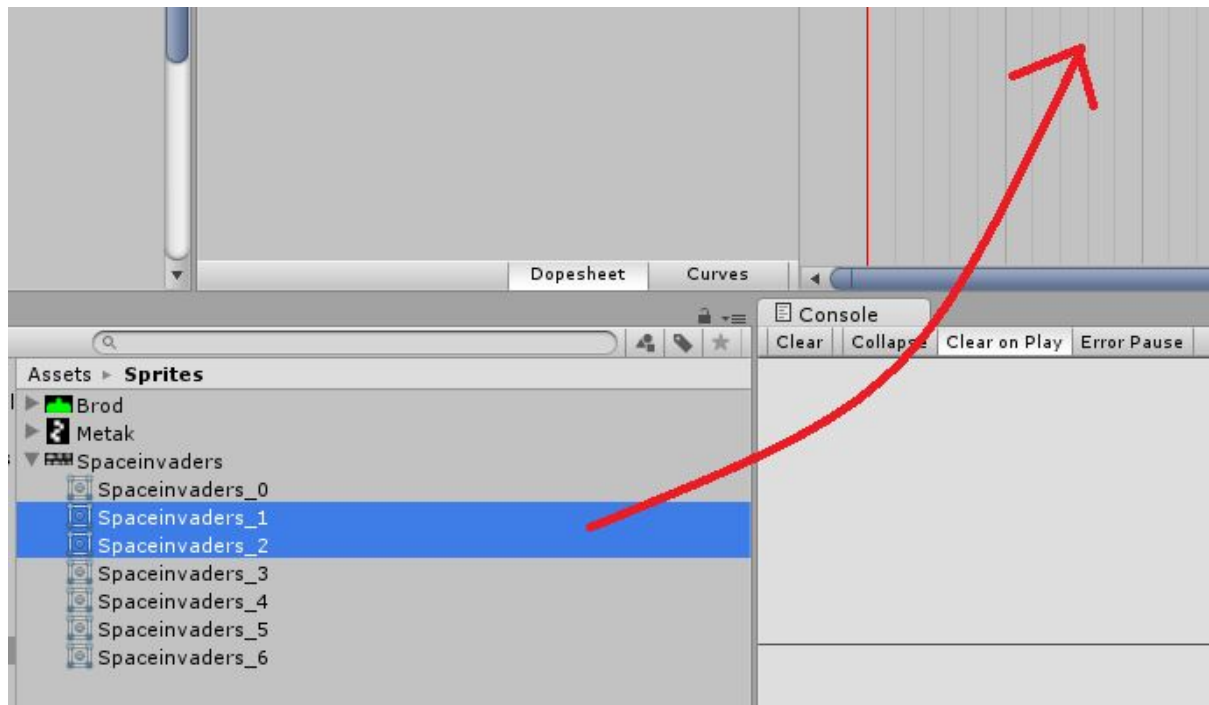
U slučaju da je samo jedna slika unutar Spaceinvaders slike. Označite za nju u inspectoru umjesto “Single” “Multiple”. Pa u sprite editoru označite mišem koje sve slike želite da izreže.

Označimo koje želimo koristiti u našoj animaciji.

Za prvog invadera to su:



Samo ih u *project* panelu označimo i povučemo u vremensku traku.



I u property dijelu promijeni sample sa 60 na 2 i animacija je gotova.
Dodamo isti animator Invader1 na svaki Invader1 *GameObject*.

I samo ponovimo postupak za Invadera2 i Invader3.

Naša igra je gotova.

14. Zaključak

Tokom pisanja skripti smo mogli uočiti da samo korištenjem funkcija *Update*, *FixedUpdate* i *OnCollisionEnter2D* smo uspjeli cijelo ponašanje igre napraviti.

Pokazali smo definiranje gibanja na više načina, sa silom i trenjem te definiranjem konstantne brzine. To npr. možemo koristiti prilikom izrade platformer-a za skakanje koristimo jako silu prema gore na jedan frame-a za gibanje lijevo desno isto kao naš svemirski brod.

Ovo smatram da može bit dobar početak. Osnove osnova su prikazani i definirani ovdje. Za dodatnu literaturu preporučujem tutorijale na službenoj stranici.

<https://unity3d.com/learn/tutorials/topics/2d-game-creation>