Colegiul Naţional "Roman-Vodă"
Roman

Space Invaders 3D - joc

Lucrare pentru obținerea atestatului profesional la informatică

Candidat,
Enia Vlad Ieftimie

Coordonator
prof. Florin Moldovanu

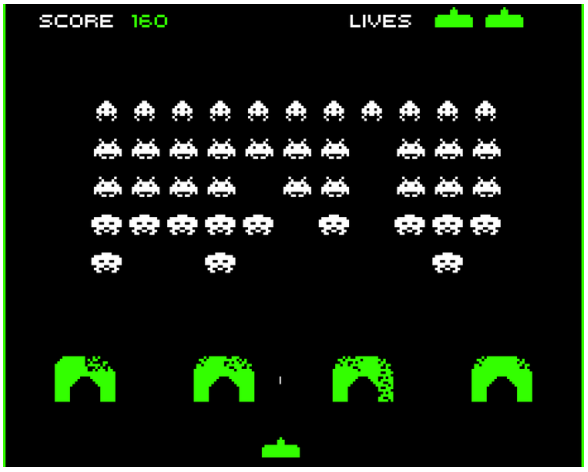
2019

Cuprins

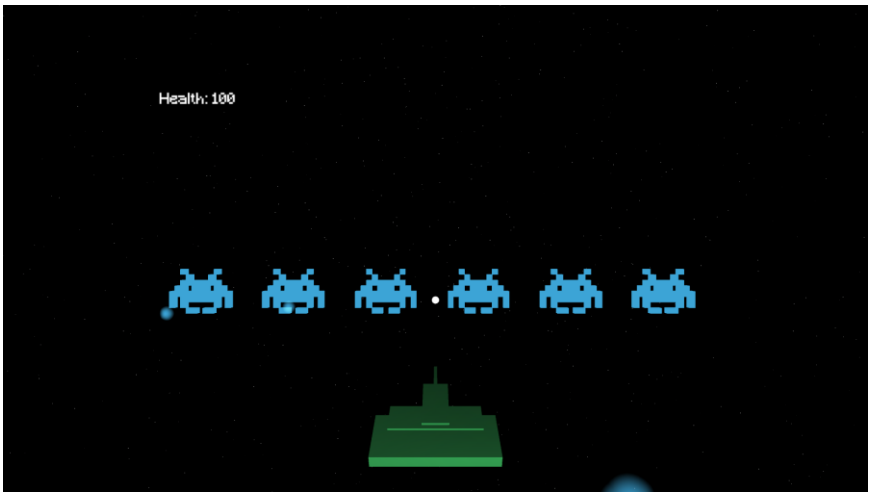
0. Argument

Am ales drept temă de atestat reinterpretarea, pentru sistemul de operare Windows, a binecunoscutului joc arcade ***Space Invaders***, dezvoltat de Tomohiro Nishikado și lansat în 1978 sub numele de *Taito*, în Japonia. Mai târziu, acesta a fost licențiat în Statele Unite de către divizia *Midway* a companiei *Bally*. *Space Invaders* este unul dintre primele jocuri cu împușcături iar obiectivul este de a rezista cât mai mult un unei invazii de nave extraterestre, structurată pe grupuri numite *wave*, cu ajutorul unui tun laser.

Așa cum am menționat mai sus, proiectul reprezintă reinterpretarea jocului, acesta fiind realizat 3D, cu perspectivă din spatele navei, în timp ce jocul original este 2D, cu perspectivă deasupra navei. Proiectul a fost realizat cu ajutorul motorului grafic ***Unity*** și a limbajului de programare **C#**.



Jocul original, cu interfață 2D



Jocul reinterpretat, cu interfață 3D

1. Introducere în Unity și C#

1.0. Despre limbajul C#

C# este unul dintre cele mai utilizate limbaje de programare multiparadigmă din lume. Este un limbaj simplu, modern, cu o flexibilitate foarte mare în ceea ce privește dezvoltarea de aplicații și portabilitatea acestora. A fost creat de *Microsoft* ca soluție standard pentru dezvoltarea aplicațiilor *Windows*, însă compilatoare C# există și pentru alte sisteme, precum *Linux* sau *Macintosh*.

Printre principalele calități ale limbajului, putem distinge:

- modernitate, simplitate, utilitate generală, productivitate mare
- stabilitate în cadrul aplicațiilor complexe, durabilitate
- este un limbaj total orientat pe obiect (orice entitate din acest limbaj este de fapt un obiect)
- oferă suport complet pentru dezvoltarea de componente necesare în medii distribuite, deci este și un limbaj orientat către componente

C# derivă din C și C++ și este o "rudă" apropiată a limbajului Java, având însă o serie de avantaje față de acestea.

Avantaje față de C++:

- Sistemul garbage-collection e nativ
- Biblioteca standard foarte bogată cu lucruri bine implementate și usor de folosit
- Clasele și metodele pot fi setate să fie interne în ansamblul în care sunt declarate
- Permite tratarea semnăturilor de clasa-metode ca funcții libere (ignoră argumentul pointer this) și creează astfel relații mai dinamice și flexibile între clase
- Permite folosirea blocurilor de cod atât native, cât și gestionate

Avantajele față de Java:

- În loc de implementările de clase și EJB, C# are constructori proprii ușor de folosit precum Properties sau Events
- Este integrat în Windows
- Are Lambda și LINQ, suportând o parte de programare funcțională
- Are variabile dinamice
- Permite definirea de tipuri noi

1.1. Prezentarea motorului grafic Unity

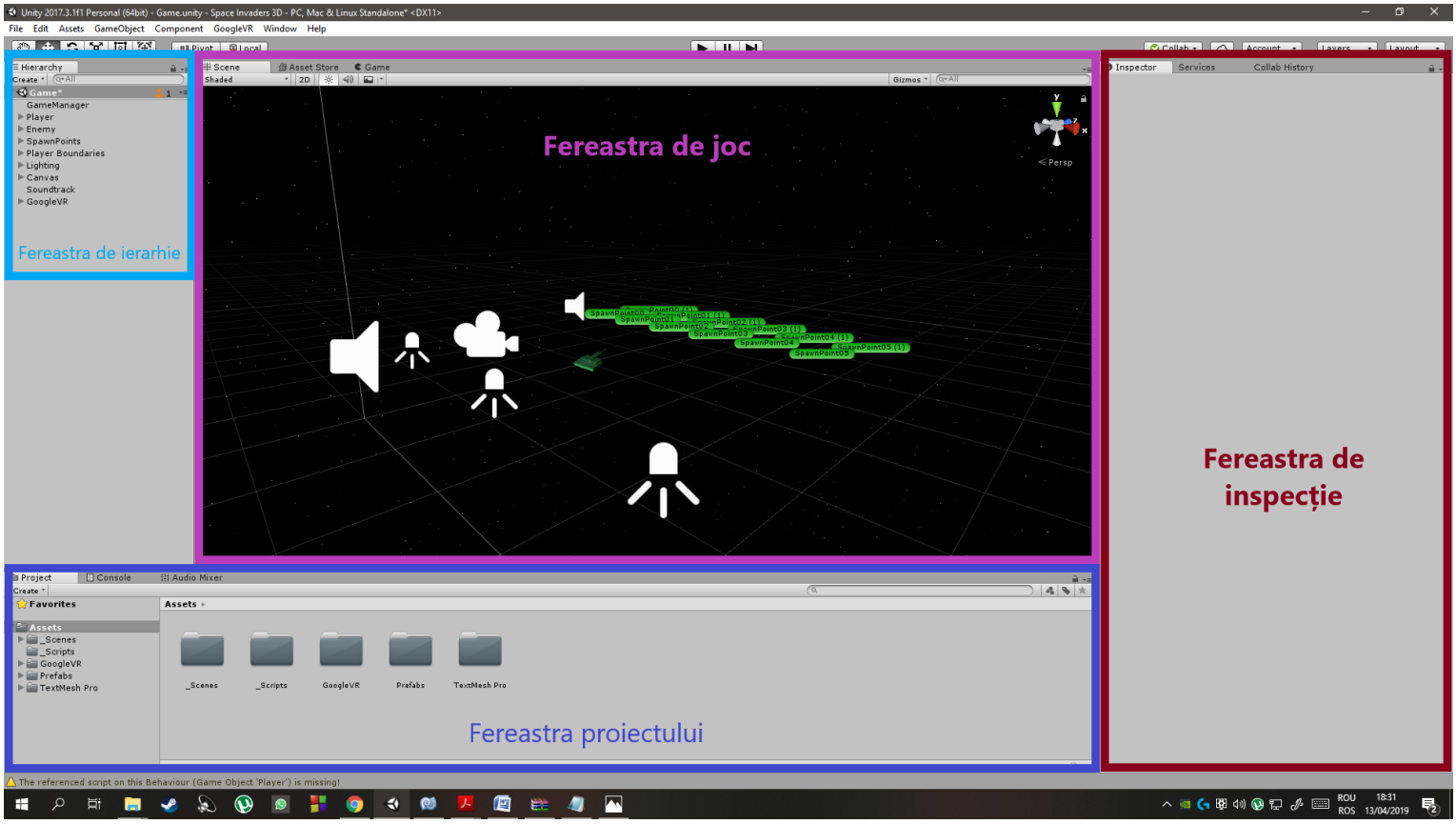
Unity este un motor grafic cu un mediu de dezvoltare integrat, fondat în 2005 de către *Unity Technologies*, utilizat pentru crearea de programe, în special de jocuri video, atât 2D cât și 3D, pentru o varietate de platforme, precum Windows, Macintosh, Linux, Android, iOS, Xbox, PlayStation, sau WebGL.

