Technisch Verslag

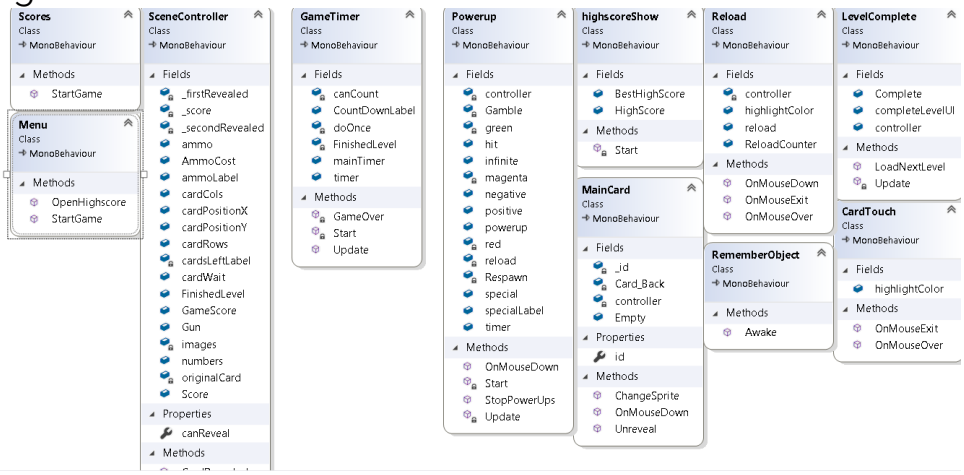
Bouke de Vries
Studentnummer: 1683542
Datum: 2-2-2019
Versie 1.1

Versiebeheer

Aanpassingen	Versie	Datum
Eerste document	1.0	5-11-2018
Tweede document met aanpassingen	1.1	2-2-2019

Inhoudsopgave

1 Klassendiagram



Figuur 1 Klassendiagram

In figuur 1 bevind zich het klassendiagram. Hierin Staan alle scripts beschreven met de bijbehorende methodes. De relaties zijn hier niet opgenomen gezien de relaties binnen Unity zelf worden afgehandeld.

CardTouch

Dit script word gebruikt om aan te geven welke kaart binnen het bord aangeraakt word. De interacties vinden zich plaats d.m.v. de muis. De muis triggert het event waardoor de methodes worden aangeroepen.

LevelComplete

Dit script heeft een methode genaamd LoadNextLevel() deze methode word aangeroepen wanneer de speler een level heeft voltooid. Wanneer een level voltooid is word een transitie aangeroepen en word aan het eind van de transitie de functie aangeroepen.

Reload

Het script hierachter vind plaats bij zowel de SceneController als de munitiekist zelf. Wanneer de speler interacteert met de muis zal het script een kogel bij de SceneController +1 bijoptellen.

Menu

Het script Menu word gebruikt voor de knoppen binnen deze game. Zowel als in de menu scene als het eind scene om het spel bij level 1 opnieuw te kunnen starten. Interactie vind plaats met de muis.

MainCard

Het script MainCard bepaalt welke kaart zojuist is geselecteerd en welke is omgelegd ook word binnen dit script vastgesteld wanneer de speler een kaart omdraaid de speler tegelijkertijd ook een kogel verliest. Zo roept het script ook de SceneController hiervoor aan. Maar gebruikt het ook de SceneController om te kijken of de combinatie van de kaarten juist zijn.

SceneController

Dit is het hart van het spel. Dit script bevat alle belangrijke waardes zoals aantal munitie en hoeveel combinaties met de kaarten nog gemaakt moeten worden voordat het level beëindigd is. Ook regelt dit script hoeveel kaarten op het bord verschijnen en wat de afstand van deze kaarten is, met daarnaast welke kaarten sprites komen hier in voor. Zo bevat het script voor het grootste gedeelte de game logica die plaats vind per level.

Scores

Het script Scores word gebruikt voor de knop binnen deze game. In het highscore menu word dit script gebruikt om een methode aan te roepen om terug te keren naar het menu.

GameTimer

Het script GameTimer bepaalt de hoeveelheid seconden binnen de game en wanneer de ronde voorbij is of niet. Deze logica zat eerst in de sceneController maar word nu apart uitgevoerd binnen dit script.

HighscoreShow

Dit script toont zowel de behaalde highscore in game als de hoogst mogelijke behaalde highscore gedurende het spel. Dit script houd de highscore bij van het spel.

