CoderDojo Unity - Opdrachten

Hieronder vind je de instructies voor de Unity dojo. Om deze handleiding niet al te lang te maken wordt er voornamelijk uitgelegd wat je moet doen en veel minder waarom je het zo moet doen. Als je vragen hebt, stel ze dan tijdens de dojo aan je begeleiders.

Dit project is gebaseerd op een bestaande demo die je in de Asset Store van Unity kan vinden, genaamd Into The Space.

Source – https://github.com/dvhirtum/coderdojo-intothespace

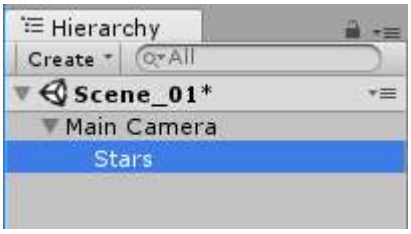
Unity - https://store.unity.com/download?ref=personal

Into the space assets - https://www.assetstore.unity3d.com/en/#!/content/20749

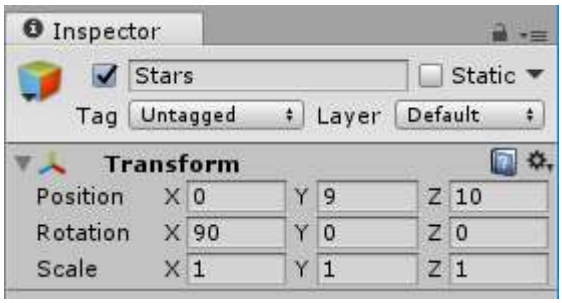
Opdracht 1 – De achtergrond

We gaan het spel in verschillende stappen opbouwen en om te beginnen gaan we iets doen aan de achtergrond. Als je het spel nu zou uitproberen, dan krijgen we alleen maar een lelijke donkerpaarse achtergrond te zien. Dat is beter dan zwart, want dat is helemaal saai, maar veel stelt het nog niet voor. Laten we als eerste eens wat sterren toevoegen!

1. Maak een leeg *GameObject* aan onder de “Main Camera” en geef het de naam “Stars”, zoals je in het plaatje hiernaast ziet.

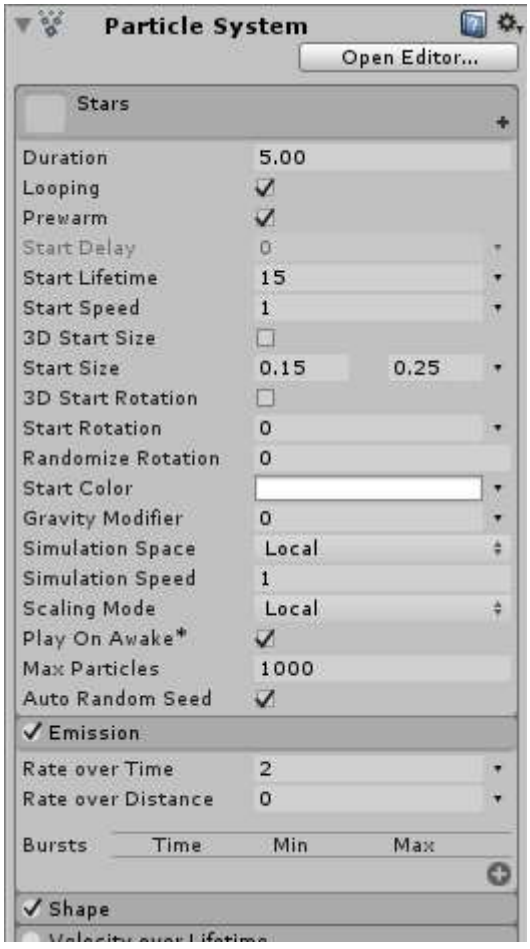


2. Stel de volgende waarden in voor de positie en rotatie:



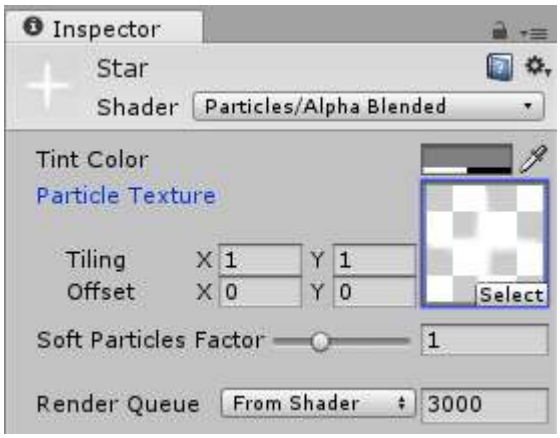
3. Voeg een *Particle System* toe en pas de volgende waarden aan (de rest kan onveranderd blijven):

```
Prewarm:           aangevinkt
Start Lifetime:    15
Start Speed:       1
Start Size:        0.15 – 0.25
Emission
  Rate over Time:  2
Renderer
  Order in Layer:  -100
  (niet afgebeeld)
```

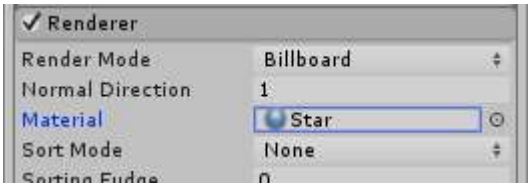


4. Maak een nieuw *Material* aan in de “Materials” folder met de naam “Star” en de volgende waarden:

   `Shader:` **`Particles/Alpha Blended`**
   `Particle Texture:` **`“star1”`**
   `(uit de Graphics/Effects folder)`



5. Koppel dit nieuwe materiaal aan het *Particle System* dat we hebben gemaakt onder “Renderer – Material”.



6. Sleep het *GameObject* “Stars” nu vanuit de *Hierarchy* naar de “Prefabs” folder. Dit zorgt ervoor dat we dit *GameObject* makkelijk kunnen hergebruiken.

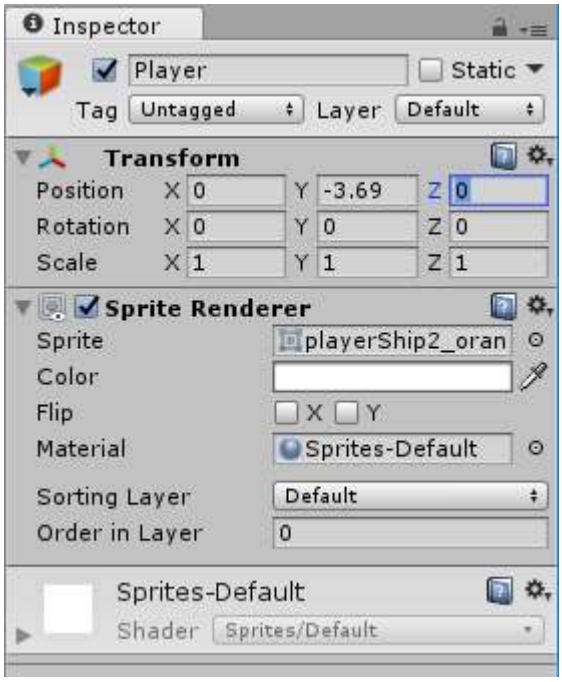
7. Zoals je ziet hebben we al een *GameObject* in de “Prefabs” folder, genaamd “SmallRocks”. Dit doet bijna hetzelfde als ons eigen “Stars” maar nu met stenen in plaats van sterren. Sleep “SmallRocks” naar de “Main Camera” zodat het onder “Stars” komt te staan.

8. Probeer het spel uit. Nu hebben we een mooie bewegende achtergrond, alleen kunnen we zelf nog niet veel doen. Dat wordt het onderwerp van de volgende opdracht.

