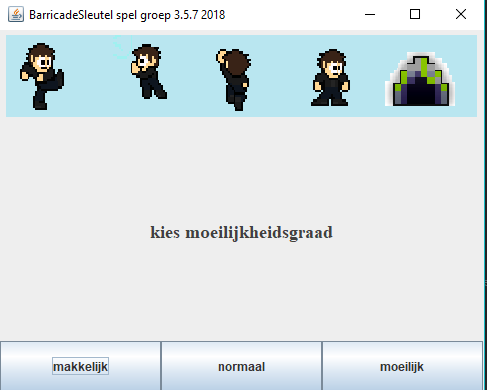
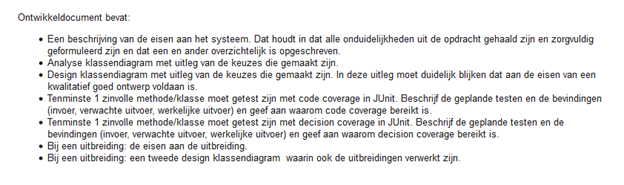
**Ontwikkeldocument Groep 3.5.7**

Door: Mucahit Coskun, Jurian van Hoorn en Tarik Cherif

Datum: 03/04/2018



Inhoudsopgave:



1. Requirements..............................................................2

2. Beschrijving van de eisen aan het systeem.................3

3. Analyse klassendiagram...............................................4

4. Design klassendiagram.................................................5

5. JUnit code coverage.....................................................6

6. JUnit decision coverage................................................8

7. Design klassendiagram uitbreiding...............................9

1. Requirements:

De requirements voor dit project hebben wij uit de Projectbeschrijving gehaald die op Blackboard staat. Het betreffende document vindt U [Hier](https://drive.google.com/open?id=1TwJtHmyBijFDir-xdbuBaEORXwvyymd0). Wij hebben dit document bestudeerd en aan de hand van de informatie die wij uit het document konden halen de beschrijving van de eisen opgesteld.

2. Beschrijving van de eisen aan het systeem:

Wikipedia citaat:”De **MoSCoW-methode** is een wijze van prioriteiten stellen in onder meer de [software engineering](https://nl.wikipedia.org/wiki/Software_engineering). De eisen aan het resultaat van een [project](https://nl.wikipedia.org/wiki/Project) worden ermee ingedeeld. Het is een afkorting, waarvan de letters staan voor:[[1]](https://nl.wikipedia.org/wiki/MoSCoW-methode#cite_note-1)

* **M** - **must haves**: deze eisen ([*requirements*](https://nl.wikipedia.org/wiki/Requirement)*) moeten* in het eindresultaat terugkomen, zonder deze eisen is het product niet bruikbaar;
* **S** - **should haves**: deze eisen zijn zeer gewenst, maar zonder is het product wel bruikbaar;
* **C** - **could haves**: deze eisen zullen alleen aan bod komen als er tijd genoeg is;
* **W** - **won't haves**: deze eisen zullen in dit project niet aan bod komen maar kunnen in de toekomst, bij een vervolgproject, interessant zijn.“

Hieronder staat de beschrijving van de eisen aan het systeem:

must have

* Een speelveld in een vierkant vlak
* Speler die kan lopen naar het eindveld
* Een speelveld heeft de muren
* Een speelveld heeft de barricades met het nummer
* Een speelveld heeft de sleutel met het nummer
* Speler kan barricade openen met passende sleutel
* Speler kan eerst sleutel pakken
* Speler kan sleutel meerdere keren gebruiken
* Speler kan één sleutel in zijn zak hebben
* Als speler een sleutel pakt, verdwijnt het uit het veld, dan zit sleutel in de zak van de speler
* Je krijgt een melding als een sleutel niet past
* De barricade verdwijnt als de sleutel wel past

Vervolg op volgende pagina..

* Speler wordt verplaatst d.m.v. pijltjestoetsen
* Halverwege spel moet het mogelijk zijn hetzelfde speelveld opnieuw te starten
* Spel heeft knoppen om spel te starten, een spel te kiezen en een +spel te stoppen
* Speler loopt niet door een muur heen
* Speler loopt niet uit het speelveld
* Spel heeft een moeilijkheidsgraad

could have

* Een uitbreidbare functie applicatie
* De afbeeldingen voor een speler
* Speelveld heeft willekeurige spelobjecten.