RememberObject

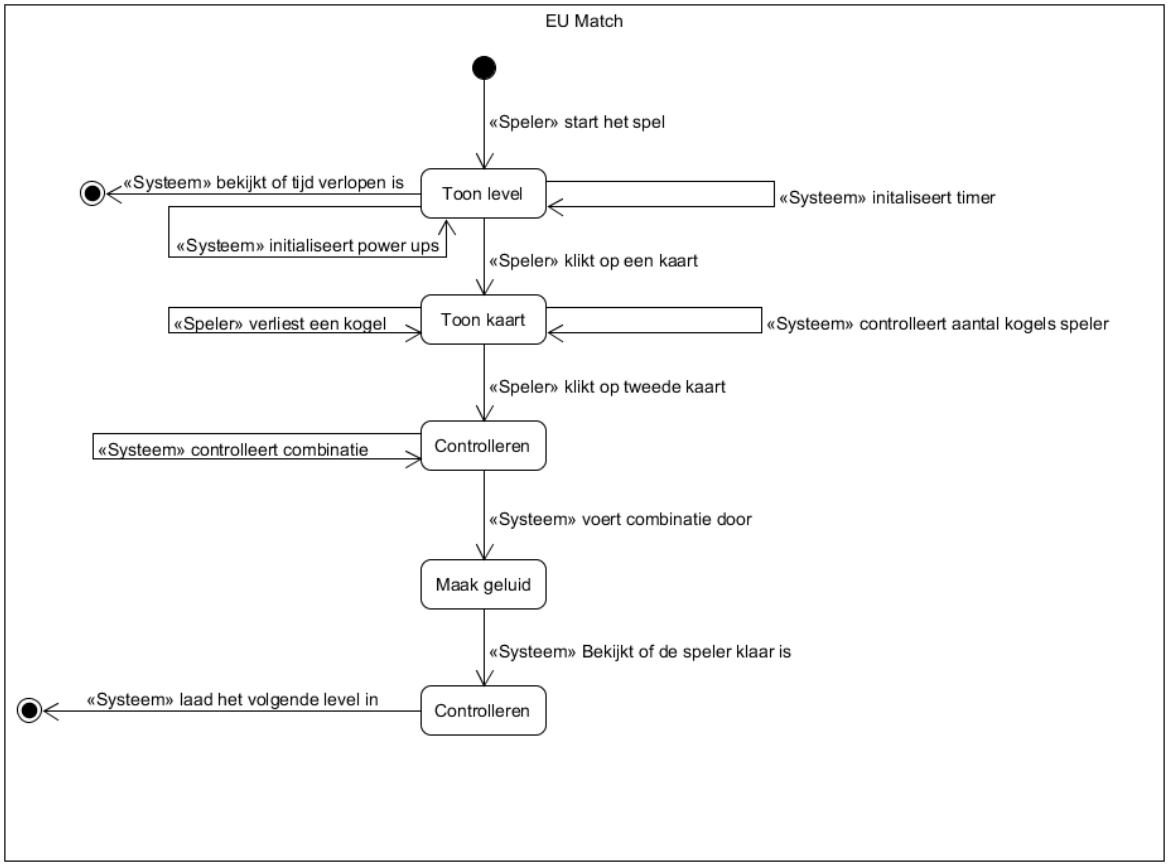
Dit script word gebruikt om variabelen niet te vernietigen per scene maar juist om deze door te laten gaan waar de variabele gebleven was. Pas op het einde van het spel wanneer het afgelopen is word deze vernietigd. Denk hierbij aan de highscore die behaald was tijdens de sessie.

Powerup

Dit script regelt het gehele power up systeem binnen de game. Zo bepaalt het script wanneer een power up tevoorschijn komt en weke power up hierbij van toepassing is. Ook regelt het script de gedurende tijd de power up actief is deze weer uit word gezet.

2 Activity Diagram

In het Activity diagram word beschreven globaal hoe het spel in zijn werking gaat.



Figuur 2 Activity Diagram EU Match

Uitleg

Wanneer het spel opgestart word de timer aangezet en de power ups aangezet. Het systeem zal gedurende het spel bekijken wanneer de power ups tevoorschijn kunnen komen en wanneer de tijd verlopen is. Wanneer de timer verlopen is dan eindigt het spel. Daarnaast bekijkt het systeem wanneer op een kaart word geschoten hoeveel kogels de speler heeft. Indien de speler genoeg kogels heeft dan verliest de speler een kogel. Wanneer de speler op de tweede kaart schiet gebeurt het proces opnieuw alleen dit maal bekijkt het systeem of de kaarten gezamenlijk een combinatie zijn. Wanneer de kaarten een combinatie vormen dan voert het systeem de combinatie door en bekijkt vervolgens of de speler klaar is. Wanneer alle combinaties gemaakt zijn en niks meer over is dan is het level afgelopen.

3 Testresultaten

De test resultaten beschreven hieronder zijn ook terug te vinden in de Assets folder onder het mapje Tests. De testen zijn unit tests die zijn uitgevoerd binnen het project.

3.1 Unit Test 1

Binnen deze test worden 2 testen uitgevoerd om te kijken of de software naar wens hierop reageert.

Scenario 1: Negatieve waarde aan de timer toegepast
Verwachting: De tijd is verstreken het level stopt de speler is hierbij af

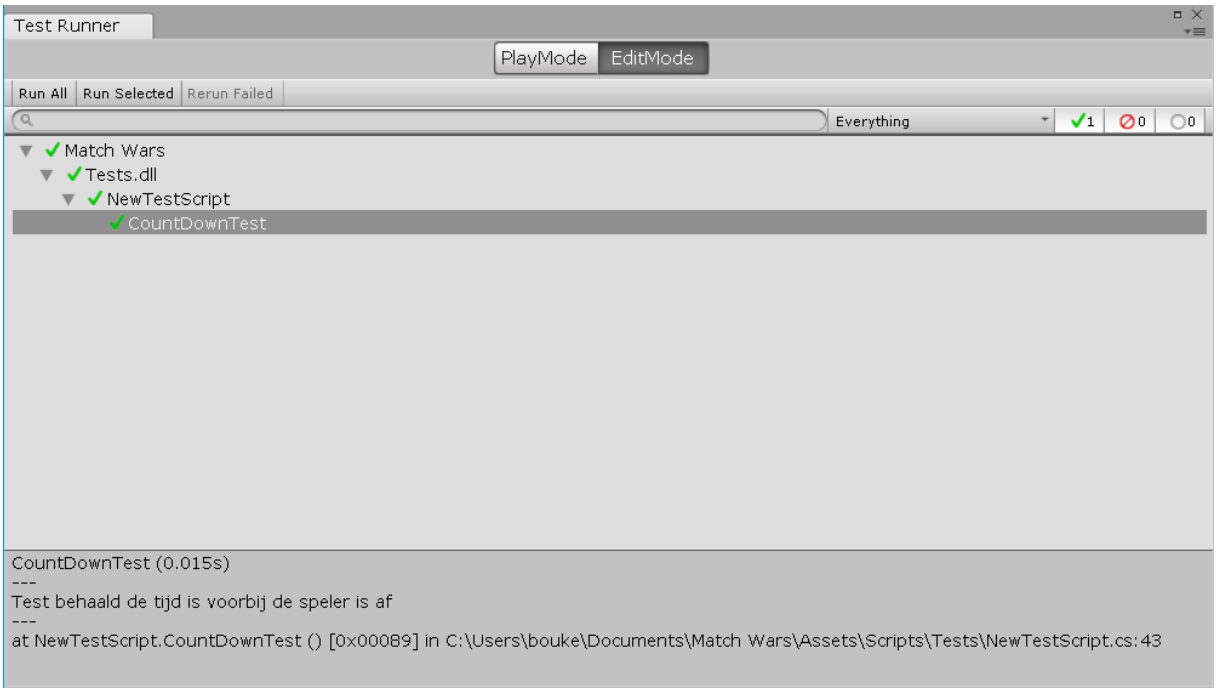
Initialisatie

```
private float mainTimer = -1;
private float timer;
private bool canCount = true;
private bool doOnce = false;
private bool FinishedLevel = false;
```