Limbajele de programare ce pot fi folosite în Unity sunt C#, UnityScript (o variantă adaptată de JavaScript).

Unity permite crearea unor jocuri deosebit de complexe, punând la dispoziție numeroase unelte vizând gameplay-ul, efectele vizuale, optimizarea și testarea rapidă a codului.

1.2. Editorul Unity și aspecte generale

Editorul Unity este alcătuit din mai multe ferestre:



Cele mai utilizate ferestre sunt:

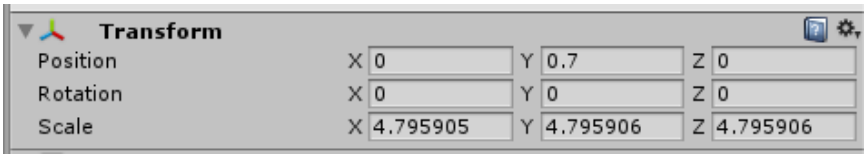
- **Fereastra de ierarhie** (Hierarchy Window), care reprezintă ierarhic toate obiectele dintr-o scenă, cu legăturile stabilite între acestea.

- **Fereastra de joc** (Scene View), care permite navigarea într-o anumită scenă a jocului și editarea componentelor sale. Aceasta poate funcționa atât în perspectivă 2D, cât și în 3D, în funcție de tipul de joc ales.
- **Fereastra de proiect** (Project Window), care conține toate resursele (imagini, modele 3D, fonturi, script-uri, sunete etc.), organizate în fișiere.
- **Fereastra de inspecție** (Inspector Window), ce permite vizualizarea și modificarea componentelor unui anumit obiect, componente ce descriu proprietățile acestuia.

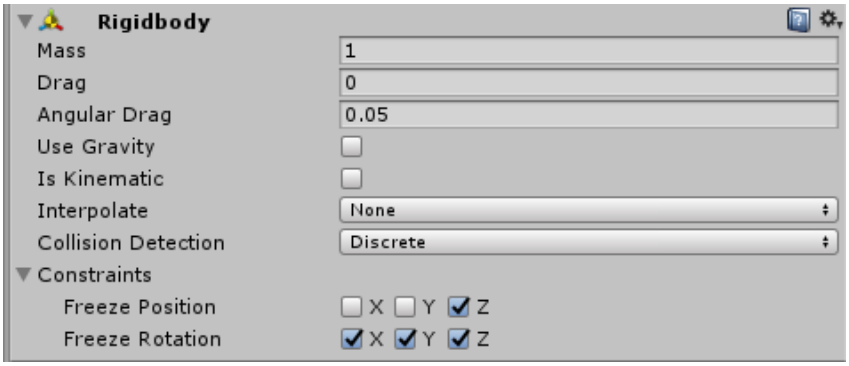
Un proiect realizat în motorul grafic Unity este structurat conform unor concepte generale, precum:

- **Scena**. Un proiect poate avea mai multe scene. Ea reprezintă mediul de desfășurare al jocului, si în cele mai multe cazuri ea poate servi rolul de meniu sau de nivel. Scena are cel mai înalt rol în fereastra de ierarhie și cuprinde obiectele din cadrul acesteia.

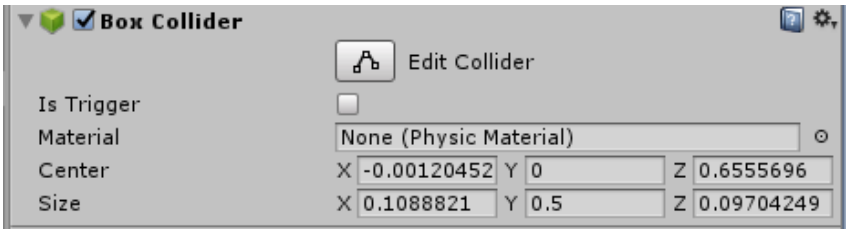
- **Obiectul** reprezintă o entitate din cadrul unei scene. El poate fi introdus sub diverse forme, fiecare având diverse proprietăți. Odată adăugate în scenă, obiectele apar atât în fereastra de ierarhizare cât și în fereastra de joc. Obiectele pot fi ierarhizate și grupate în cadrul ferestrei de ierarhizare.
- **Componentele** definesc proprietățile unui obiect. Ele apar în fereastra de inspecție atunci când obiect este selectat. Cele mai importante dintre acestea sunt:
    - **Transform**, care conține pozitia, dimensiunile, și rotația unui obiect conform sistemului de axe XYZ.



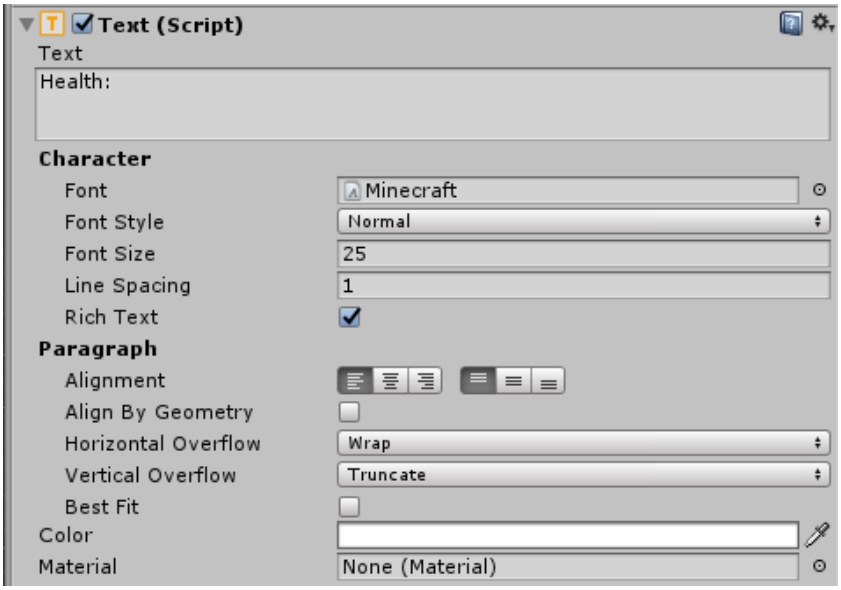
    - **RigidBody**, care cuprinde aspectele fizice ale unui obiect. În cadrul acestei componente, se pot modifica masa, rezistența, acțiunea unor diverse forțe, fixarea pe anumite axe și multe alte aspecte fizice.



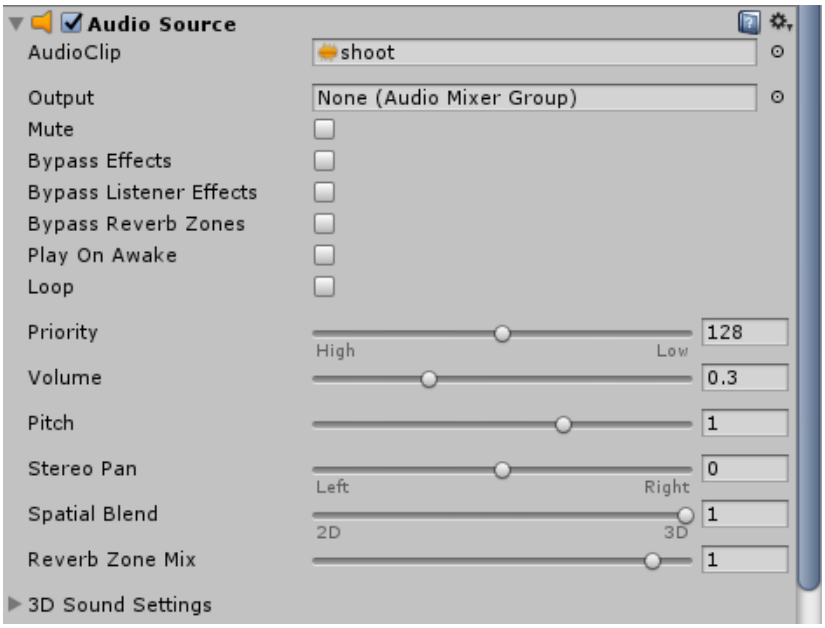
- **Collider**, care reprezintă componenta cu ajutorul căreia obiectele interacționează între ele prin coliziuni. Această componentă poate avea diverse forme ce pot fi plasate în cadrul unui obiect pentru a delimita suprafața de interacțiune a acestuia.



- **Text**. Cu ajutorul acestei componente se poate adăuga text într-o scenă. Aceste componente apar cel mai des în obiectele de interfață, cum ar fi butoanele sau titlurile ce apar într-un meniu și multe altele.

