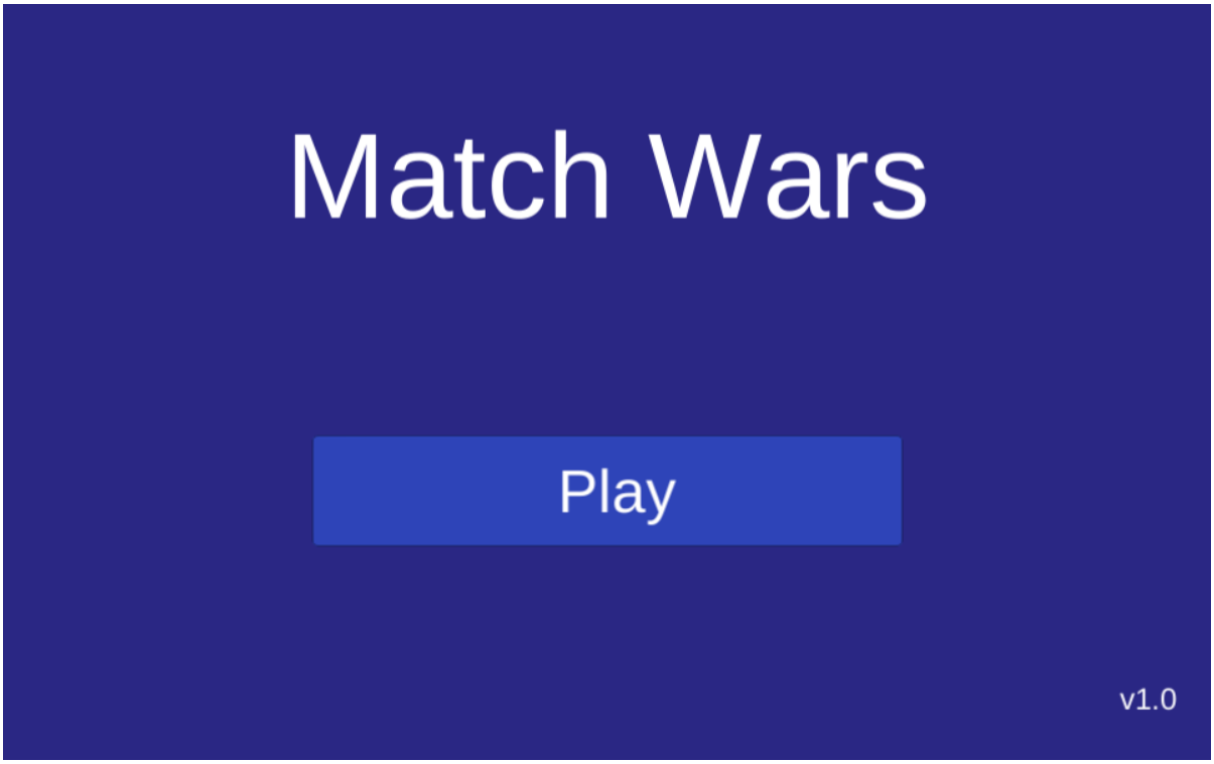
Technisch Verslag



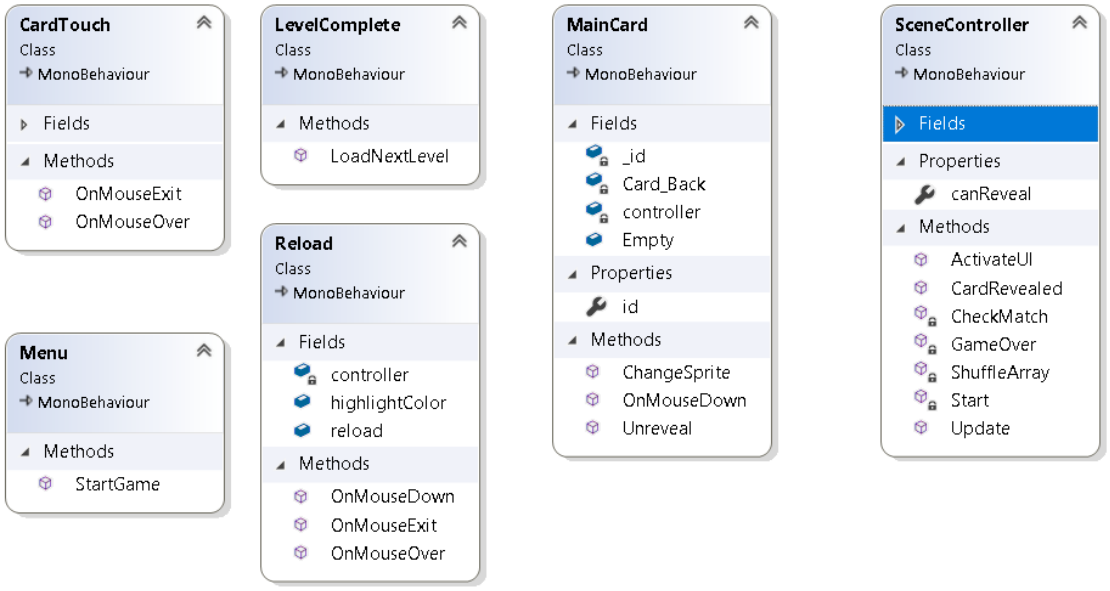
Bouke de Vries
Studentnummer: 1683542
Datum: 5-11-2018
Versie 1.0