Code

```
public void CountDownTest() {
    timer = mainTimer;
    if (timer >= 0.0f && canCount && FinishedLevel == false) {
        timer -= Time.deltaTime;
        Assert.Fail();
    } else if (timer <= 0.0f && !doOnce) {
        Assert.Pass("Test behaald de tijd is voorbij de speler is af");
    }
}
```

Uitkomst:



Scenario 2: 0 waarde aan de timer toegepast
Verwachting: De test word niet behaald want een negatieve waarde word verwacht

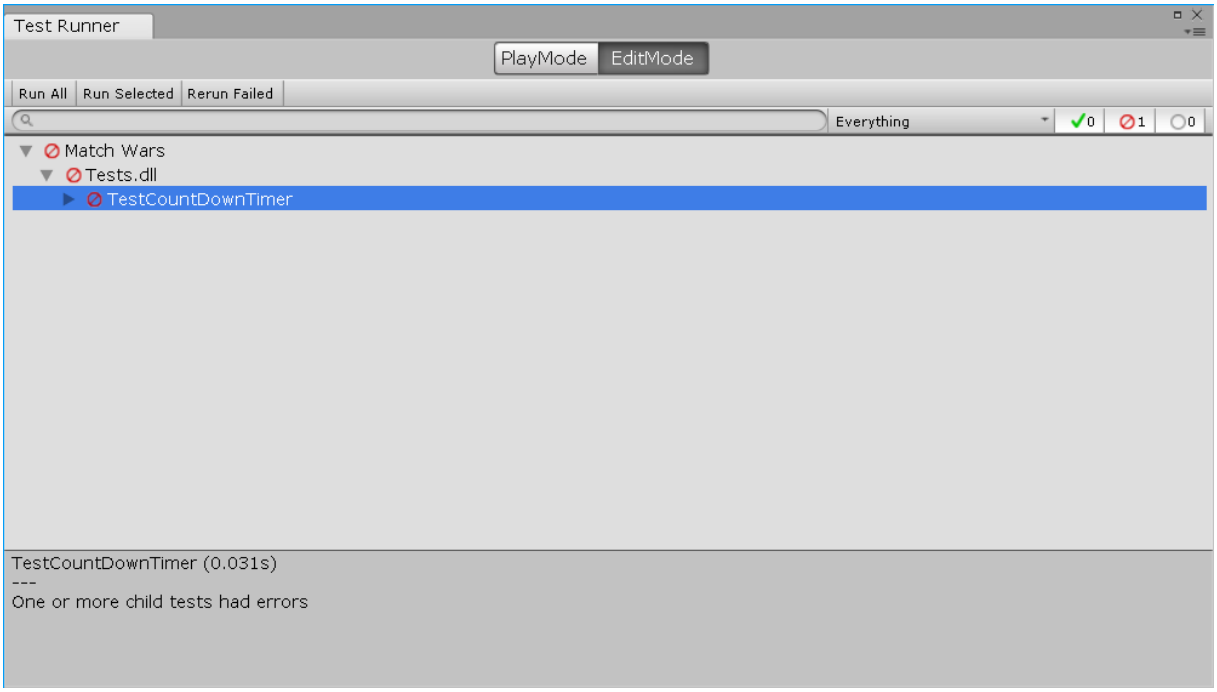
Initialisatie

```
private float mainTimer = 0;
private float timer;
private bool canCount = true;
private bool doOnce = false;
private bool FinishedLevel = false;
```

Code

```
public void CountDownTest() {
    timer = mainTimer;
    if (timer >= 0.0f && canCount && FinishedLevel == false) {
        timer -= Time.deltaTime;
        Assert.Fail();
    } else if (timer <= 0.0f && !doOnce) {
        Assert.Pass("Test behaald de tijd is voorbij de speler is af");
    }
}
```

Uitkomst:



3.2 Unit Test 2

Binnen deze test word gekeken of de speler door kan blijven schieten wanneer de waarde negatief is. De complexe logica die hoort wanneer de kogels niet leeg zijn, zijn in deze test eruit gelaten. Zo focussen we alleen op het munitie gedeelte.