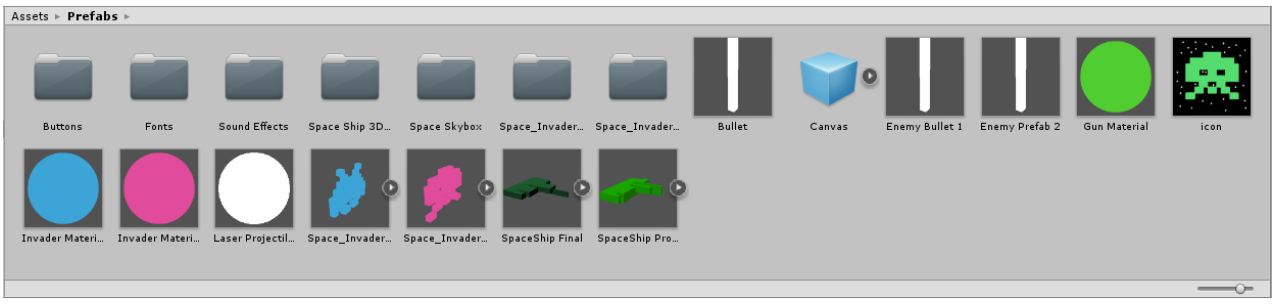


- **Audio source**, reprezintă componenta prin care putem atribui efecte de sunet unui obiect. În cadrul acesteia, se pot modifica diverse prorietăți audio cum ar fi volumul, înălțimea, felul în care acesta se propagă prin scenă, și multe altele.



- **Material**, care conține proprietăți legate de aspectul obiectului, cum ar fi culoarea, textura, opacitatea, strălucirea și multe altele.
- **Script-ul.**

- **Script-ul** este folosit pentru a defini comportamentul unui obiect. El este scris în limbajul C#. Un script conține variabile publice (care pot fi modificate în editor, mai exact în fereastra de inspecție) și private (care apar doar în cadrul script-ului), care la rândul lor pot fi din categoria componentelor menționate mai sus, sau de tip int, float, bool și asa mai departe.
- **Prefab-ul**, care reprezintă o resursă a jocului și de cele mai multe ori are forma unui obiect 3D care poate fi plasat de ori se dorește în scenă. Totuși, prefab-uri sunt considerate și alte resurse ale unui proiect, cum ar fi fișierele audio, fonturile, obiecte de interfață și multe altele. Prefab-urile sunt organizate în fereastra de proiect.

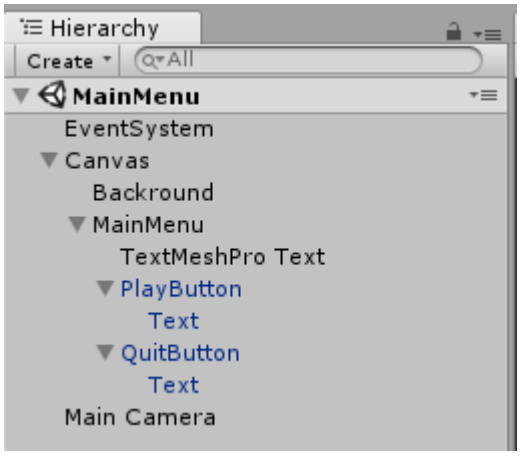


2. Prezentarea proiectului

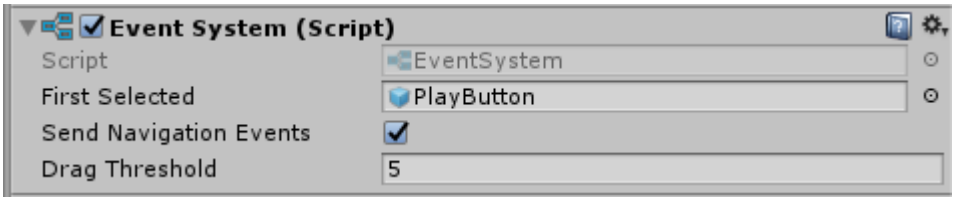
Pentru prezentarea proiectului vom aborda fiecare scenă, fiecare obiect din cadrul acesteia și componentele acestuia, în ordinea lor din fereastra de ierarhizare.

2.0. Scena meniului principal (MainMenu)

După cum se poate vedea, meniul este unul simplu, similar cu cel din jocul original. Însă în cadrul acestei scene, există și alte obiecte, care nu sunt vizibile, după cum ne sugerează fereastra de ierarhizare:

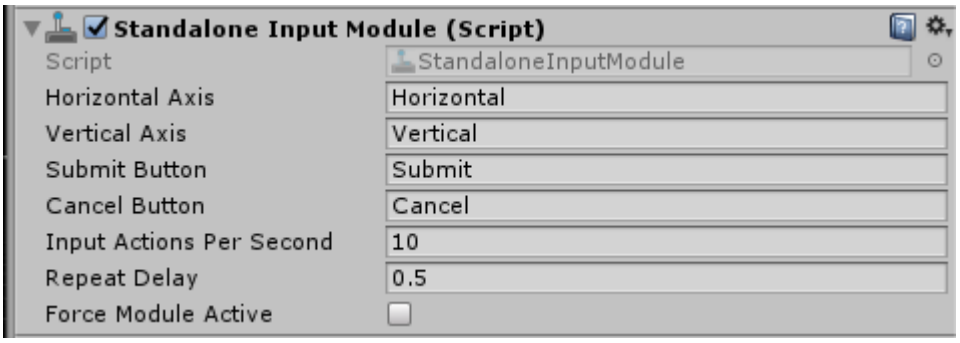


- **EventSystem** definește aspecte generale ale comportamentului scenei. Conține următoarele componente:
    - Event System (Script)



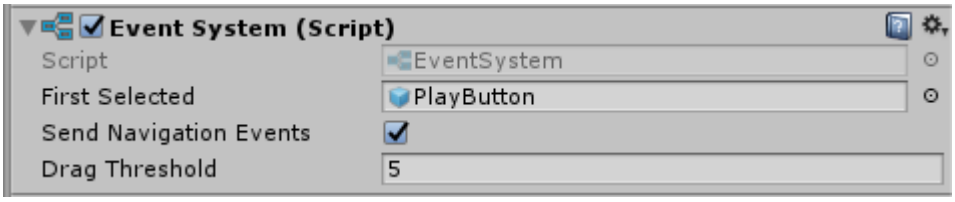
Cel mai important de menționat aici este câmpul *First Selected*, în care este introdus butonul de play. Astfel, la fiecare accesare a meniului, *PlayButton* este selectat în mod implicit.

    - Standart Input Module (Script)



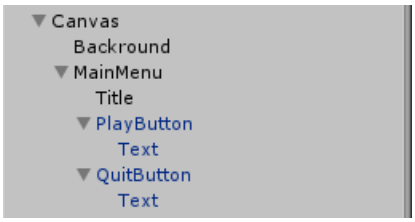
Așa cum sugerează și numele, această componentă reține elementele de imput, adică tastele care interacționează cu jocul. Astfel, avem tastele cu săgeți verticale pentru mișcarea pe axa verticală, tastele cu săgeți orizontale pentru mișcarea pe axa orizontală, tastele *enter* si *space* (care se identifică prin tag-ul *submit*) pentru submit, și botunul *escape* (identificat prin tag-ul *cancel*) pentru acțiunea de anulare.

    - Event System (Script)



Cel mai important de menționat aici este câmpul *First Selected*, în care este introdus butonul de play. Astfel, la fiecare accesare a meniului, *PlayButton* este selectat în mod implicit.

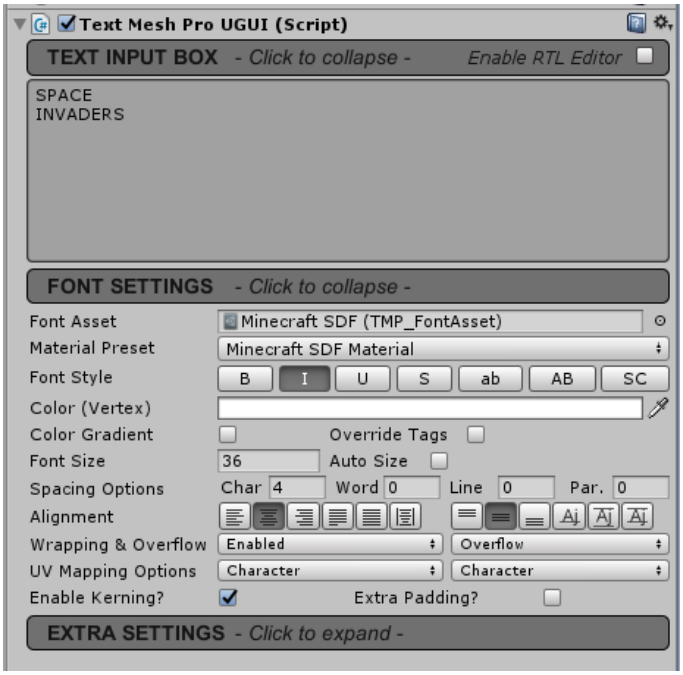
- **Canvas**, așa cum sugerează și numele, este ca o "pânză", pe care de regulă se plasează diferite elemente de interfață:



  - **Backround** reprezintă imaginea de fundal a canvas-ului, reprezentată în cazul de față de spațiul presărat cu stele, reprezentativ pentru tema acestui proiect.
  - **MainMenu** reprezintă obiectul care conține toate elementele vizibile ale meniului principal și scriptul respnsabil pentru funcționalitatea acestuia

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.SceneManagement;
5.
6.  public class MainMenu : MonoBehaviour {
7.
8.      public void PlayGame()      // Aceasta functie asigura functionalitatea butonului de play;
9.      {
10.         SceneManager.LoadScene ("Game"); // Astfel, prin apasarea butonului play,se va
                                             trece la următoarea scena, cea a jocului propriu-zis;
11.     }
12.
13.     public void QuitGame()      // Aceasta functie asigura functionalitatea butonului quit;
14.     {
15.         Debug.Log ("QUIT");
16.         Application.Quit ();
17.
18.     }
19. }
```

  - **Title**, reprezintă obiectul care conține textul titlului din centrul meniului. Acesta nu este un obiect de text simplu, ce este de tip *Text Mesh Pro*, o unealtă descărcată de pe de pe magazinul virtual *Asset Store*, pus la dispoziție de Unity. Această unealtă de oferă mai multe și mai bine structurate opțiuni de editare a textului.



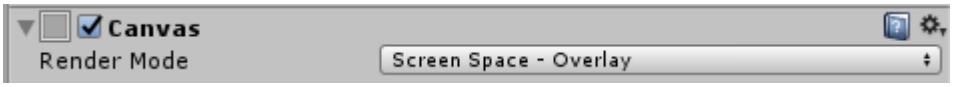
După cum se poate observa, avem o multitudine de opțiuni de editare a textului. Cel mai important de menționat este fondul jucăuș, reprezentativ pentru aspectul retro al jocului, lucru realizat prin importarea unui font numit *Minecraft*.

- **PlayButton**, așa cum sugerează și numele, reprezintă un obiect de tip buton, care lansează jocul, prin încărcarea scenei propriu-zise a acestuia. Funcționalitatea acestuia este asigurată de script-ul din obiectul *MainMenu*. De menționat este modul în care butonul reacționează atunci când se trece cu pointer-ul mouse-ului deasupra acestuia, lucru realizat prin modificarea unor setări alea componentei de material a acestuia:

Totodată, butonul conține și un obiect de tip *Text Mesh Pro,* care conține textul "Play".

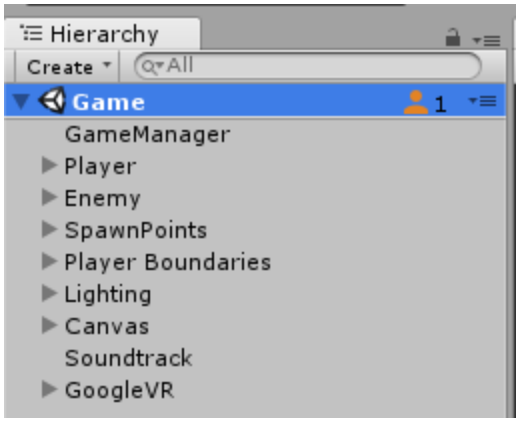
- **QuitButton** este un obiect identic cu PlayButton, singurele diferențe fiind funcția acestuia de a părăsi jocul și textul pe care acesta îl conține.

- **MainCamera** reprezintă un obiect crucial în orice scenă, deoarece el este responsabil cu ceea ce se afișează atunci când se rulează jocul. Astfel, ceea ce se vede în joc variază în funție de poziția, rotația, și unghiul de vizualizare al camerei. În cazul de față, canvas-ul este programat să apară pe ecran indiferent de orientarea camerei, cu ajutorul acestei opțiuni:

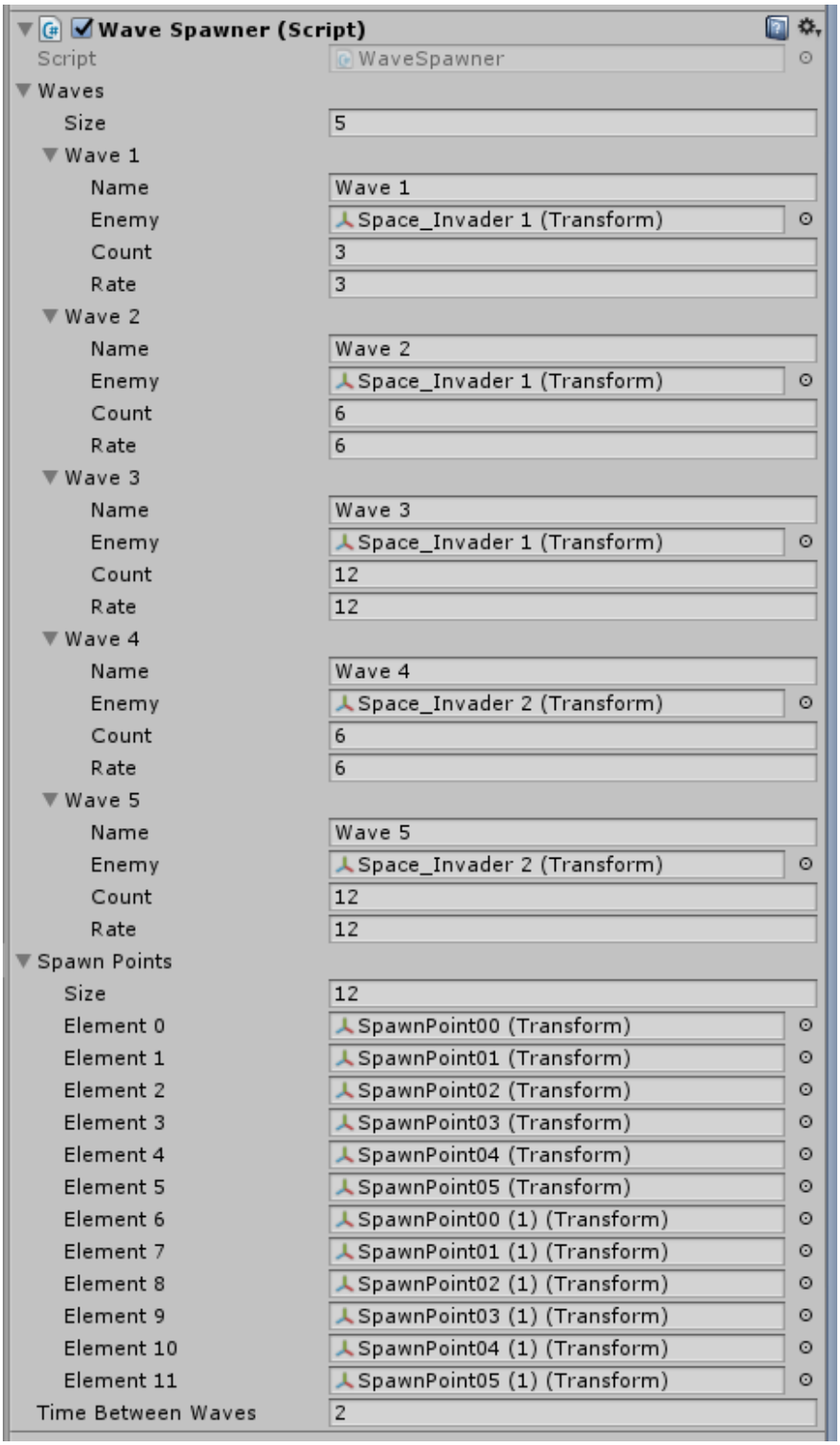


2.1. Scena jocului propriu-zis (Game)

Scena aceasta reprezintă esența jocului. Deși acesta a fost reinterpretat, conceptul acestuia a rămas fidel jocului care a inspirat acest proiect: jucătorul este în controlul unei nave (asemănătoare cu nava originală, însă cu aspect 3D), care se poate mișca pe axa orizontală și care trage cu un tun laser spre inamici (de asemenea cu aspect identic cu cel din jocul original).
Aceștia, la rândul lor, lansează și ei proiectile spre jucător, care în cazul unei coliziuni degradează starea navei, până la distrugere, care înseamnă GAME OVER. Dupa încetarea jocului, pe ecran va apărea un meniu de game over, împreună cu opțiunile de restart, întoarcere la meniul principal și de părăsire a jocului.
De asmenea, jucătorul poate întrerupe jocul în orice moment, acesta fiind întâmpinat cu un meniu de pauză, care conține opțiunile de reulare a jocului, de întoarcere la meniul principal, și de quit.
În joc apar două tipuri de inamici, unul mai distrugător ca celălalt, iar invazia acestora este structurată în grupuri numite *waves.* Astfel, după distrugerea unui wave, în locul acestuia va aparea altul, care poate conține mai mulți inamici decât cel precedent.
Fereastra de ierararhizarea a scenei jocului conține următoarele:

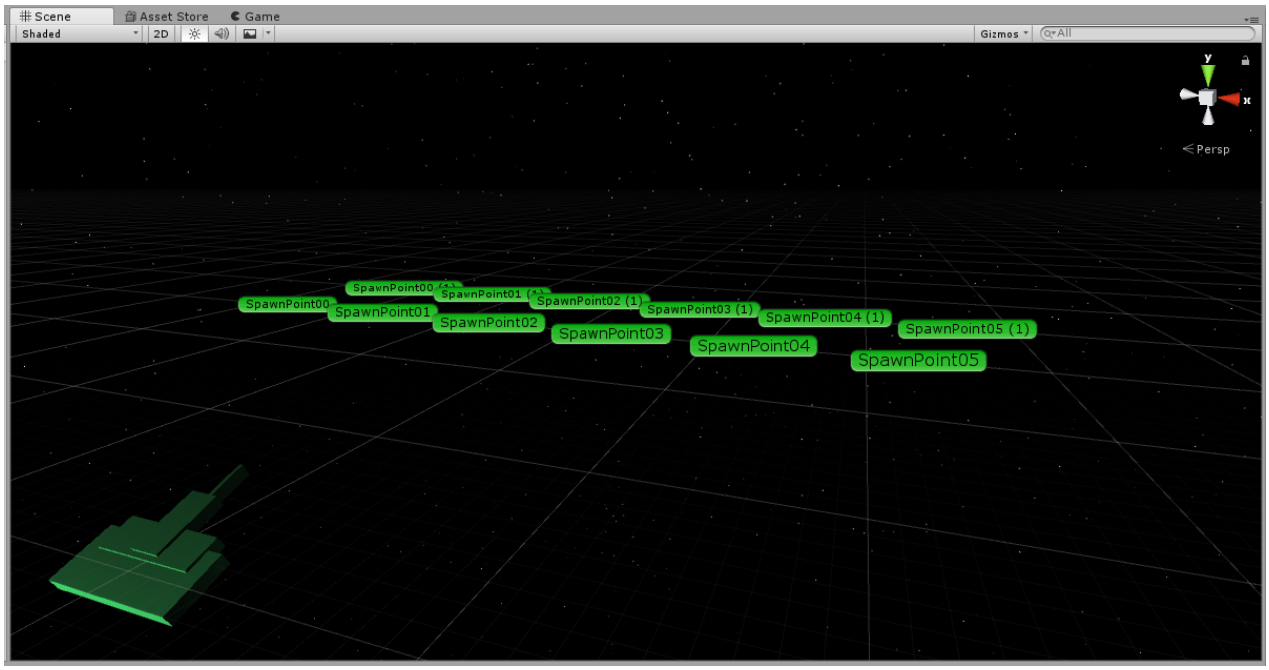


- **GameManager.** Așa cum sugerează și numele, obiectul acesta este ca un supervizor al scenei, întrucât el asigură cel mai important aspect al acesteia, și anume generarea inamicilor, realizată cu ajutorul unei singure componente, și anume WaveSpawner, cu următoarea structură:



Aceasta este alcătuită din doi vectori:

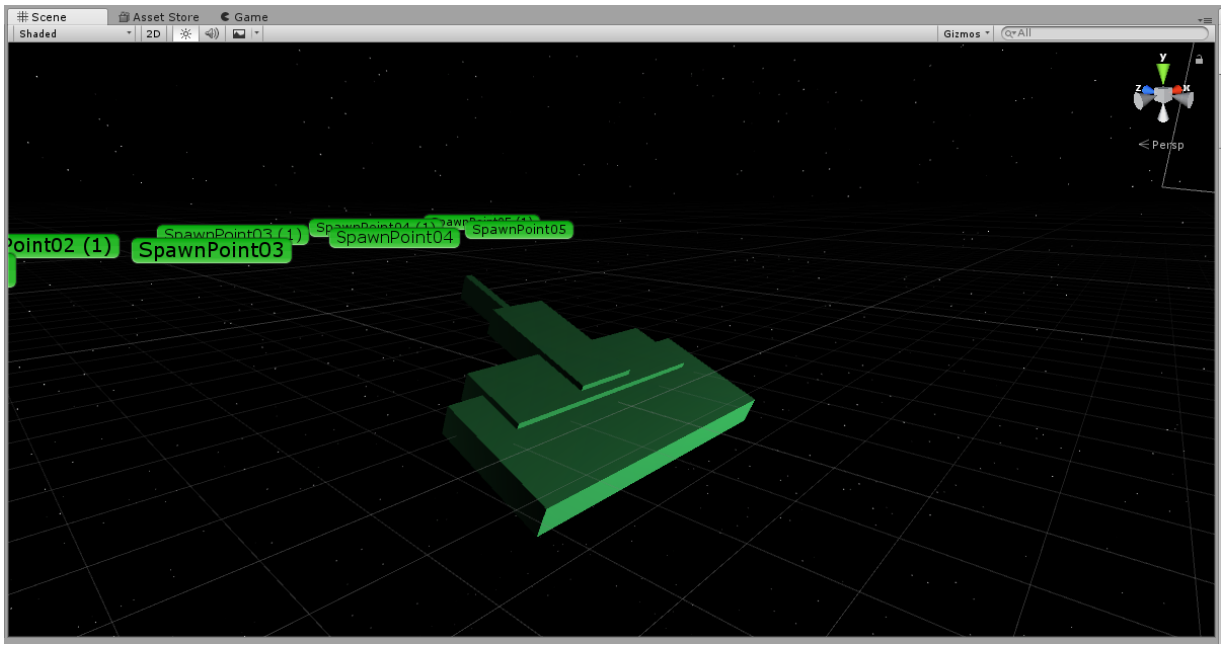
- **Waves**, cu dimensiunea *Size*, si apoi elementele acestuia, fiecare conținând
  - Numele wave-ului;
  - Tipul inamicului pe care în generează, reprezentat de câmpul *Enemy;*
  - Numărul de inamici din acel wave - *count;*