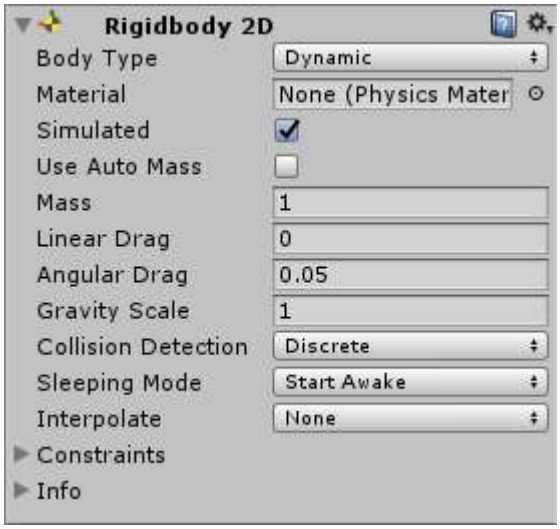
Opdracht 2 – De speler

Als tweede stap gaan we het ruimteschip van de speler bouwen. Na deze opdracht kunnen we met ons eigen schip rondvliegen in de achtergrond die we al hebben.

1. Zoek in de folder “Graphics\Player” het plaatje op met de naam “playerShip2_orange” en sleep het naar de scene. Dit maakt er een *GameObject* van met een *Transform* en een *Sprite Renderer*. Pas de naam aan naar “Player” in vul de positie in zoals in het plaatje hiernaast.

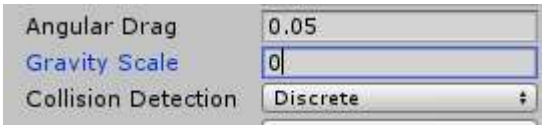


2. Voeg een *RigidBody2D* toe. Dit is een component die er voor zorgt dat we de snelheid en acceleratie een *GameObject* kunnen regelen.

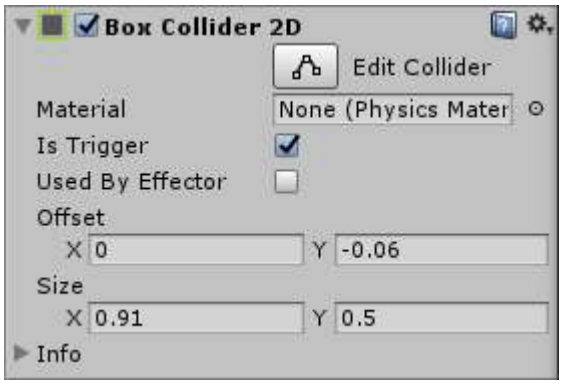


3. Start het spel nu, wat gebeurt er?

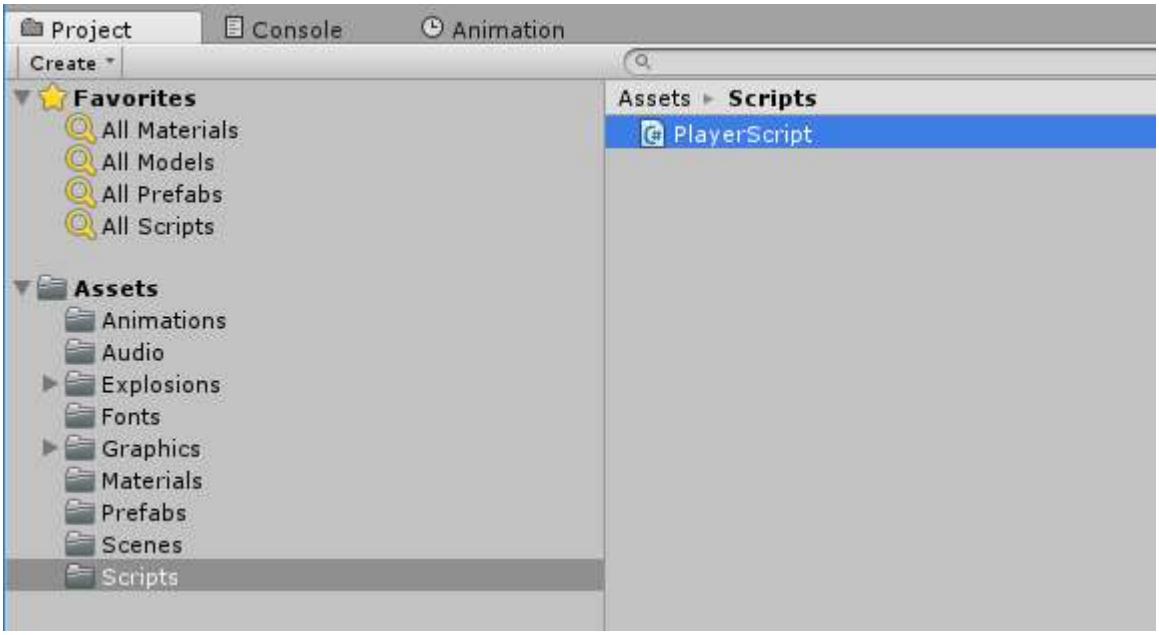
4. Naast snelheid regelt een *RigidBody2D* ook dingen als zwaartekracht voor ons. Dat hebben we in een ruimtespel niet nodig, dus zet *Gravity Scale* op 0.



5. Voeg ook een *BoxCollider2D* toe. Deze component (en alle andere *Collider* componenten) worden gebruikt om te bepalen of objecten in het spel elkaar raken. Zorg dat *Is Trigger* is aangevinkt en dat de *Offset* en *Size* zijn ingevuld zoals in de afbeelding hiernaast.



6. Nu moeten we er voor gaan zorgen dat we het ruimteschip kunnen besturen. Om dat te doen hebben we een script nodig. Maak een nieuwe folder “Scripts” aan onder “Assets” en voeg daar een *C# Script* aan toe. Noem dit script “PlayerScript”. Als alles goed is gegaan ziet je project er uit zoals in dit plaatje:



7. Dubbelklik op “PlayerScript”. Dit opent het script in de editor die in Unity is ingesteld. Vervang de inhoud die Unity automatisch heeft gemaakt door de volgende code:

```
using UnityEngine;

public class PlayerScript : MonoBehaviour
{
    public float Speed;

    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector2 movement = new Vector2(moveHorizontal, moveVertical);

        Rigidbody2D rigidBody = GetComponent<Rigidbody2D>();
        rigidBody.velocity = movement * Speed;

        rigidBody.position = new Vector2
        (
            Mathf.Clamp(rigidBody.position.x, -3, 3),
            Mathf.Clamp(rigidBody.position.y, -4.5f, 2)
        );
    }
}
```

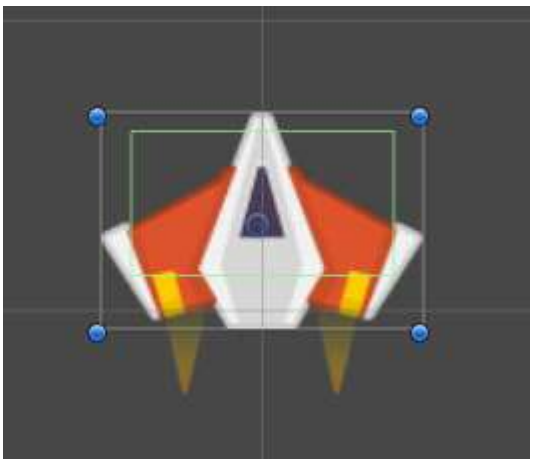
8. Koppel het script aan het “Player” *GameObject*. Nu kun je een nieuwe waarde invullen, namelijk *Speed*. Vul daar de waarde 6 in.

9. Probeer het spel uit. Nu kun je het ruimteschip besturen met de pijltjestoetsen.

10. Voordat we verder gaan met het maken van tegenstanders, gaan we ons ruimteschip nog een beetje verfraaien. Sleep het plaatje “fire13” vanuit de “Graphics\Effects” folder twee keer naar “Player” in de *Hierarchy*. Noem het eerste *GameObject* wat hierdoor aangemaakt wordt “MotorFireLeft” en de tweede “MotorFireRight”.