Versiebeheer

Aanpassingen	Versie	Datum
Eerste document	1.0	5-11-2018

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

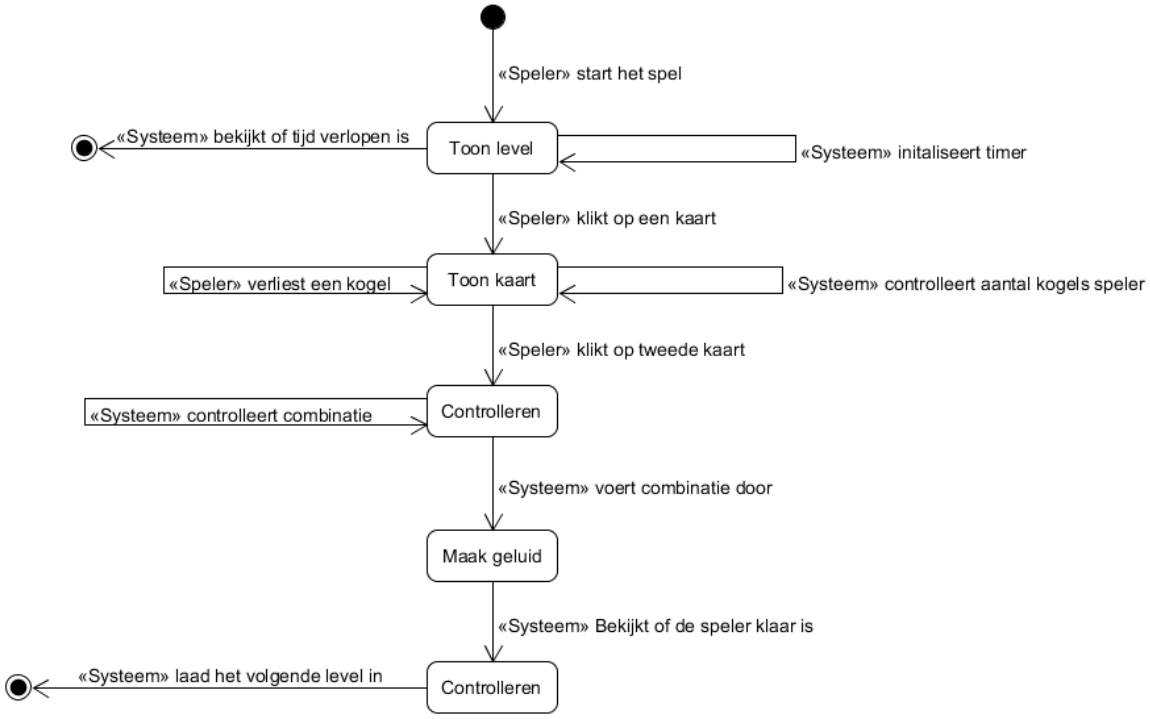
Het script de MainCard bepaalt welke kaart zojuist is geselecteerd en welke is omgelegd ook word binnen dit script vastgesteld wanneer de speler een kaart omdraaid de speler tegelijkertijd ook een kogel verliest. Zo roept het script ook de SceneController hiervoor aan. Maar gebruikt het ook de SceneController 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,tijd 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.

2 Activity Diagram

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



Figuur 2 Activity Diagram Match Wars

Uitleg

Zo word wanneer het spel opgestart word de timer aangezet en bekijkt het systeem of de timer verlopen is of niet. 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