  - Frecvența cu care aceștia apar - *rate;* după cum se poate observa, în cazul setărilor actuale, count și rate au aceeași valoare, astfel încât în fiecare wave inamicii vor fi generați toți odată.
- **SpawnPoints**, cu dimensiunea *Size*, care reprezintă punctele în care inamicii vor apărea; astfel, în fiecare câmp al elementului vectorului este atribuit câte un obiect de tip transform, preluat din scenă, care conține informațiile legate de poziția punctelor în care inamicii vor fi generați.



Printscreen din scenă, care cuprinde cele 12 spawnpoint-uri, plasate în fața navei jucătorului.

- **Player** este un alt obiect vital din scena proiectului. El reprezintă mai degrabă un ansamblu de malte cateva obiecte, care împreună asigură funcționalitatea corectă a acestuia:

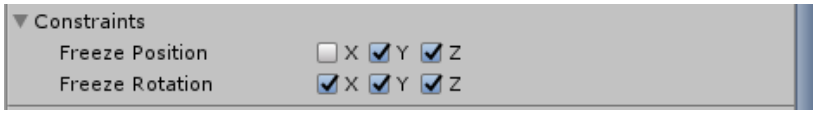
  - **MainCamera**, în această scenă, este atașată de entiatea *Player*, deoarece ne dorim ca aceasta să se miște impreună cu nava jucătorului;
  - **SpaceShip Final** reprezintă modelul 3D al navei, care este și el atașat entității *Player*.



Prinscreen cu modelul 3D al navei, plasat în scenă

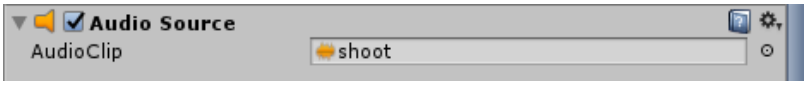
Alte elemente care nu sunt vizibile sunt cele care contribuie cel mai mult la funcțiile navei jucătorului. Acestea sunt reprezentate de componentele obiectului, care includ:

  - **RigidBody**, care, așa cum am menționat mai sus, reprezinta componenta care asigura aspectele fizice ale obiectului, cum ar fi masa, rezistența, și alte forțe care sunt aplicate. Cel mai important de menționat este faptul ca obiectul navei jucatorului se miscă doar pe axa orizontală, el rămânând fix pe celelalte axe și, totodată, el nu se poate roti. Acest lucru l-am asigurat prin următoarele constrângeri, în componenta RigidBody:



  - **Audio Source**. Obiectul navei dispune de 2 surse audio diferite, astfel încât el poate emite două tipuri de sunete:

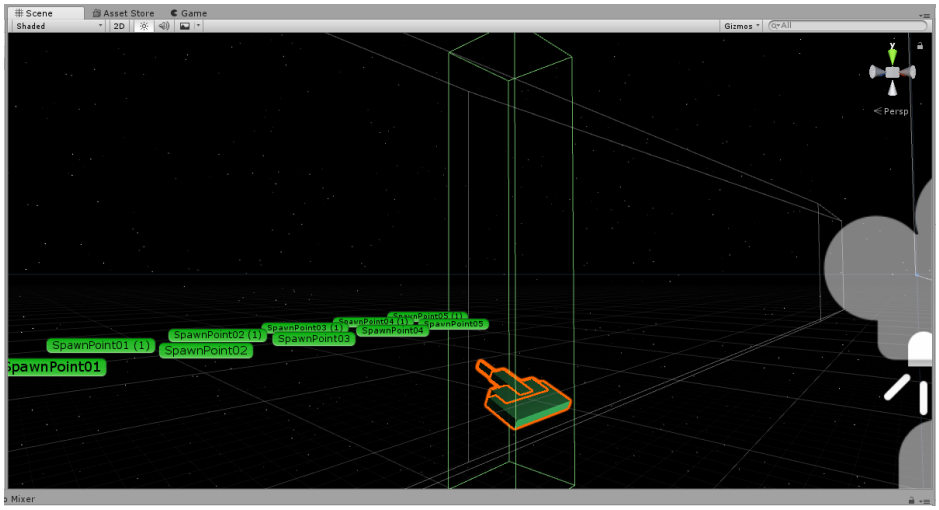
- Sunetul produs la lansarea proiectilului din "tunul laser":



- Sunetul emis la coliziunea unui proiectil inamic cu nava, asemănător unei explozii:

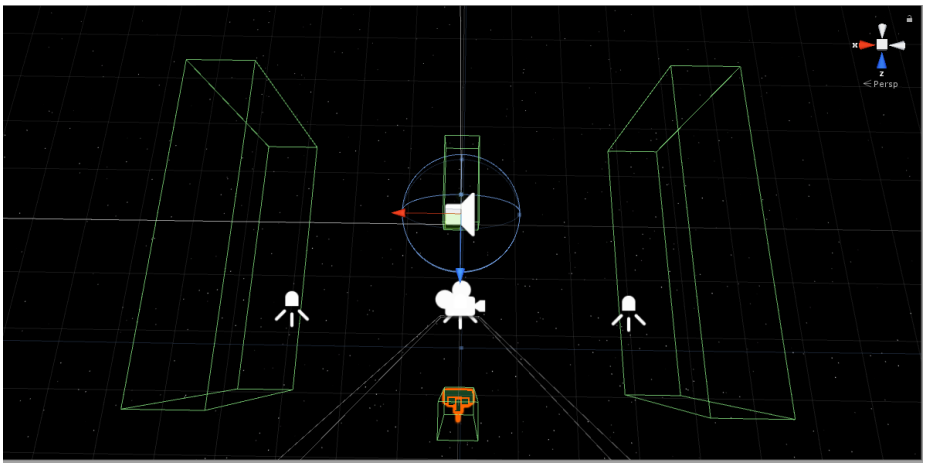
De menționat este faptul că efectele de sunet au fost preluate din jocul original, și importate în proiect, tocmai pentru a facilita crearea unei atmosfere asemănătoare.

- **Collider.** În cazul de față, obiectul player include 2 collidere, fiecare având câte un rol important:
  - Collider-ul ce înconjoară nava; acesta are rolul de a interacționa cu proiectilele lansate de inamici, și cu ajutorul unui script, în cazul unei coliziuni, se produce sunetul de explozie menționat mai sus și scade viața jucătorului;



Collider-ul este reprezentat în scena sub forma unui chenar transparent, cu muchiile verzi.

  - Collider-ul plasat în spatele navei, care ne ajută la impunerea unei limite în care nava se poate deplasa. Acest lucru e realizat cu ajutorul altor 2 collidere, plasate la limita din stânga și din dreapta a spațiului de joc:



Ansamblul de collidere care ne asigură că jucătorul nu va părăsi spațiul de joc.

- **Script-uri**. În cazul de față, obiectul navei include 3 script-uri, scrise în C#, fiecare fiind respnsabil cu o funcționalitate.
    - **Player Controller** reprezintă script-ul care ne oferă posibilitatea de a mișca nava folosind tastele cu săgeți orizontale:

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4.
5. public class PlayerController : MonoBehaviour {
6.
7.     private Rigidbody rb;       //Rigidbody reprezinta aspectul fizic al entitatii jucatorului, care
                                     poate fi miscat si care se supune legilor fizicii;
8.     public float speed;          //Variabila "speed" reprezinta viteza cu care putem misca jucatorul
                                     Deorece este de tip public, ea poate fi modificata din editor;
9.     void Start()
10.         {
11.             rb = GetComponent<Rigidbody> ();
                //Variabila "rb" este legata de aspectul rigidbody al jucatorului;
12.         }
13.
14.         void FixedUpdate ()
15.         {
16.             float moveHorizontal = Input.GetAxis ("Horizontal");
17.             //Jucatorul poate fi miscat doar
18.             in stanga si dreapta, adica doar pe axa orizontala a scenei;
19.             Vector3 movement = new Vector3 (moveHorizontal, 0.0f, 0.0f);
20.             //Variabila "movement" este de tip vector3; acest tip modifica pozitia obiectului pe cel
                e 3 axe, incepand cu axa orizontala;
21.             rb.velocity = movement*speed;
22.             //Asfel, pozitia jucatorului va fi modificata cu viteza "speed", declarata
                mai sus;
23.         }
24.     }
```