11. Stel de positie van de twee MotorFire-objecten als volgt in. Als het goed is ziet je ruimteschip er dan uit zoals het plaatje hiernaast.

```
MotorFireLeft:
X = -0.27       en Y = -0.431715
MotorFireRight:
X =  0.2565265 en Y = -0.431715
```



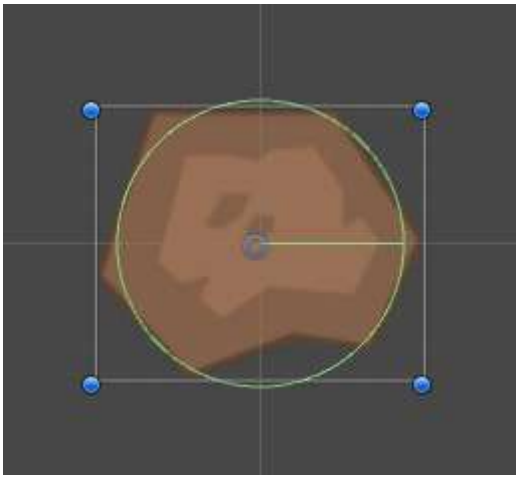
12. Koppel de *Animation Controller* “MotorFireLeft” uit de “Animations” folder aan het *GameObject* “MotorFireLeft”. Doe hetzelfde voor “MotorFireRight”. Beide *GameObjecten* zien er daarna in de *Inspector* uit zoals hiernaast: (het plaatje geeft “MotorFireRight” weer, de waarden van “MotorFireLeft” zijn uiteraard iets anders.)

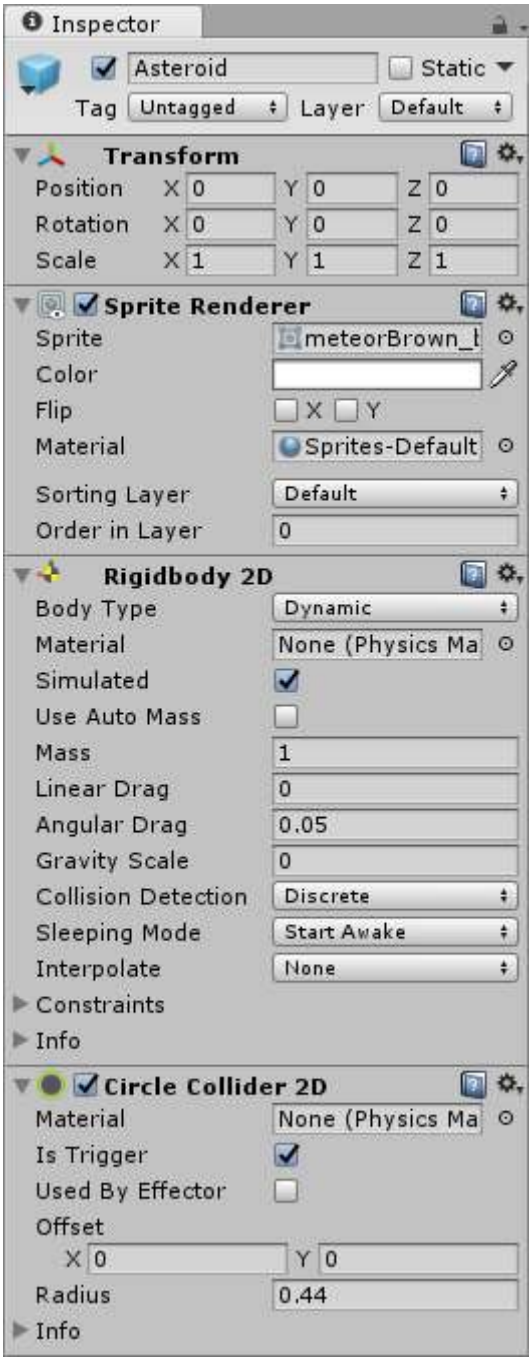
    Dit animeert de motoruitlaten, door elke 0.1 seconden het plaatje te verwisselen.

Opdracht 3 – De tegenstanders

Nu we zelf rond kunnen vliegen wordt het tijd voor wat tegenstanders. Daarvoor gaan we een paar nieuwe *Prefabs* maken. De procedure hiervoor is vergelijkbaar als de *Prefabs* die we al eerder hebben gemaakt en omdat we er vier achter elkaar maken, hoeven we telkens maar een paar waardes aan te passen.

1. Sleep als eerste “meteorBrown_big1” van de folder “Graphics/Rocks” naar de *Hierarchy*. Hernoem het *GameObject* “Asteroid”. Voeg een *RigidBody2D* (vergeet *Gravity Scale* = 0 niet) en een *CircleCollider2D* toe (*Is Trigger* aangevinkt). Stel de *Radius* van de *CircleCollider2D* in op 0.44. Als alles goed gaat ziet je “Asteroid” *GameObject* er nu zo uit:





2. Maak een nieuw script in de “Scripts” folder en noem deze “AsteroidScript”. Vervang de inhoud door de volgende code:

```
using UnityEngine;

public class AsteroidScript : MonoBehaviour
{
    public float MinTumble;
    public float MaxTumble;
    public float Speed;

    void Start ()
    {
        Rigidbody2D rigidBody = GetComponent<Rigidbody2D>();
        rigidBody.angularVelocity = Random.Range(MinTumble, MaxTumble);
        rigidBody.velocity = -1 * transform.up * Speed;
    }
}
```

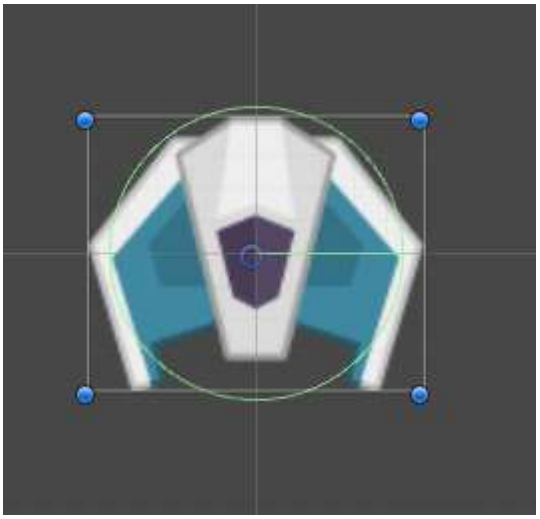
Hiermee kunnen we de snelheid van de “Asteroid” instellen en ook hoe snel hij om zijn as draait (*Tumble*).

3. Voegt het nieuwe script aan het *GameObject* toe en stel de volgende waarden in:


```
Min Tumble = 7
Max Tumble = 12
Speed      = 2
```

4. Sleep het *GameObject* naar de “Prefabs” folder, maar gooi hem nog niet weg uit de *Hierarchy*.

5. Verander de naam van het *GameObject* in de *Hierarchy* (dus niet die in de “Prefabs” folder) in “EnemyBlue”. Verander *Sprite* onder *Sprite Renderer* in “enemyBlue3”. Verander *Radius* onder *Circle Collider 2D* in 0.45.