Scenario 1: Munitie met een negatieve waarde
Verwachting: De test word behaald de speler kan niet verder schieten

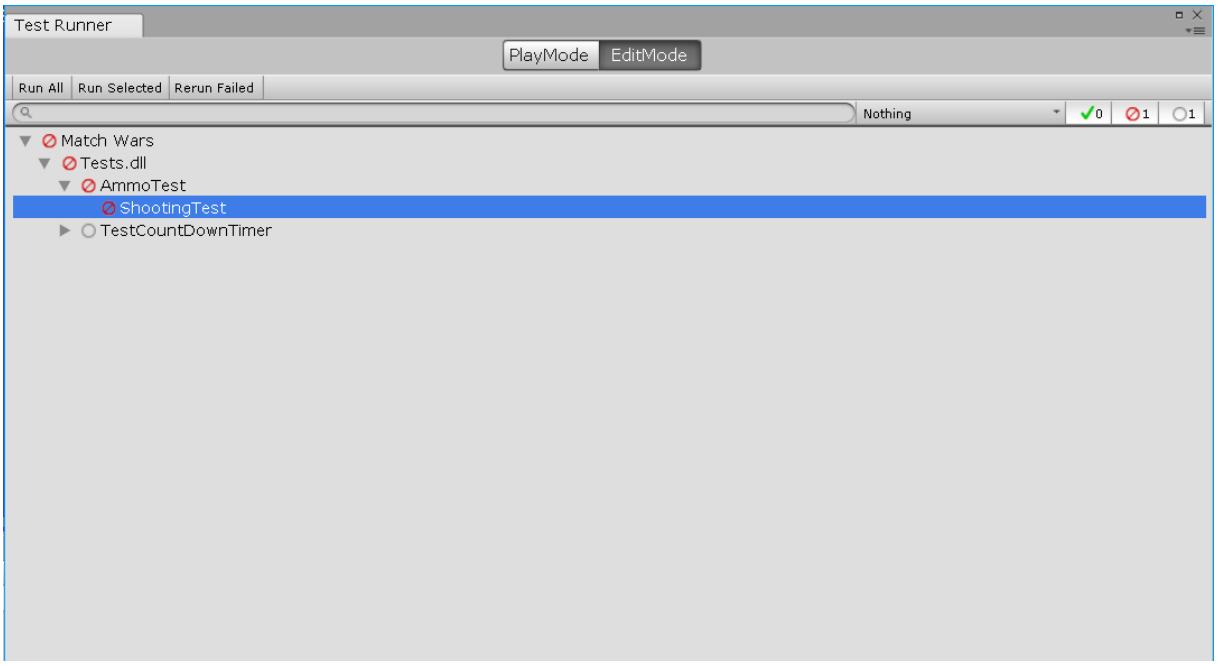
Initialisatie

```
public int ammo = -3;
```

Code

```
public void ShootingTest() {
    if (ammo != 0) {
        Assert.Fail();
    } else if (ammo <= 0) {
        Assert.Pass("Test behaald de speler kan niet langer schieten");
    }
}
```

Uitkomst:



Conclusie

De test is niet gehaald terwijl de verwachting was dat dit behaald zou worden.

Scenario 2: Munitie met een negatieve waarde [Code aangepast]
Verwachting: De test word behaald de speler kan niet verder schieten

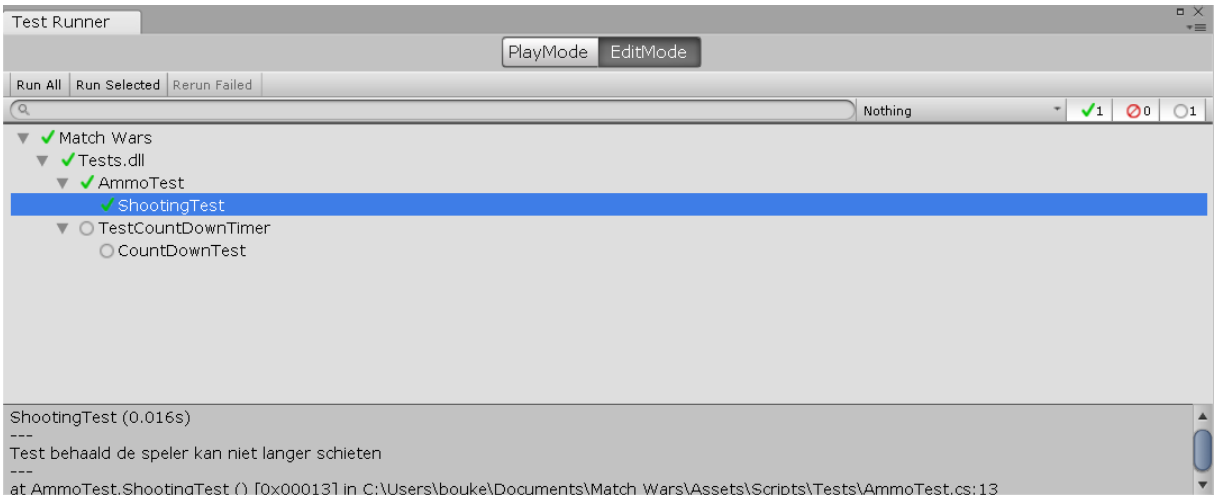
Initialisatie

```
public int ammo = -3;
```

Code

```
public void ShootingTest() {

    if (ammo <= 0) {
        Assert.Pass("Test behaald de speler kan niet langer schieten");
    } else {
        Assert.Fail();
    }

}
```

Uitkomst:



Conclusie

Om het probleem te voorkomen en op te lossen word als eerst gekeken of de speler lager of gelijk aan 0 munitie over heeft. Wanneer dit niet het geval is zal de logica van de kaarten worden toegepast.