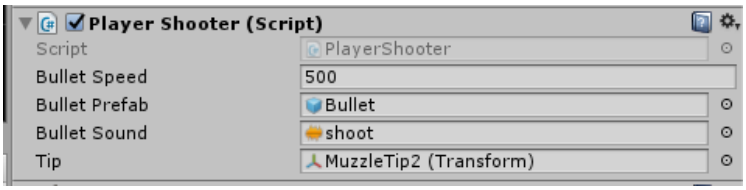
Câmpul *speed*, corespunzător variabilei *publice* din script, poate fi modificat în fereastra de inspecție:

- **Player Shooter** este script-ul care ne oferă posibilitatea de a lansa proiectile spre inamici. Totodată, el este responsabil și pentru emiterea sunetului produs de tunul laser:

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class PlayerShooter : MonoBehaviour {
6.      public float bulletSpeed; //Variabila "bulletspeed" reprezinta viteza cu care proiectilul se va
        deplasa in scena; deoarece este de tip public, ea va putea fi modificata din editor;
7.      public GameObject bulletPrefab; //Variabila "bulletPrefab" reprezinta entitatea caruia ii vom
                                        putea atribui obiectul 3D al proectilului, care va aparea in scena;
8.
9.      public AudioClip bulletSound;//Variabilei "bulletSound" ii vom atribui sunetul produs de jucator
                                     atunci cand lanseaza proiectilul;
10.     public Transform tip; //Variabila "tip(=varful navei)" este de tip Transform, adica reprezinta
```

```
                            locatia dincare proiectilul va fi lansat;
    void Update ()
    {
        if(Input.GetButtonDown ("Submit")) //La apasarea butonului de pe tastatura cu ta-
    gul "submit",
                                           reprezentate de butoanele "enter" si "space"
        {
            AudioSource audio = GetComponent <AudioSource> ();
            audio.PlayOneShot (bulletSound);                    //se produce sunetul;
            OnShoot();                                          //si se apeleaza functia OnShoot();

}
    }


    public void OnShoot()
    {
        Debug.Log("I shoot");
        GameObject bullet = Instantiate(bulletPrefab, tip.position, tip.rotation, transform);
    //se creeaza un nou proiectil in scena, avand pozitia (tip.position) si rotatia (tip.rotation)
    varfului navei (tip)

        bullet.transform.Rotate(Vector3.right, 90f);
    //din cauza felului in care a fost construit modelul 3d al proiectiluli, el va trebui rotit la
    90 grade;