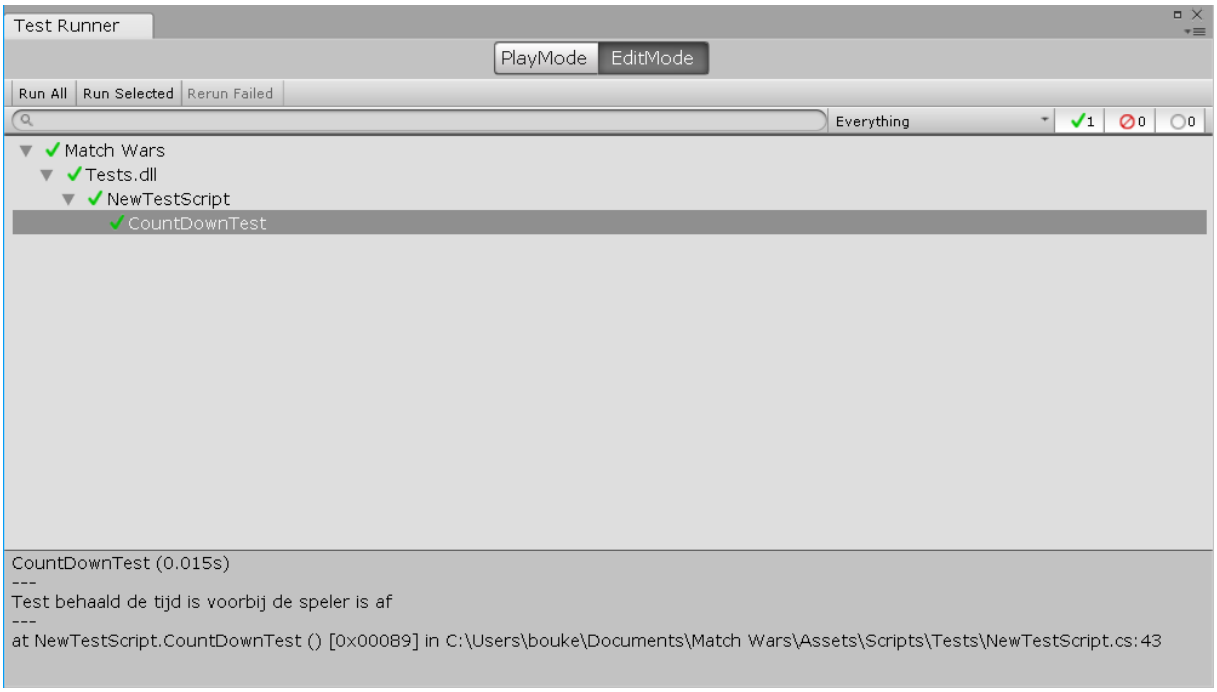
Initialisatie

```
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

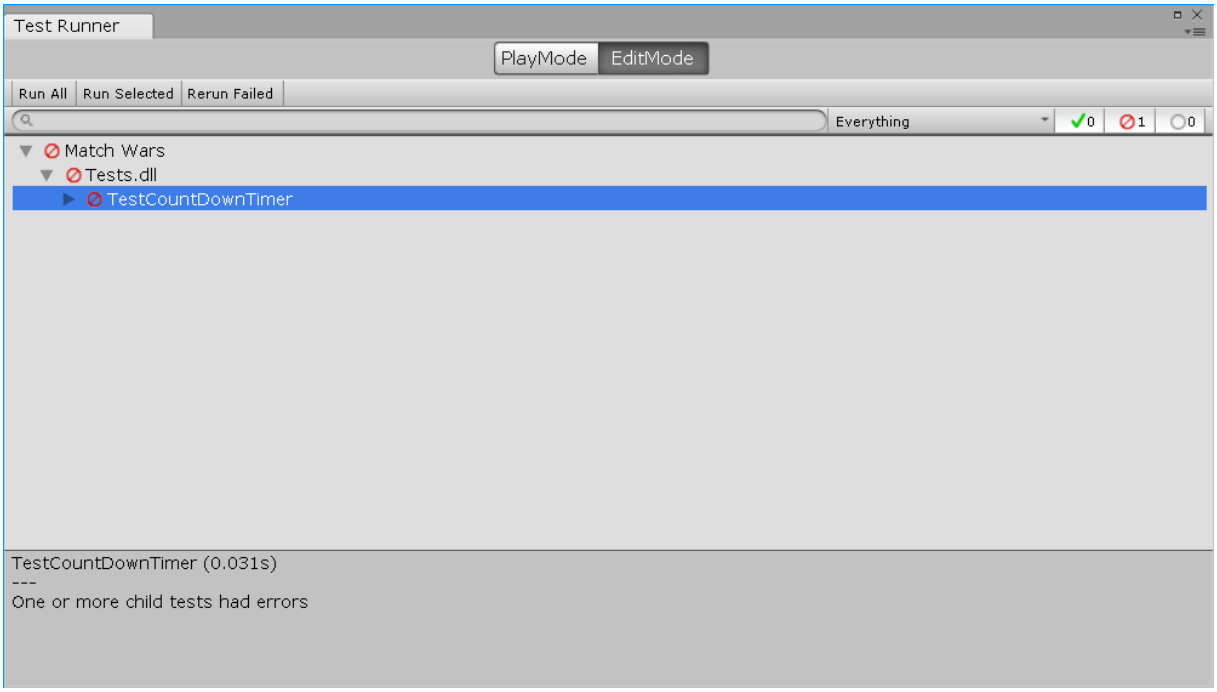
Initialisatie