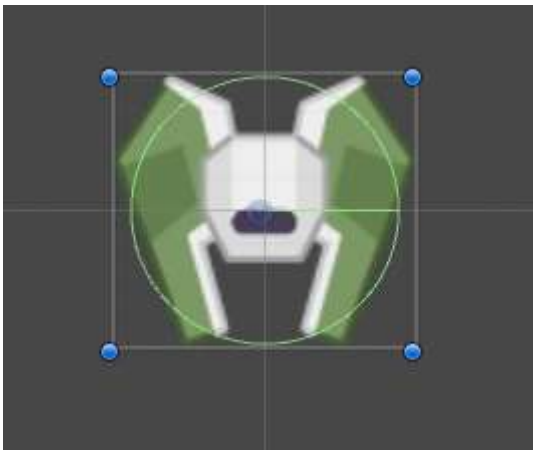
6. Maak een nieuw script in de “Scripts” folder en noem deze “EnemyScript”. Vervang de inhoud door:

```
using UnityEngine;

public class EnemyScript : MonoBehaviour
{
    public float Speed;

    void Start ()
    {
        GetComponent<Rigidbody2D>().velocity = -1 * transform.up * Speed;
    }
}
```

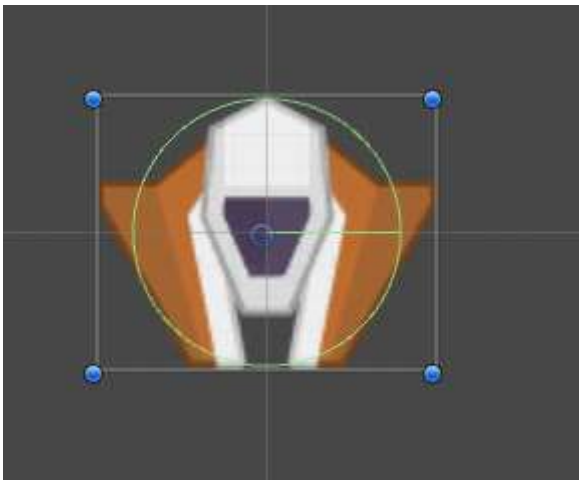
Dit regelt de snelheid, net als bij “AsteroidScript”.

7. Koppel het nieuwe script aan het *GameObject* en stel *Speed* in op 1. Hierna kun je het *GameObject* naar de “Prefabs” folder slepen.

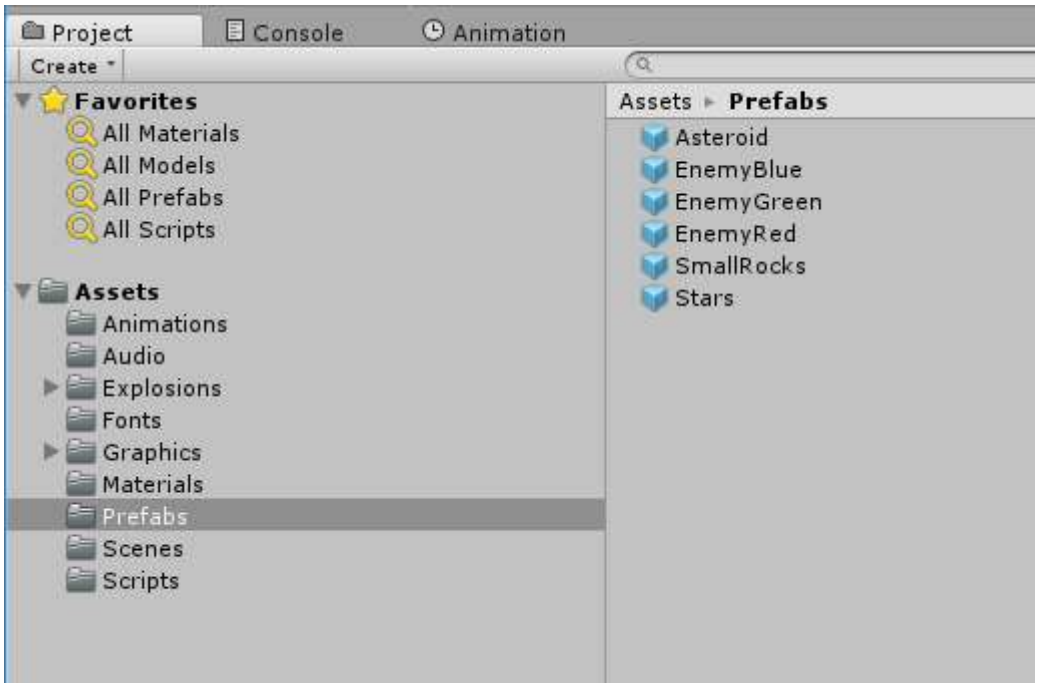
8. Voor de laatste twee *Prefabs* hoef je nu alleen nog maar een paar waarden aan te passen. Hernoem het *GameObject* (in de *Hierarchy*) naar “EnemyGreen”. Verander *Sprite* in “enemyGreen1”. Verander *Radius* in 0.41 en *Speed* in 2.5. Sleep het gewijzigde *GameObject* naar de “Prefabs” folder.



9. Laatste tegenstander! Verander de naam in “EnemyRed”, de *Sprite* in “enemyRed2” en *Speed* in 3 en sleep het naar de “Prefabs” folder. Nu kun je het *GameObject* uit de *Hierarchy* verwijderen.



10. Als het goed is heb je nu 6 *Prefabs*, “Stars” en “SmallRocks” uit Opdracht 1 en de 4 tegenstanders die we net hebben gemaakt:

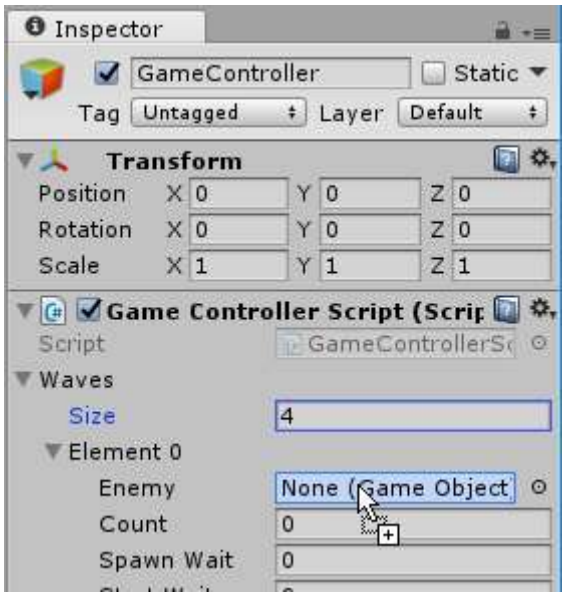


11. Nu moeten we nog iets verzinnen om de tegenstanders te laten verschijnen. Hier hebben we een script voor nodig, maar om te zorgen dat een script wordt uitgevoerd hebben we een *GameObject* nodig. Voeg daarom een nieuw leeg *GameObject* toe aan de *Hierarchy* en noem het “GameController”.

12. Maak een nieuw script “GameControllerScript” in de “Scripts” folder, met de volgende inhoud:

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

[System.Serializable]
public class Wave
{
    public GameObject Enemy;
    public int Count;
    public float SpawnWait;
    public float StartWait;
    public float WaveWait;
}

public class GameControllerScript : MonoBehaviour
{
    public Wave[] Waves;

    void Start ()
    {
        foreach (Wave wave in Waves)
        {
            StartCoroutine(spawnWaves(wave));
        }
    }

    IEnumerator spawnWaves(Wave wave)
    {
        yield return new WaitForSeconds (wave.StartWait);

        while (true)
        {
            for (int i = 0; i < wave.Count; i++)
            {
                Vector2 spawnPosition = new Vector2 (Random.Range (-3, 3), 6);
                Quaternion spawnRotation = Quaternion.identity;
                Instantiate (wave.Enemy, spawnPosition, spawnRotation);
                yield return new WaitForSeconds (wave.SpawnWait);
            }
            yield return new WaitForSeconds (wave.WaveWait);
        }
    }
}
```

13. Koppel “GameControllerScript” aan “GameController” en vul de volgende waarden in:




```
Size          = 4
Element 0
  Enemy       = Asteroid
  Count       = 4
  Spawn Wait  = 1
  Start Wait  = 1
  Wave Wait   = 5
Element 1
  Enemy       = EnemyBlue
  Count       = 3
  Spawn Wait  = 1
  Start Wait  = 2
  Wave Wait   = 6
Element 2
  Enemy       = EnemyGreen
  Count       = 3
  Spawn Wait  = 2
  Start Wait  = 8
  Wave Wait   = 8
Element 3
  Enemy       = EnemyRed
  Count       = 2
  Spawn Wait  = 2
  Start Wait  = 14
  Wave Wait   = 12
```

14. Probeer het spel uit. Zie je de tegenstanders verschijnen? We hebben alleen nog een probleem met de huidige versie. Als je tijdens het spelen naar de *Hierarchy* kijkt, dan valt je misschien op dat er steeds meer objecten bij komen. Dit komt omdat we de tegenstanders die uit beeld verdwijnen nooit opruimen. Als we hier niets aan doen, dan zal op een gegeven moment het geheugen van de computer vollopen en crasht het spel (of de computer)!

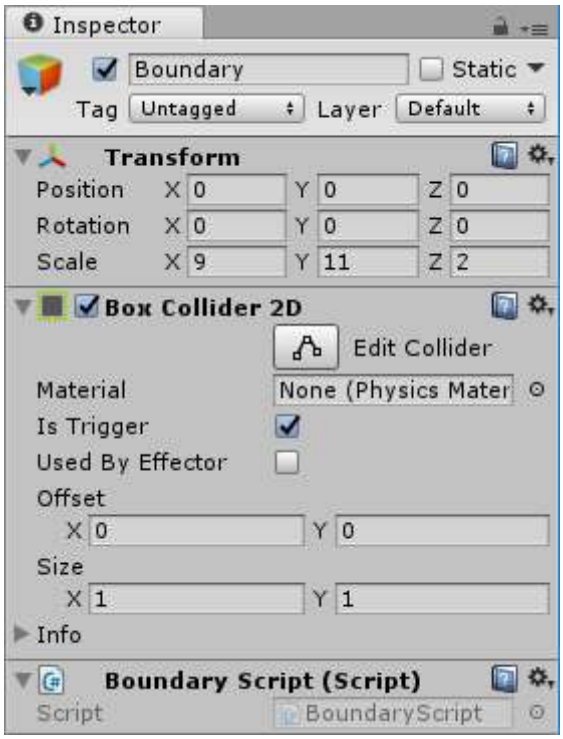
15. Om dit probleem op te lossen gaan we een nieuw *GameObject* maken met de naam “Boundary”. Zorg dat de positie-waarden allemaal op 0 staan en vul de volgende waarden in voor *Scale*:

    X = **9**
    Y = **11**
    Z = **2**

16. Maak een nieuw script “BoundaryScript” met de volgende inhoud:

```
using UnityEngine;

public class BoundaryScript : MonoBehaviour
{
    void OnTriggerExit2D(Collider2D other)
    {
        Destroy(other.gameObject);
    }
}
```

17. Koppel het script aan het “Boundary” object, wat er daarna als het goed is zo uit ziet:

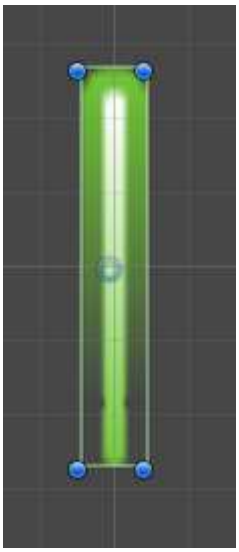


18. Als je het spel nu weer uitprobeert zal je zien dat de tegenstanders ook uit de *Hierarchy* verdwijnen als ze uit beeld vliegen.

Opdracht 4 – Lasers en explosies

Al die tegenstanders die op je af komen zijn leuk, maar erg bedreigend zijn ze niet. Ze vliegen gewoon door je heen! En zou het niet leuk zijn als we ze neer kunnen schieten in plaats van alleen maar ontwijken? Dat zijn de onderwerpen van deze vierde opdracht.

1. We beginnen met schieten. Daarvoor hebben we weer een paar *Prefabs* nodig zodat we makkelijk meerdere laser-schoten kunnen maken. De procedure daarvoor is inmiddels bekend: Sleep "laserGreen1" (uit de folder "Graphics/Lasers") naar de *Hierarchy*. Hernoem het *GameObject* naar "PlayerLaserBolt". Voeg een *RigidBody2D* (*Gravity Scale* = 0) en een *BoxCollider2D* (*Is Trigger* aangevinkt) toe.



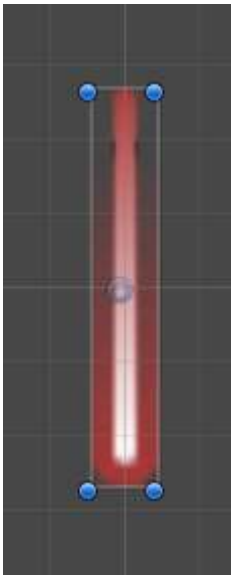
2. Voeg een script "LaserBoltScript" toe en koppel het aan het *GameObject*. De inhoud van het script is:

```
using UnityEngine;

public class LaserBoltScript : MonoBehaviour
{
    public float Speed;
    public bool ShootUp;