        bullet.GetComponent<Rigidbody>().AddForce(tip.forward * bulletSpeed , ForceMode.Acceleration
    );
    //proiectilul este lansat din fata varfului navei (tip.forward) cu viteza "bulletSpeed";
    }
}
```

Astfel, variabilele *publice* vor apărea sub formă de câmpuri pe care le putem completa în fereastra de inspecție:

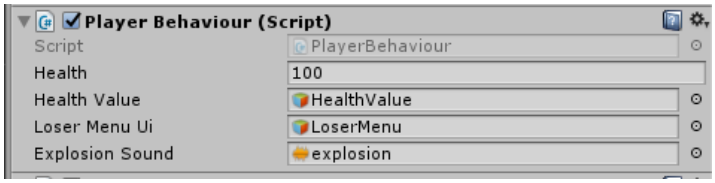


- **Player Behaviour** controleaza mai multe aspecte legate de comportamentul navei jucătorului, cum ar fi viața cu care pornește, felul în care ea este afectată de proiectilele inamice, sunetul produs de coliziuni, dar și afișarea meniului de game over:

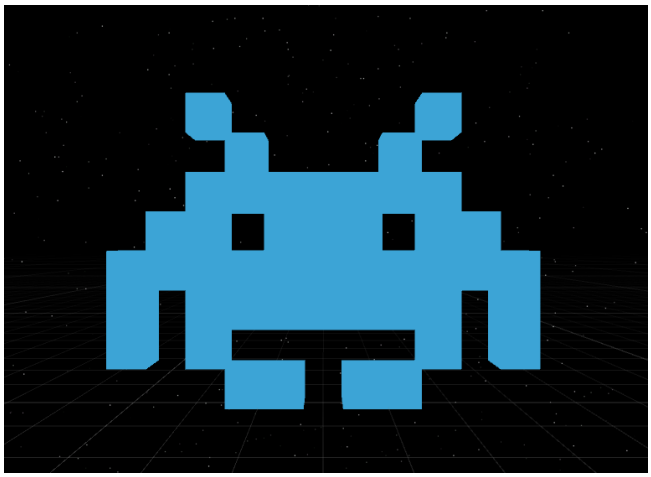
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PlayerBehaviour : MonoBehaviour {

    public int health;              //Variabila "health" reprezinta variabila de tip int cu care
      script-ul va                          lucra
    public GameObject healthValue; //Variabila "healthValue" contine viata jucatorului;
     public GameObject loserMenuUi; //Variabila "loserMenuUi" reprezinta obiectul care contine
                                      meniul de game over
    public AudioClip explosionSound; //Variabila "explosionSound" contine sunetul produs la impactul
                                      dintre proiectilul inamicului si jucatorul;
```

```
13.     public void OnCollisionEnter (Collision collision) //Aceasta functie este folosita pentru a
                                                          ilustra efectul unei lovituri din partea inamicului;
14.     {
15.         Debug.Log("Hit!!");
16.         if (collision.gameObject.tag == "Enemy Bullet") //Astfel, la coliziunea cu proiectilul
17.                                                         inamicului
18.         {
19.             AudioSource audio = GetComponent <AudioSource> ();
20.             audio.PlayOneShot (explosionSound);
21.             if (health > 25) {                          //daca avem mai mult de 25 viata ramasa,
22.                 health -= 25;                           //viata va scadea cu 25
23.                 healthValue.GetComponent<Text> ().text = health.ToString ();
24.                    // iar valoarea curenta a vietii va fi afisata pe ecran;
25.             } else {
26.                //daca nu mai avem destula viata, atunci se va apela functia pause();
27.                 health -= 25;
28.                 healthValue.GetComponent<Text> ().text = health.ToString ();
29.                 Pause ();
30.             }
31.         }
32.
33.     }
34.
35.     void Pause() //Aceasta functie este responsabila pentru accesarea meniului de game over;
36.     {
37.         loserMenuUi.SetActive (true); //Astfel, pe ecran va aparea meniul de game over in sine;
38.         Time.timeScale = 0f; //Timpul se va "opri", jocul intrand in repaus;
39.     }
40. }
```

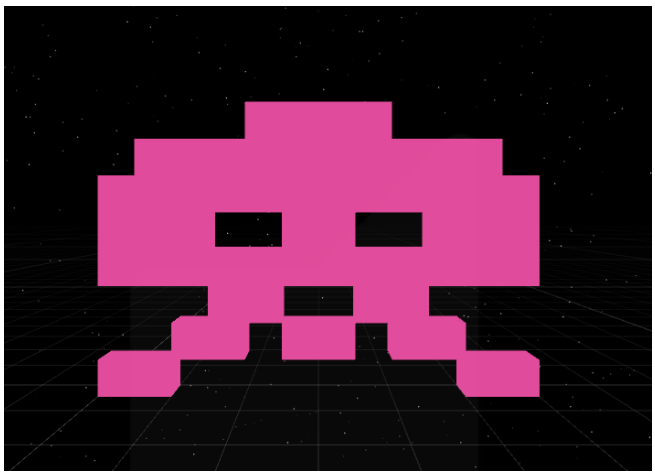
La fel ca și la scriptul anterior, variabilele de tip *public* vor apărea sub formă de câmpuri în fereastra de inspecție:



- **Enemy** reprezintă entitatea inamicilor din joc. De-a lungul invaziei vor apărea 2 tipuri de nave extraterestre, una mai periculoasă ca cealaltă:



Space Invader 1

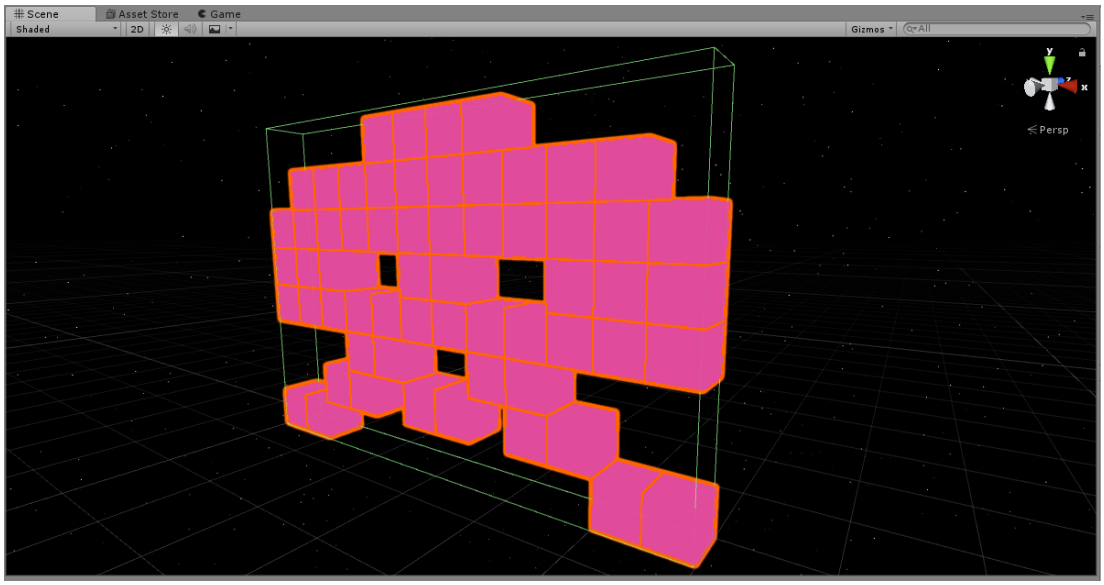


Space Invader 2

Cei doi inamici împărtășesc aceleași componente și deci, aceleași proprietăți, singurele diferențe constând în aspect și în viteza cu care se deplasează proiectilele.

Dintre componente inamicului, sunt de menționat:

- **Collider**, reprezentat de chenarul ce înconjoară inamicul și care are rolul de a interacționa cu proiectilul lansat de jucător:



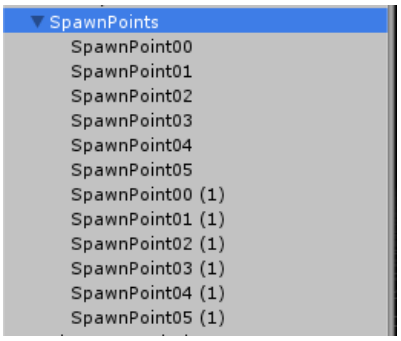
- **Space Invader Behaviour** reprezintă script-ul responsabil pentru comportamentul inamicului.