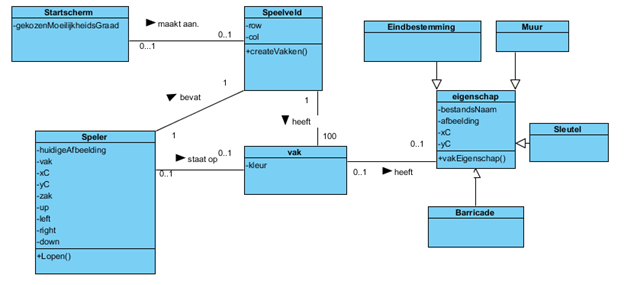
should have

* N.v.t

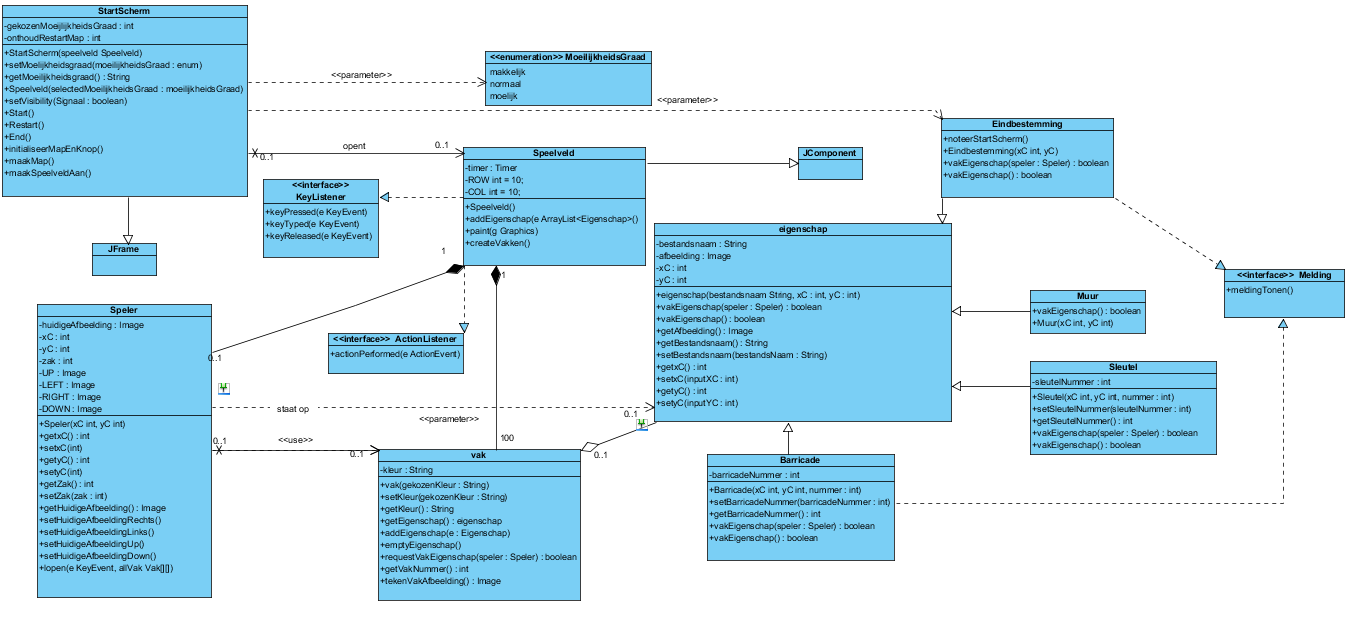
wont have

* N.v.t

3. Analyse klassendiagram:

Wij wilden voor ons design de game zo ontwerpen dat wij een startscherm hebben en een speelveld. Vanuit dit startscherm kan je naar het speelveld en de game spelen. Een speler staat op een vak en een speelveld bevat 100 vakken (dit verandert niet). Er is een hoofdklasse 'eigenschap' , die is onderverdeeld in de typen vakken barricade, sleutel, muur en eindbestemming. Een vak hoeft geen eigenschap te bevatten, dan is dit vak een leeg vak waar de speler op mag lopen.