    void Start ()
    {
        Rigidbody2D rigidBody = GetComponent<Rigidbody2D>();
        rigidBody.velocity = (ShootUp ? 1 : -1) * transform.up * Speed;
    }
}
```

3. Vul voor *Speed* de waarde 10 in en zorg dat *Shoot Up* is aangevinkt.

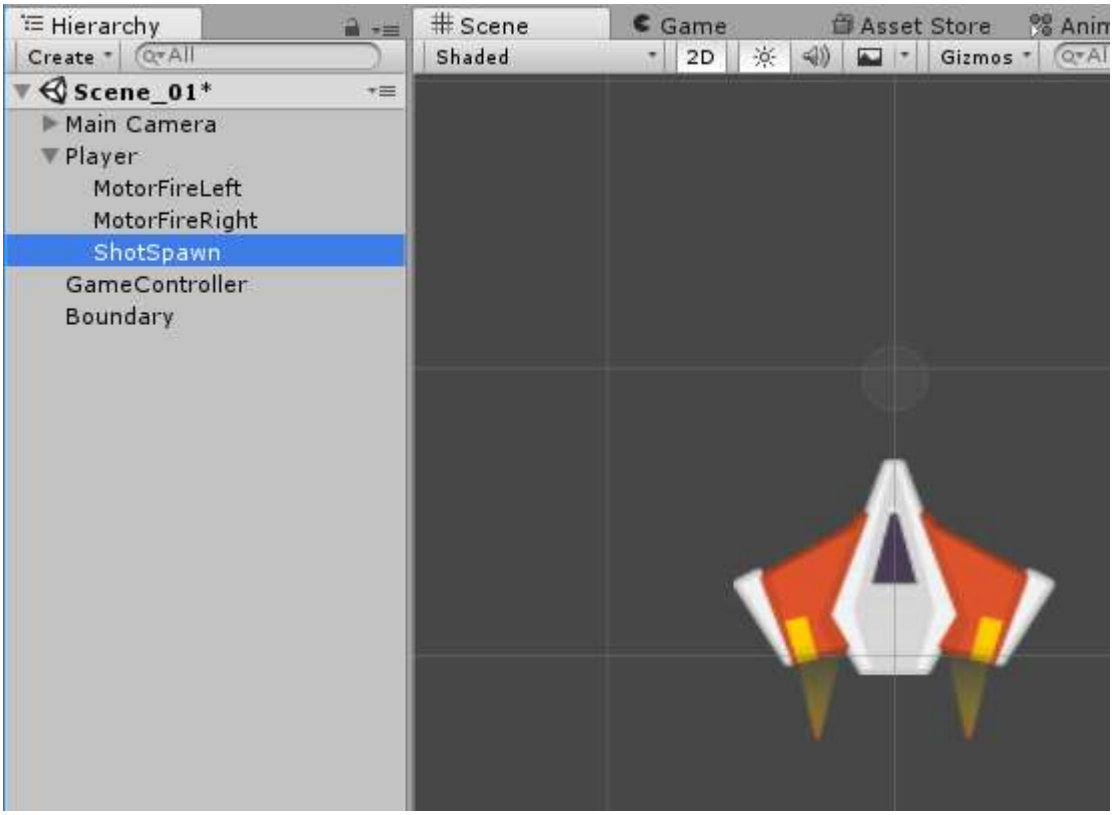
4. Sleep het *GameObject* naar de "Prefabs" folder.

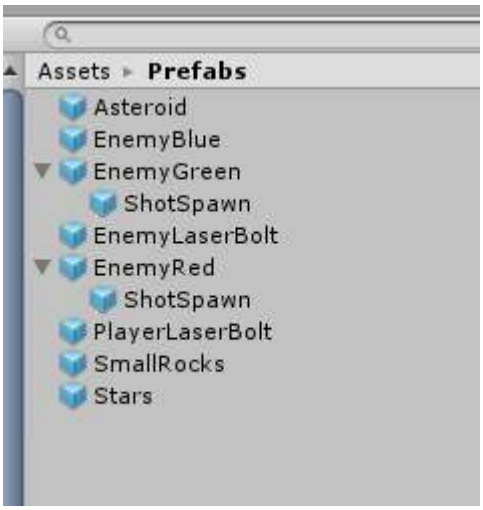
5. Pas de naam van het *GameObject* (in de *Hierarchy*) aan naar "EnemyLaserBolt". Het enige dat je hoeft te wijzigen zijn *Sprite* ("laserRed01"), *Speed* (waarde 5) en *Shoot Up* (uitgevinkt). Sleep hierna het *GameObject* opnieuw naar de "Prefabs" folder.



6. Nu kun je het *GameObject* verwijderen uit de *Hierarchy*.

7. De *GameObjecten* die kunnen schieten; “Player”, “EnemyGreen” en “EnemyRed” hebben een punt nodig waar vandaan de laserbolt zal vertrekken. Dit zouden we in code kunnen oplossen, maar het is netter om hiervoor een *GameObject* te gebruiken dat een vaste positie heeft ten opzichte van zijn *parent*. Voeg dus een nieuw *GameObject* toe aan “Player”, “EnemyGreen” en “EnemyRed” en noem het nieuwe object in alle drie de gevallen “ShotSpawn”. Als het goed gaat ziet het er uit zoals in de volgende twee plaatjes:





8. Vul de volgende waardes in voor de positie van de “ShotSpawn”-objecten:

```
Player
X = 0, Y =  0.6555326
EnemyGreen
X = 0, Y = -0.438956
EnemyRed
X = 0, Y = -0.438956
```

9. Het schieten van laserbolts gaan we regelen met een script. Maar dus een nieuw script aan met de naam “ShootScript” met de volgende inhoud:

```
using UnityEngine;

public class ShootScript : MonoBehaviour
{
    public float FireRate = 0.5f;
    public GameObject LaserBolt;
    public Transform ShotSpawn;

    private float nextFire = 0.0f;

    void Update()
    {
        if (Time.time > nextFire)
        {
            nextFire = Time.time + FireRate;
            Instantiate(LaserBolt, ShotSpawn.position, ShotSpawn.rotation);
        }
    }
}
```

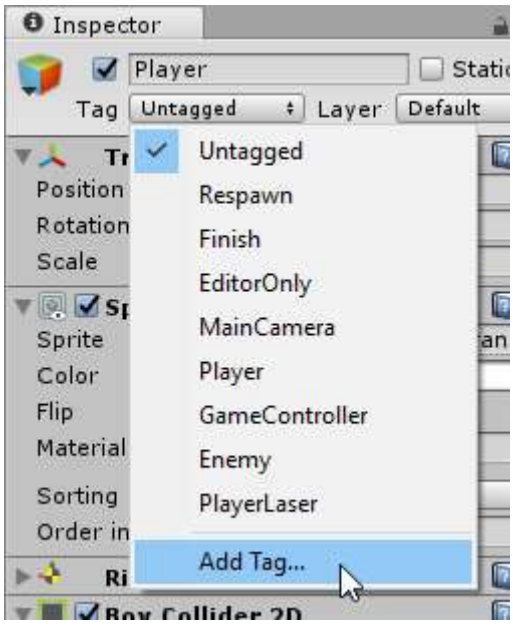
10. Voeg “ShootScript” toe aan “Player”, “EnemyGreen” en “EnemyRed” en stel de volgende waardes in:

```
Player
Fire Rate   = 0.3
Laser Bolt  = PlayerLaserBolt
Shot Spawn  = ShotSpawn
EnemyGreen
Fire Rate   = 2
Laser Bolt  = EnemyLaserBolt
Shot Spawn  = ShotSpawn
EnemyRed
Fire Rate   = 2
Laser Bolt  = EnemyLaserBolt
Shot Spawn  = ShotSpawn
```

Zorg er voor dat je het juiste “ShotSpawn”-object koppelt aan het juiste script.

11. Probeer het spel uit. Zowel jouw ruimteschip als dat van de tegenstanders vuurt lasers af. Er gebeurt alleen nog niets als een laser iets raakt, dus daar moeten we snel wat aan doen!

12. Om te weten wie wat raakt is het handig om *Tags* aan sommige objecten te koppelen. Hiervoor gaan we eerst een aantal *Tags* maken. Selecteer een willekeurig *GameObject* en klik op het menuutje naast *Tag*. Kies voor *Add Tag…* en voeg vervolgens 3 *Tags* toe: “Enemy”, “PlayerLaser” en “EnemyLaser”.



13. Koppel de *Tag* “Enemy” aan alle vier de tegenstander-*Prefabs*. Koppel de *Tag* “EnemyLaser” aan de “EnemyLaserBolt”-*Prefab* en de *Tag* “PlayerLaser” aan de “PlayerLaserBolt”-*Prefab*. Dit doe je via hetzelfde menuutje waarmee je ook de *Tags* hebt aangemaakt.

14. Pas nu “PlayerScript” aan. Voeg onderstaande code toe:

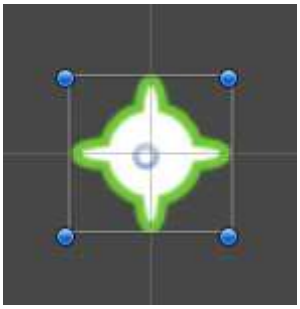
```
using UnityEngine;

public class PlayerScript : MonoBehaviour
{
    public float Speed;
    public GameObject Explosion;

    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector2 movement = new Vector2(moveHorizontal, moveVertical);

        Rigidbody2D rigidBody = GetComponent<Rigidbody2D>();
        rigidBody.velocity = movement * Speed;

        rigidBody.position = new Vector2
        (
            Mathf.Clamp(rigidBody.position.x, -3, 3),
            Mathf.Clamp(rigidBody.position.y, -4.5f, 2)
        );
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Enemy" || other.tag == "EnemyLaser")
        {
            Instantiate (Explosion, transform.position , transform.rotation);
            Destroy(gameObject);
        }
    }
}
```

15. Dit voegt een nieuwe property toe aan “Player”, namelijk *Explosion*. Vul deze met de *Prefab* “explosion_player” uit de folder “Explosions/Prefabs”.

16. Als je nu het spel uitprobeert en ergens tegenaan vliegt, dan volgt er een mooie explosie! Alleen kan je zelf nog niets terug doen. Flauw!

17. Als jij ergens tegenaan vliegt, dan ben je meteen af. Dat maakt het spel wel zo spannend. Maar als dat ook voor alle tegenstanders zou gelden, dan zou het spel veel te makkelijk en dus saai worden. Dus we moeten een mogelijkheid maken dat tegenstanders meer dan eens geraakt kunnen worden voordat ze kapot gaan. Maar voordat we dat doen gaan we eerst nog een nieuwe *Prefab* maken.

18. Sleep “laserGreen14” van de folder “Graphics/Lasers” naar de *Hierarchy*. Noem de nieuwe *GameObject* “LaserHit”.



19. Voeg een nieuwe script “DestoryScript” toe en koppel het aan het *GameObject* “LaserHit”. De inhoud van “DestroyScript” is:

```
using UnityEngine;

public class DestroyScript : MonoBehaviour
{
    public float Delay;

    void Start ()
    {
        Destroy(gameObject, Delay);
    }
}
```

20. Vul bij *Delay* in het “LaserHit” *GameObject* de waarde 0.05 in.

21. Sleep “LaserHit” naar de “Prefabs”-folder. Nu hebben we een object dat even zichtbaar wordt en 0.05 seconden later weer verdwijnt. Dit kunnen we mooi gebruiken om aan te geven als een tegenstander geraakt wordt.

22. Tot slot hebben we nog een script nodig om bij te houden hoe vaak een tegenstander geraakt is en als het maximum is bereikt een explosie weergeeft. Maak hiervoor een nieuw script met de naam “LaserHitScript” en de volgende inhoud:

```
using UnityEngine;

public class LaserHitScript : MonoBehaviour
{
    public int Health;
    public GameObject LaserHit;
    public GameObject Explosion;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "PlayerLaser")
        {
            Instantiate (LaserHit, transform.position , transform.rotation);
            Destroy(other.gameObject);

            if (Health > 0)
                Health--;

            if (Health <= 0)
            {
                Instantiate (Explosion, transform.position , transform.rotation);

                Destroy(gameObject);
            }
        }
    }
}
```

23. Koppel “LaserHitScript” aan alle vier de tegenstander-*Prefabs*, met de volgende waarden:


```
Asteroid
Health    = 1
LaserHit  = LaserHit
Explosion = explosion_asteroid
EnemyBlue
Health    = 2
LaserHit  = LaserHit
Explosion = explosion_enemy
EnemyBlue
Health    = 2
LaserHit  = LaserHit
Explosion = explosion_enemy
EnemyBlue
Health    = 4
LaserHit  = LaserHit
Explosion = explosion_enemy
```

24. Nu kunnen jij en je tegenstanders elkaar naar hartenlust afmaken! Gezellig.

25. Voordat we verder gaan met de laatste opdracht moeten we nog een probleem oplossen. Net als eerder met de tegenstanders blijven explosies die zijn afgegaan eeuwig bestaan. Hoe kan dat nou? De “Boundary” ruimt toch alles op? Het probleem is dat de “Boundary” alles op ruimt dat uit beeld verdwijnt. Maar omdat explosies geen snelheid hebben zullen ze nooit uit

beeld verdwijnen. Ze zijn onzichtbaar, maar technisch gesproken blijven ze gewoon bestaan.

Gelukkig is hier een makkelijke oplossing voor: We hebben al een “DestroyScript” gemaakt voor de “LaserHit”-*Prefab*. Die kunnen we gewoon hergebruiken! Koppel dus het “DestroyScript” aan alle drie de explosion-*Prefabs* en stel de *Delay* in op 2 seconden.

Opdracht 5 – Losse eindjes

In de laatste opdracht gaan we het spel nog wat verfraaien door geluid, muziek en een score toe te voegen.

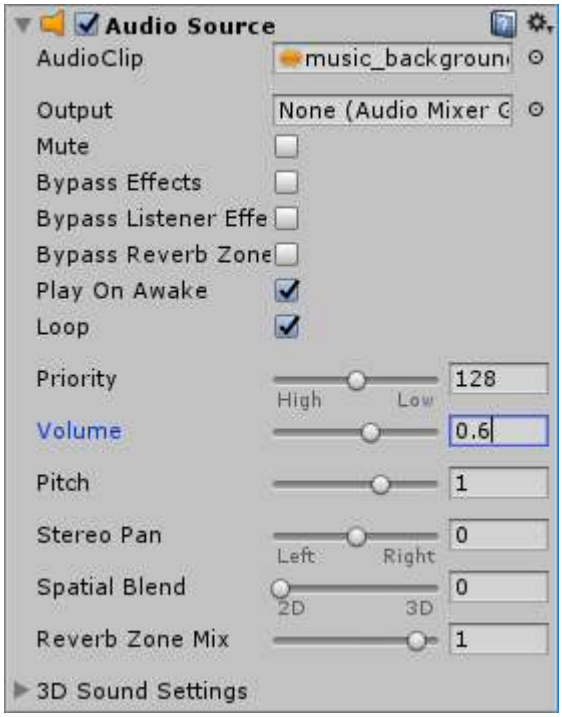
1. We beginnen met geluid. Misschien is het je opgevallen dat de explosies al geluid maken. Dat komt omdat de *Prefabs* al een *Audio Source* hebben. Maar daardoor valt het wel erg op dat het schieten van lasers geen geluid veroorzaakt. Dat is heel makkelijk opgelost: Sleep “weapon_player” uit de “Audio”-folder naar “Player” en “weapon_enemy” naar “EnemyGreen” en “EnemyRed”.

2. Zorg dat bij alle drie de nieuwe *Audio Sources* de optie *Play On Awake* is uitgevinkt en pas het *Volume* aan naar een waarde waardoor je oren niet spontaan van je hoofd vallen, bijvoorbeeld 0.5.

3. Omdat we *Play On Awake* hebben uitgezet zal het geluid niet automatisch te horen zijn. Daarom moeten we “ShootScript” aanpassen:

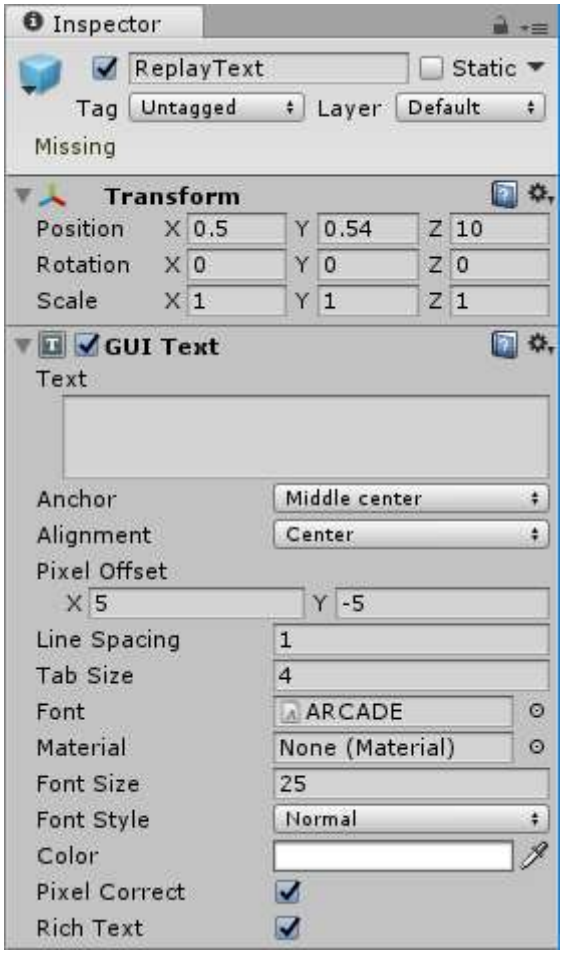
```
using UnityEngine;

public class ShootScript : MonoBehaviour
{
    public float FireRate = 0.5f;
    public GameObject LaserBolt;
    public Transform ShotSpawn;