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  public class SpaceInvaderBehave1 : MonoBehaviour
5.  {
6.      public AudioSource diesound;   //Variabilei "diesound" ii vom atribui sunetul produs
                                         atunci cand inamicul este distrus;
7.      public GameObject bulletPrefab1; //Variabila "bulletPrefab" reprezinta entitatea caruia
          ii                               vom putea atribui obiectul 3d al proectilului,
                                           care va aparea in scena;
8.      public Transform enemyGun;  //Variabila "enemyGun" este de tip Transform, adica reprezin
          ta                          locatia din care proiectilul va fi lansat;
9.      public int rate; //Rate reprezinta frecventa cu care inamicul lanseaza proiectilele; deo
          arece        este de tip public, ea va putea fi modificata din editor;
10.     public float bulletSpeed; //Variabila "bulletspeed" reprezinta viteza cu care proiectilu
          l se                      va deplasa in scena; deoarece este de tip public, ea va
11.                                 putea fi modificata din editor;
12.     private int ok=1;  //Cu variabila "ok" vom verifica daca inamicul este distrus sau nu;
13.
14.     void Start ()
15.     {
16.         diesound = GetComponent<AudioSource> ();
17.     }
18.
19.
```

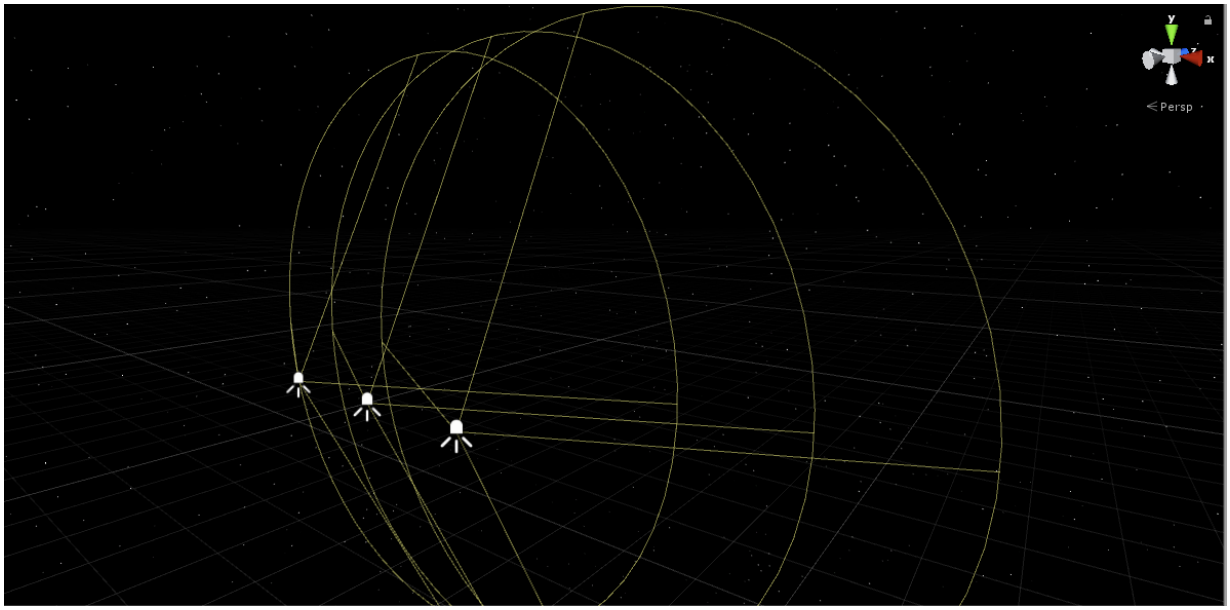
```
20.     void Update()
21.     {
22.         if (ok == 1) {
23.             StartCoroutine (EnemyShooter ());
24.         //Daca inamicul nu este mort, atunci el va lansa proiectile cu frecventa "rate";
25.           acest lucru este realizat cu ajutorul functiei EnemyShooter();
26.         }
27.     }
28.
29.     public IEnumerator EnemyShooter()
30.       //Aceasta functie este identica cu cea din PlayerShooter, singura diferenta fiind ca e
          a nu este apelata la apasarea unui buton, ci cu frecventa "rate" si cat timp inamicul
          traieste;
31.     {
32.         ok = 0;
33.         yield return new WaitForSeconds(rate);
34.         Debug.Log ("Enemy shot");
35.         GameObject bullet = Instantiate (bulletPrefab1, enemyGun.position, enemyGun.rotation
          , transform);
36.         bullet.transform.Rotate(Vector3.right, 90f);
37.         bullet.GetComponent<Rigidbody> ().AddForce (enemyGun.forward * bulletSpeed, ForceMod
          e.Acceleration);
38.         ok = 1;
39.     }
40.
41.
42.     public void OnCollisionEnter (Collision collision) //Aceasta functie este responsabila p
          entru                                              distrugerea inamicului;
43.     {
44.         if (collision.gameObject.tag == "Bullet") //astfel, daca inamicul intra in coliziune
           cu                                        proectilul jucatorului, care are tag-
          ul "Bullet"
45.         {
46.             diesound.Play ();            //se va produce un sunet
47.             StartCoroutine(KillEnemy());
48.              //se va apela functia KillEnemy, care distruge inamicul;
                 acest apel nu este unul simplu, ci prin metoda StartCoroutine, care asigura
                 faptul ca mai intai se va produce sunetul, si apoi va fi distrus inamicul;
49.              //daca am fi folosit un apel simpu, inamicul tot ar fi fost ditrus, insa fara p
                 roducerea sunetului;
50.         }
51.     }
52.
53.     public IEnumerator KillEnemy()
54.     {
55.         yield return new WaitForSeconds(0.2f);
56.
          //WaitForSeconds reprezinta intervalul de tip intre producerea sunetului si distrugere
          a
57.           inamicului;
58.         Destroy(gameObject);
59.     }
60. }
```

- **Audio Source,** adică sursa audio a sunetului produs de inamic atunci când este distrus:

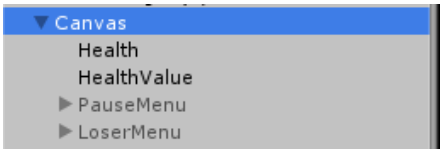
- **Spawn Points,** reprezintă cele 12 puncte plasate în scenă, în care se vor genera inamicii.



- **Player Boundaries** reprezintă cele 2 collidere care sunt folosite pentru a delimita spațiului de joc în care nava se poate deplasa.
- **Lighting,** așa cu sugerează și numele, reprezită obiectul care include sursele de lumină din scenă. În cazul de față, avem 3 surse de lumină, plasate în spatele navei, care emit lumină după forma cu muchiile galbene:



- **Canvas.** Ca și în cazul scenei meniului principal, canvas-ul din această scenă conține textul care apare permanent în stânga sus, în care este indicat nivelul vieții, dar și cele doua meniuri, de pauză și de game over.



  - Textul care ne indică starea vieții este alcătuit din 2 componente: *Health* și *HealthValue* (care se modifică odată cu viața navei)

- **Pause Menu** reprezintă meniul de pauză, care poate fi accesat în orice moment al jocului. El are butoane de *resume* (reluare a jocului), de întoarcere spre meniul principal, și de *quit*. Aspectul, butoanele și felul în care acestea interacționează cu mouse-ul sunt identice cu cele de la meniul principal.

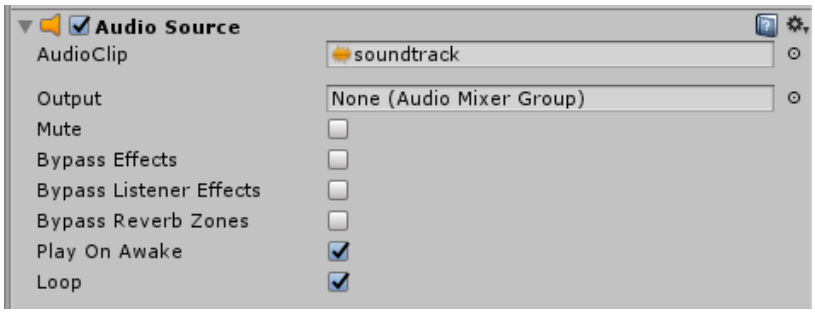
- **Game Over Menu,** este foarte similar cu *meniul de pauză*, doar că în plus acesta afișează textul "GAME OVER" în partea de sus a ecranului și include un buton de restart.

- **Pause Menu (Script)** este respnsabil cu funcționalitatea ambelor meniuri din această scenă

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.SceneManagement;
5.
6.  public class PauseMenu : MonoBehaviour {
7.      public static bool GameIsPaused = false; //Aceasta variabila va fi folosita pentru a put
         ea
           verifica daca jocul este in pauza sa nu;
8.      public GameObject pauseMenuUi; //Acestui obiect ii vom atribui meniul de pauza in sine;

9.
10.     void Update ()
11.     {
12.         if (Input.GetKeyDown (KeyCode.Escape))
13.      //Pentru intrarea, dar si iesirea din meniul de pauza, folosim tasta Escape;
14.         {
15.             if (GameIsPaused)
16.
         //Daca deja ne aflam in meniul de pauza, atunci la apasarea tastei escape, vom iesi din
          meniu;
17.             {
18.                 Resume();
19.             }
20.             else                 //Daca nu, vom intra in meniul de pauza;
21.             {
22.                 Pause();
23.             }
24.         }
25.     }
26.
27.     public void Resume() //Aceasta functie este responsabila pentru parasirea meniului de pa
         uza;
28.     {
29.         pauseMenuUi.SetActive (false); //Astfel, meniului de pauza va disparea de pe ecran;

30.         Time.timeScale = 1f; //Jocul va iesi din repaus;
31.         GameIsPaused = false;
32.     }
33.
34.     void Pause() //Aceasta functie este responsabila pentru accesarea meniului de pauza;
35.     {
36.         pauseMenuUi.SetActive (true); //Astfel, pe ecran va aparea meniul de pauza in sine;

37.         Time.timeScale = 0f; //Jocul va intra in repaus;
38.         GameIsPaused = true;
39.     }
40.
41.     public void LoadMenu()
42.      //Aceasta functie asigura functionalitatea butolului de accesare a meniului principal;

43.     {
44.         Time.timeScale = 1f;
45.         SceneManager.LoadScene ("Menu");
46.         Debug.Log ("Loading Main Menu");
47.     }
48.
```

```
49.     public void QuitGame()
50.      //Aceasta funtie asigura functionalitatea butonului de iesire din joc;
51.     {
52.         Debug.Log ("QUIT");
53.         Application.Quit ();
54.     }
55.
56.
57.     public void Restart() //Aceasta funtie asigura functionalitatea butonului de restart;
58.     {
59.         Time.timeScale = 1f;
60.         SceneManager.LoadScene ("Game");
61.     }
62. }
```

- **Sound Track** reprezintă obiectul care conține muzica ce se aude pe fundal în timpul jocului. Ea a fost preluată din jocul original pentru a recrea o atmosferă autentică.



De menționat aici sunt opțiunea bifată *Play On Awake*, care înseamnă că melodia de pe fundal se va auzi odată cu încărcarea scenei jocului propriu-zis.

4. Bibliografie

- https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
- https://en.wikipedia.org/wiki/Unity_(game_engine)
- Documentația oficială Unity : https://docs.unity3d.com/Manual/index.html
- https://en.wikipedia.org/wiki/Space_Invaders
- Elemente preluate din jocul original (efecte de sunete): http://www.classicgaming.cc/classics/space-invaders/