3.3 Unit Test 3

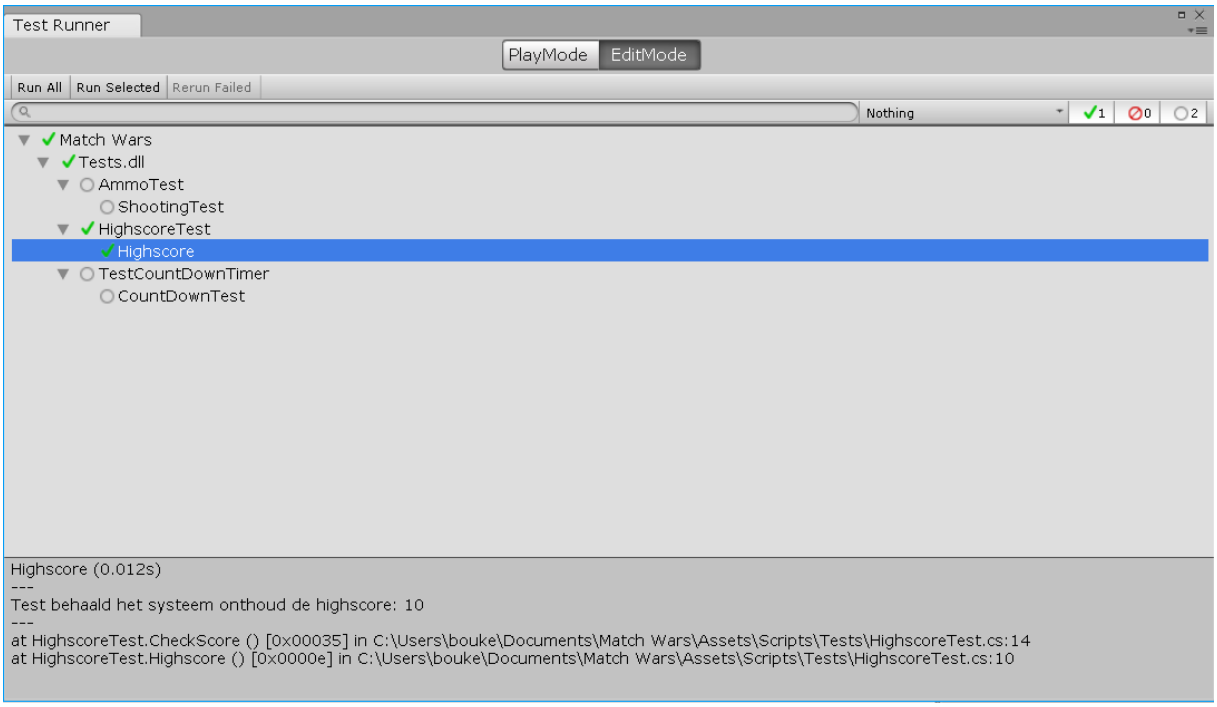
Binnen deze test word gekeken hoe de playerprefs binnen unity werken. We willen immers dat de highscore per scene opnieuw opgehaald kan worden en telkens opgeslagen kan worden

Scenario 1: De highscore word opgeslagen onder de sleutel HighscoreKey met een waarde van 10
Verwachting: Dat de waarde van de sleutel HighscoreKey de juiste waarde 10 bevat

Code

```
public class HighscoreTest  {
  [Test]
  public void Highscore() {
        PlayerPrefs.SetInt("HighscoreKey", 10);
        CheckScore();
    }
    public void CheckScore() {
        if (PlayerPrefs.GetInt("HighscoreKey") == 10) {
            Assert.Pass("Test behaald het systeem onthoud de highscore: "+
PlayerPrefs.GetInt("HighscoreKey").ToString());
        }
        Assert.Fail();
    }
```

}**Uitkomst:**



Conclusie

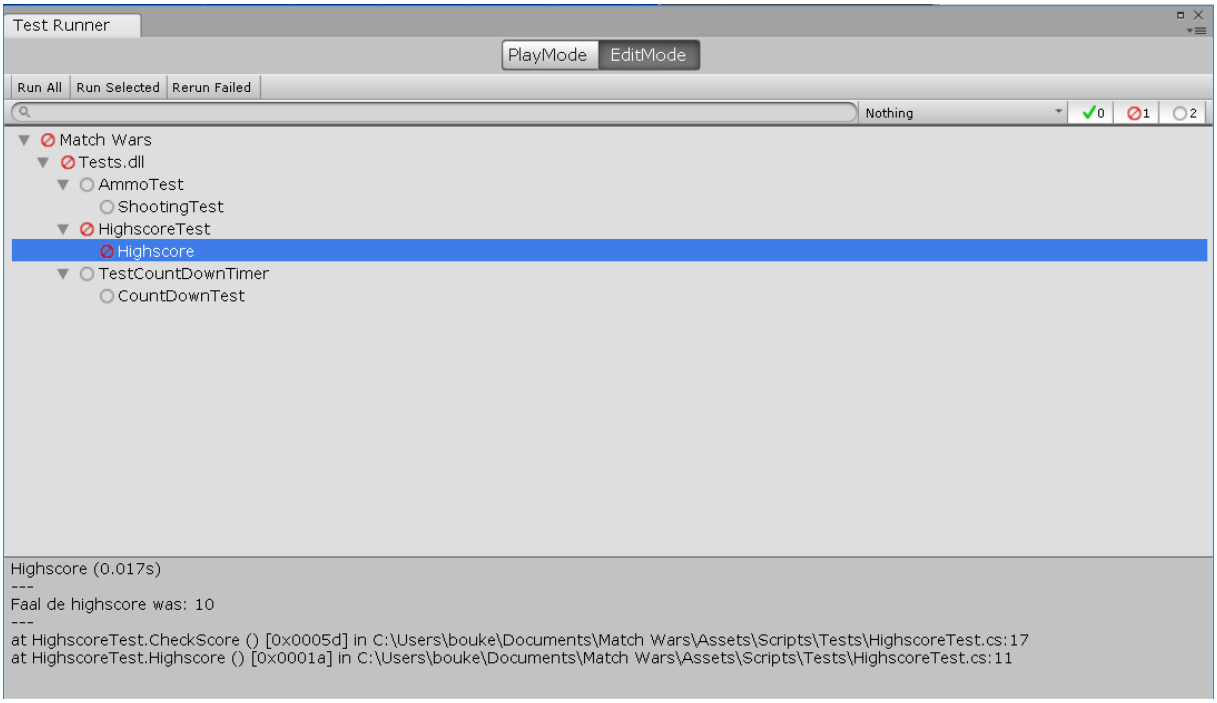
De test is succesvol behaald het deed aan de verwachtingen.