    private float nextFire = 0.0f;

    void Update()
    {
        if (Time.time > nextFire)
        {
            nextFire = Time.time + FireRate;
            Instantiate(LaserBolt, ShotSpawn.position, ShotSpawn.rotation);
            GetComponent<AudioSource>().Play();
        }
    }
}
```

4. Om ook wat achtergrond muziek toe te voegen sleep je “music_background” uit de “Audio”-folder naar “GameController”. Zorg nu dat *Play On Awake* wel aangevinkt is, net als *Loop*. Stel *Volume* in op een waarde die je prettig vind. Hier is gekozen voor 0.6.



5. Het spel is nu bijna af, maar het zou nog leuk zijn als we kunnen zien wat onze score is. Daarvoor moeten we een paar dingen doen: Tekst toevoegen zodat we kunnen zien wat de score is en bijhouden hoeveel punten we al hebben gescoord. We beginnen met tekst toevoegen.

6. Voeg vier *GameObjecten* toe aan “Main Camera”. Hierdoor kunnen we makkelijk hun positie ten opzicht van de camera bepalen en zo zorgen dat ze altijd duidelijk in beeld zijn. Noem de *GameObjecten*: “ScoreText”, “GameOverText”, “FinalScoreText” en “ReplayText”. Voeg aan alle vier de objecten een *GUI Text* component toe en vul de volgende waarden in:



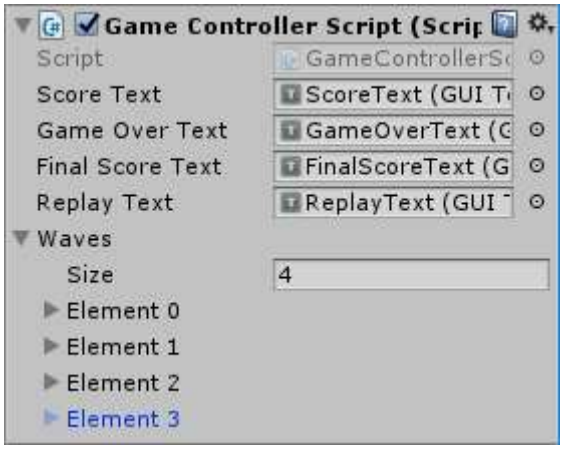

```
ScoreText
  Position: X = 0, Y = 1, Z = 10
  Pixel Offset: X = 5, Y = -5
  Font: ARCADE (Fonts folder)
  Font Size: 25
GameOverText
  Position: X = 0.5, Y = 0.65,
            Z = 10
  Anchor: Middle center
  Alignment: Center
  Pixel Offset: X = 5, Y = -5
  Font: ARCADE
  Font Size: 50
FinalScoreText
  Position: X = 0.5, Y = 0.59,
            Z = 10
  Anchor: Middle center
  Alignment: Center
  Pixel Offset: X = 5, Y = -5
  Font: ARCADE
  Font Size: 50
ReplayText
  Position: X = 0.5, Y = 0.54, Z = 10
  Anchor: Middle center
  Alignment: Center
  Pixel Offset: X = 5, Y = -5
  Font: ARCADE
  Font Size: 25
```

7. Nu moeten we “GameControllerScript” aanpassen om te bepalen wat we in de tekstvelden willen laten zien:

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

[System.Serializable]
public class Wave
{
    public GameObject Enemy;
    public int Count;
    public float SpawnWait;
    public float StartWait;
    public float WaveWait;
}

public class GameControllerScript : MonoBehaviour
{
    public GUIText ScoreText;
    public GUIText GameOverText;
    public GUIText FinalScoreText;
    public GUIText ReplayText;

    public Wave[] Waves;

    private int score;
    private bool gameOver;

    public void IncrementScore(int value)
    {
        score += value;
    }

    public void GameOver(){
        gameOver = true;
    }

    void Start ()
    {
        foreach (Wave wave in Waves)
        {
            StartCoroutine(spawnWaves(wave));
        }
    }
```

```
    void Update ()
    {
        if (Input.GetKey("r"))
        {
            SceneManager.LoadScene("Scene_01", LoadSceneMode.Single);
        }
    }

    void FixedUpdate()
    {
        ScoreText.text = "SCORE: " + score;

        if (gameOver == true)
        {
            GameOverText.text = "GAME OVER";
            FinalScoreText.text = "" + score;
            ReplayText.text = "PRESS R TO REPLAY";
        }
    }

    IEnumerator spawnWaves(Wave wave)
    {
        yield return new WaitForSeconds (wave.StartWait);

        while (true)
        {
            for (int i = 0; i < wave.Count; i++)
            {
                Vector2 spawnPosition = new Vector2 (Random.Range (-3, 3), 6);
                Quaternion spawnRotation = Quaternion.identity;
                Instantiate (wave.Enemy, spawnPosition, spawnRotation);
                yield return new WaitForSeconds (wave.SpawnWait);
            }
            yield return new WaitForSeconds (wave.WaveWait);
        }
    }
}
```

8. Koppel de tekstvelden die je aan de “Main Camera” hebt gehangen aan de nieuwe properties van “GameController”.



9. Nu kan “GameController” de teksten invullen, maar we moeten nog wel regelen dat hij dit ook op het juiste moment doet. Daarvoor moeten we de twee methoden “IncrementScore” en “GameOver” die we aan “GameControllerScript” hebben toegevoegd kunnen aanroepen vanuit andere scripts. Om dat te doen is het handig om een *Tag* aan “GameController” te

koppelen. Gelukkig is er al een *Tag* met exact dezelfde naam, dus koppel deze.

10. Pas nu “LaserHitScript” aan:

```
using UnityEngine;

public class LaserHitScript : MonoBehaviour
{
    public int Health;
    public int ScoreValue;
    public GameObject LaserHit;
    public GameObject Explosion;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "PlayerLaser")
        {
            Instantiate (LaserHit, transform.position , transform.rotation);
            Destroy(other.gameObject);

            if (Health > 0)
                Health--;

            if (Health <= 0)
            {
                Instantiate (Explosion, transform.position , transform.rotation);

                GameObject gameController = GameObject.FindWithTag("GameController");
                GameControllerScript script = gameController.GetComponent<GameControllerScript>();
                script.IncrementScore(ScoreValue);

                Destroy(gameObject);
            }
        }
    }
}
```

11. Deze aanpassing voegt een nieuwe property toe, *Score Value*. Geef deze de volgende waarden:

```
Asteroid:    10
EnemyBlue:   15
EnemyGreen:  25
EnemyRed:    40
```

12. Tot slot moeten we de “GameOver”-methode nog aanroepen vanuit “PlayerScript” als de speler een tegenstander raakt of tegen een laser aanloopt:

```
using UnityEngine;

public class PlayerScript : MonoBehaviour
{
    public float Speed;
    public GameObject Explosion;

    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");
        Vector2 movement = new Vector2(moveHorizontal, moveVertical);

        Rigidbody2D rigidBody = GetComponent<Rigidbody2D>();
        rigidBody.velocity = movement * Speed;

        rigidBody.position = new Vector2
        (
            Mathf.Clamp(rigidBody.position.x, -3, 3),
            Mathf.Clamp(rigidBody.position.y, -4.5f, 2)
        );
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Enemy" || other.tag == "EnemyLaser")
        {
            Instantiate (Explosion, transform.position , transform.rotation);

            GameObject gameController = GameObject.FindWithTag("GameController");
            GameControllerScript script = gameController.GetComponent<GameControllerScript>();
            script.GameOver();

            Destroy(gameObject);
        }
    }
}
```

13. Speel het spel. Als het goed is hoor je muziek, maken de lasers geluid als ze schieten en zie je links boven in het scherm je score. Je krijgt zelfs een mooi overzicht van je eindscore als je af gaat en een tip hoe je opnieuw kan spelen!