```
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

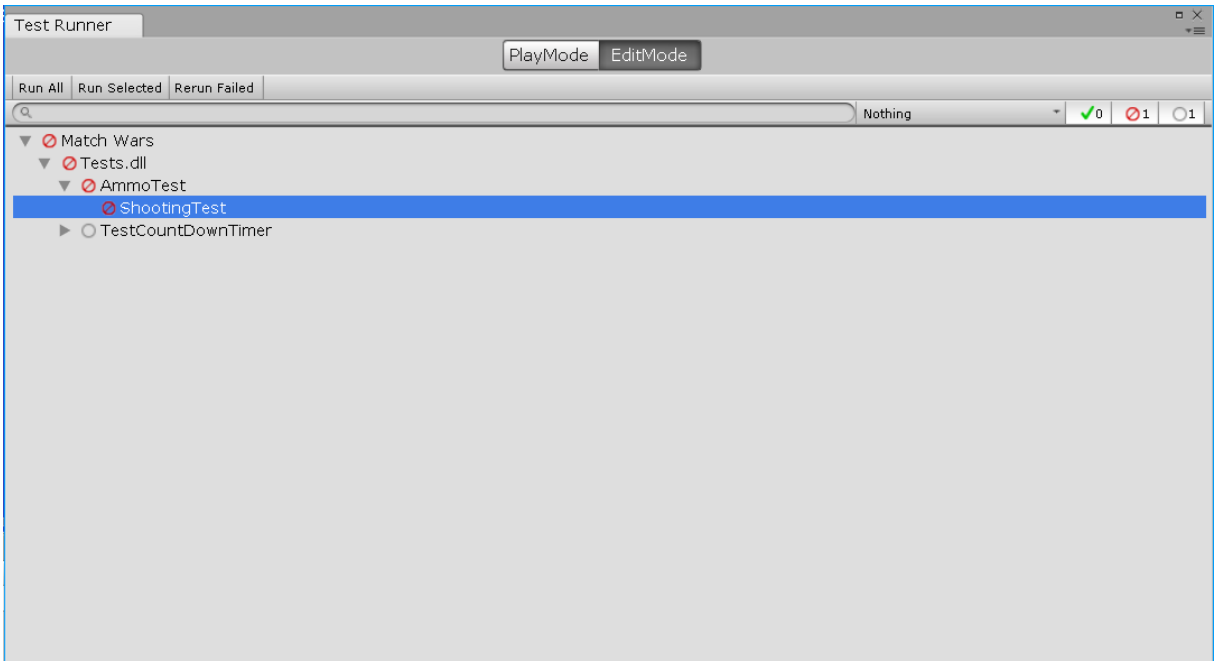
Initialisatie

```
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;
```

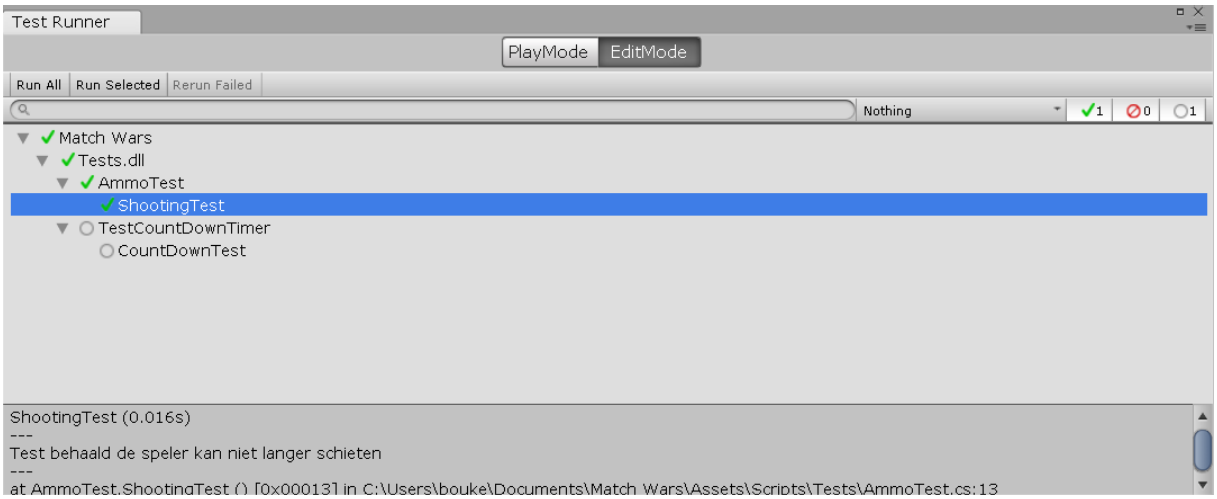
Code

```
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.