Scenario 2: Score toevoegen en de value met 10 ophogen binnen de value van de key [Code aangepast]
Verwachting: De test word behaald het systeem registreert dat de value 20 is.

Code

```
public class HighscoreTest  {
  [Test]
  public void Highscore() {
        PlayerPrefs.SetInt("HighscoreKey", 10);
        PlayerPrefs.SetInt("HighscoreKey", +10);
        CheckScore();
    }
    public void CheckScore() {
        if (PlayerPrefs.GetInt("HighscoreKey") == 20) {
            Assert.Pass("Test behaald het systeem onthoud de highscore: "+
PlayerPrefs.GetInt("HighscoreKey").ToString());
        }
        Assert.Fail();
    }


}
```

Uitkomst:



Conclusie

Een waarde erbij toevoegen met +10 binnen de waarde van de sleutel zorgt er niet voor dat deze word verdubbeld. De test was bij deze mislukt gezien de verwachting was dat de waarde van de highscore alsnog bij opgeteld zou worden.

3.4 Unit Test 4

Binnen deze test word gekeken hoe ervoor gezorgd kan worden dat een powerup een keuze kan hebben uit meerdere powerups. De tijd en duur zijn hierbij weggelaten. In deze test is de focus of het systeem de juiste powerup kan vinden en of deze zich ook aan de verwachting houd

Scenario 1: Er zijn meerdere power ups het systeem kan deze gebruiken
Verwachting: Toont alleen de power ups die verwerkt zijn het systeem zal geen fouten opleveren

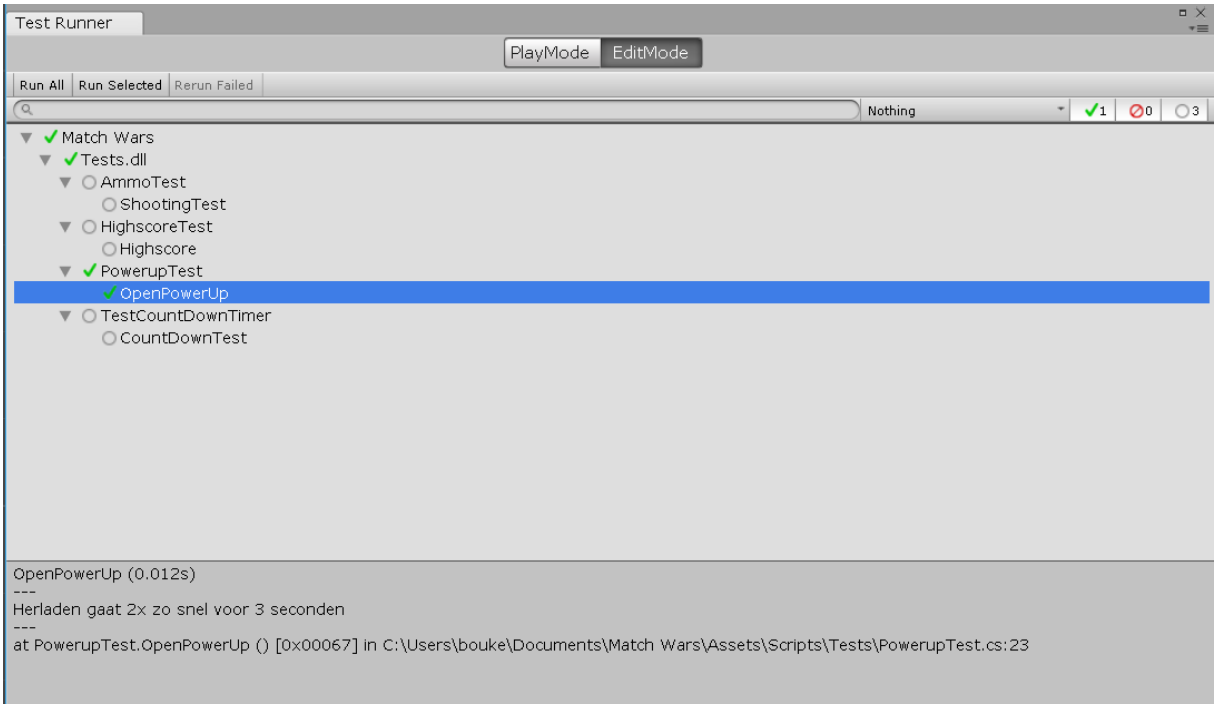
Initialisatie

```
    private int Gamble;
    private int Respawn;
    public float timer;
```

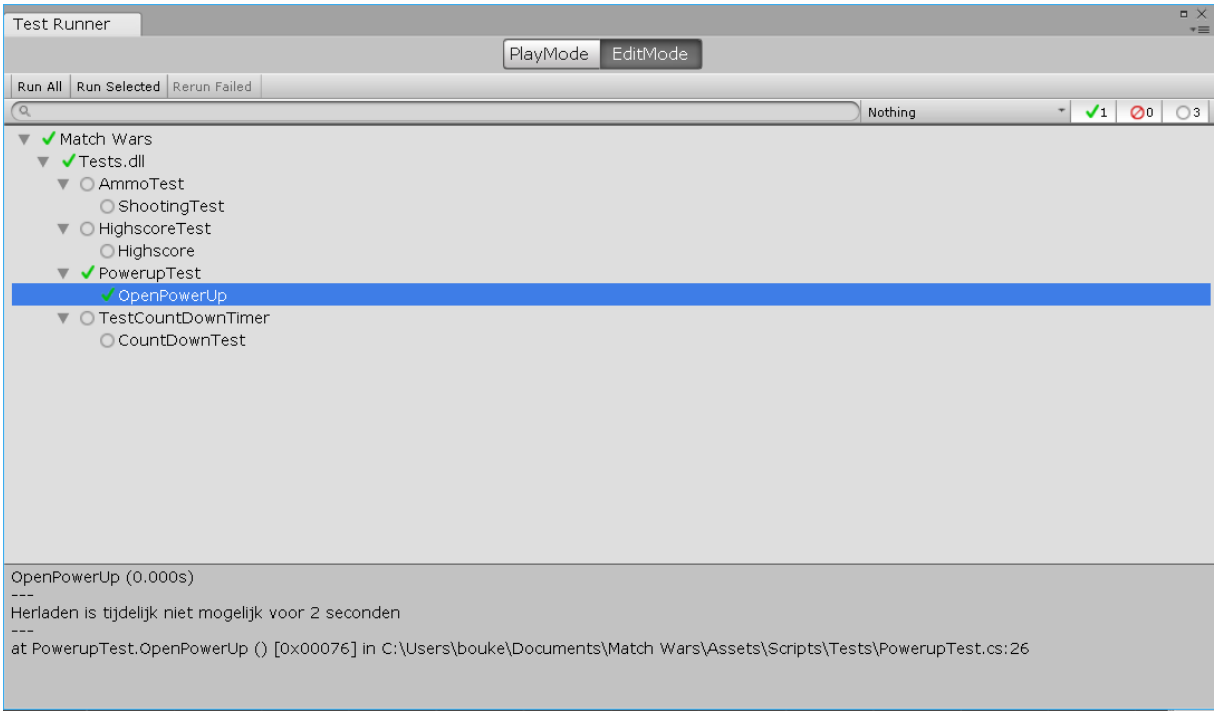
Code

```
    public void OpenPowerUp() {
        System.Random rnd = new System.Random();
        Gamble = rnd.Next(7);
        switch (Gamble) {
                //Add +5 ammunition voor de speler
                case 1:
                Assert.Pass("Higscore + 5");
                break;
                case 2:
                Assert.Pass("Kaarten draaien sneller om voor 15 seconden");
                break;
                case 3:
                Assert.Pass("Herladen gaat 2x zo snel voor 3 seconden");
                break;
                case 4:
                Assert.Pass("Herladen is tijdelijk niet mogelijk voor 2 seconden");
                break;
                case 5:
                Assert.Pass("Herladen gaat fout voor 3 seconden");
                break;
                case 6:
                Assert.Pass("-5 ammunitie toegevoegd");
                break;
                case 7:
                Assert.Pass("Beast mode voor 13 seconden");
                break;
                default:
                Assert.Fail("powerup kwam niet in voor");
                break;

        }
    }

}
```

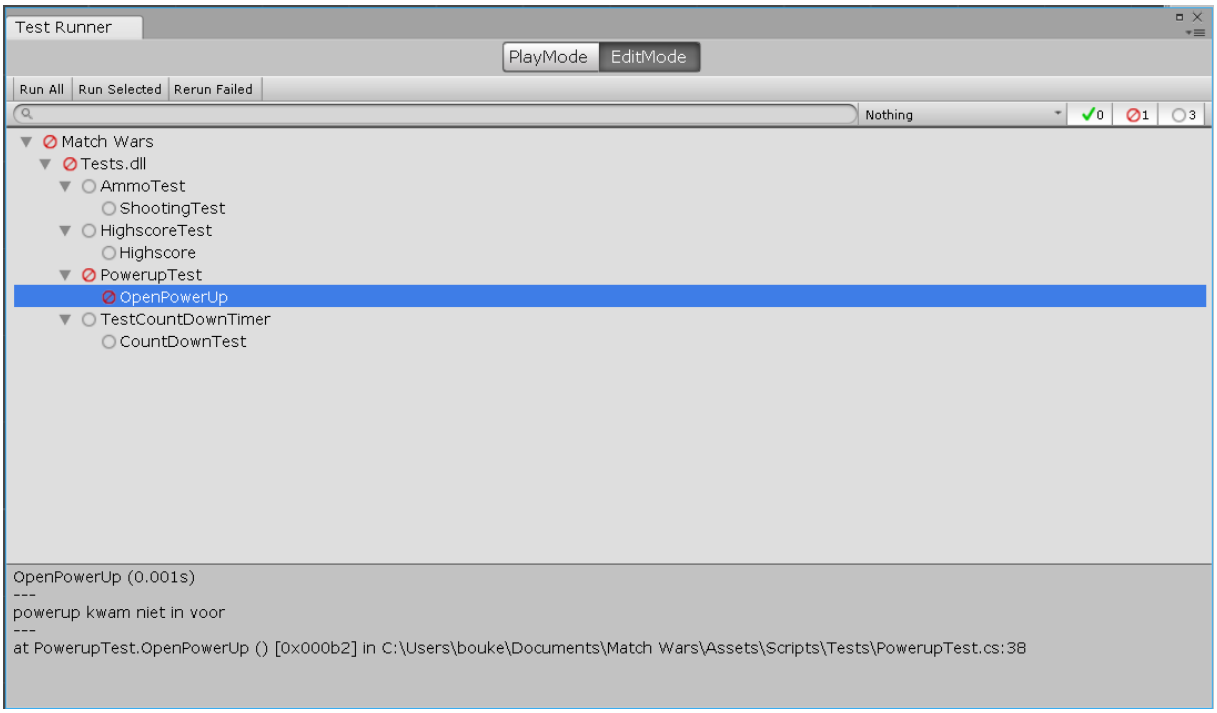
Uitkomst:


Test Runner
PlayMode
EditMode
Run All
Run Selected
Rerun Failed
Nothing
Match Wars
Tests.dll
AmmoTest
ShootingTest
HighscoreTest
Highscore
PowerupTest
OpenPowerUp
TestCountDownTimer
CountDownTest
OpenPowerUp (0.012s)

Herladen gaat 2x zo snel voor 3 seconden
---
at PowerupTest.OpenPowerUp () [0x00067] in C:\Users\bouke\Documents\Match Wars\Assets\Scripts\Tests\PowerupTest.cs:23


Test Runner
PlayMode
EditMode
Run All
Run Selected
Rerun Failed
Nothing
Match Wars
Tests.dll
AmmoTest
ShootingTest
HighscoreTest
Highscore
PowerupTest
OpenPowerUp
TestCountDownTimer
CountDownTest
OpenPowerUp (0.000s)

Herladen is tijdelijk niet mogelijk voor 2 seconden
---
at PowerupTest.OpenPowerUp () [0x00076] in C:\Users\bouke\Documents\Match Wars\Assets\Scripts\Tests\PowerupTest.cs:26



Conclusie

De test is niet behaald er zit een fout in de Gamble waarde. Het probleem is dat een random value van 0 naar … begint ik ging hierbij vanuit dat het bij 1 begon.

Scenario 2: Er zijn meerdere power ups het systeem kan deze gebruiken [Code aangepast]
Verwachting: De test word behaald en zal nooit op default uitkomen

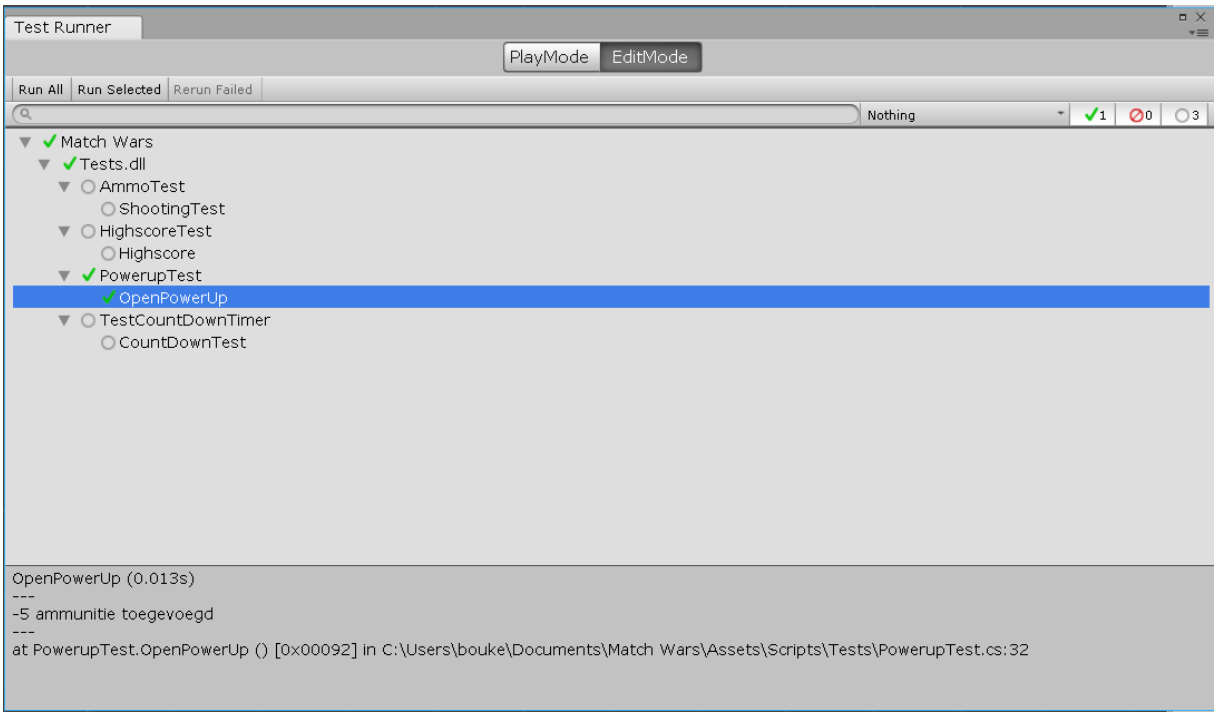
Initialisatie

```
    private int Gamble;
    private int Respawn;
    public float timer;
```

Code

```
    public void OpenPowerUp() {
        System.Random rnd = new System.Random();
        Gamble = rnd.Next(7);
        switch (Gamble) {
                //Add +5 ammunition voor de speler
                case 0:
                Assert.Pass("Higscore + 5");
                break;
                case 1:
                Assert.Pass("Kaarten draaien sneller om voor 15 seconden");
                break;
                case 2:
                Assert.Pass("Herladen gaat 2x zo snel voor 3 seconden");
                break;
                case 3:
                Assert.Pass("Herladen is tijdelijk niet mogelijk voor 2 seconden");
                break;
                case 4:
                Assert.Pass("Herladen gaat fout voor 3 seconden");
                break;
                case 5:
                Assert.Pass("-5 ammunitie toegevoegd");
                break;
                case 6:
                Assert.Pass("Beast mode voor 13 seconden");
                break;
                default:
                Assert.Fail("powerup kwam niet in voor");
                break;

        }
    }

}
```

Uitkomst:



Conclusie

Gezien de waarde steeds anders is heb ik een flink aantal keer deze test opnieuw uitgevoerd en kwam altijd groen uit. Alle power ups werden uiteindelijk gebruikt. De test is hierbij geslaagd.