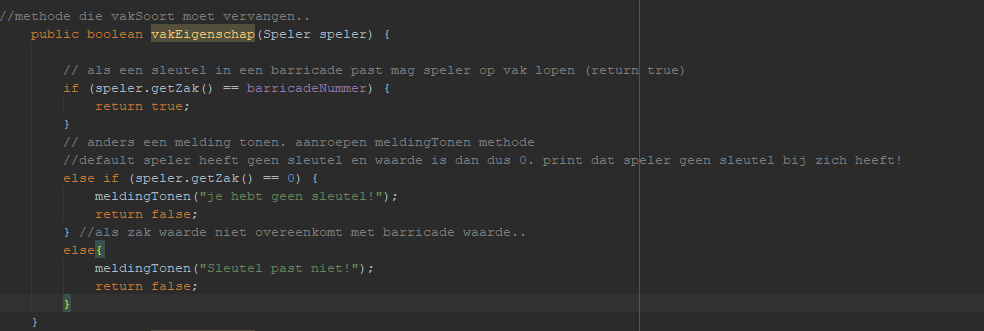
4. Design klassendiagram:



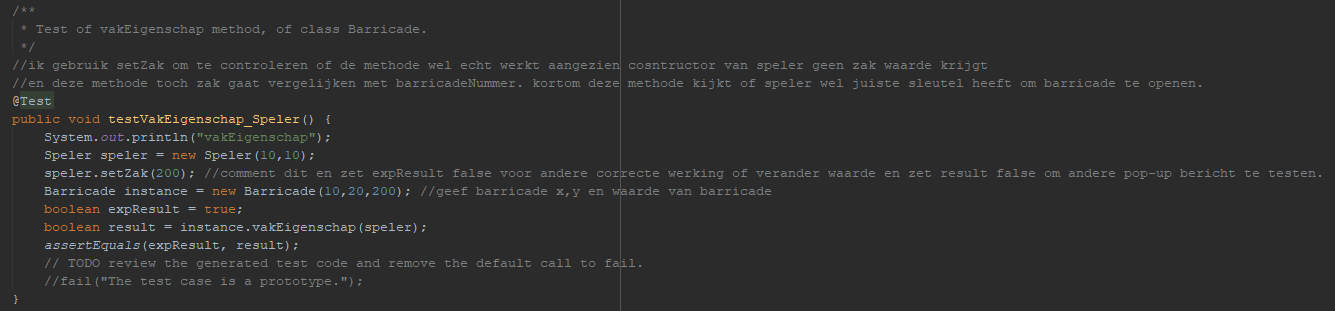
Over ons Design Klasse Diagram, zelfde klassen als in het analyse diagram maar met methodes, typen en interfaces, soorten interactie tussen de klassen en aangeven of het gebruik maakt van JFrame, Actionlistener en dergelijke. Wij hebben de logica voor het bewegen in de Speler klasse gezet en roepen dit aan bij speelveld. Dit omdat speelveld de KeyEvents ophaalt en dit aan speler geeft.

5. JUnit Tests met uitleg:

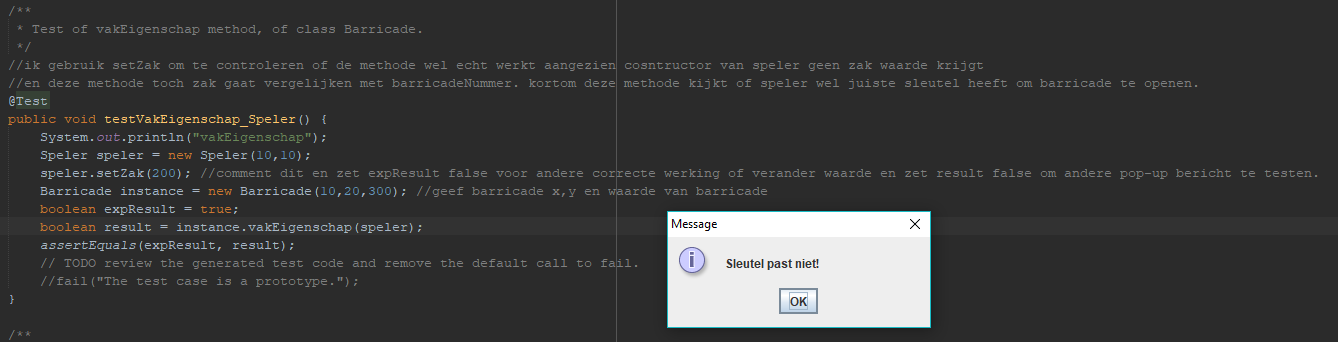
Test 1, klas Barricade wordt getest, wij testen hier de vakEigenschap methode. Deze methode heeft als doel de Zak waarde van een speler te vergelijken met een barricadeNummer van een barricade.

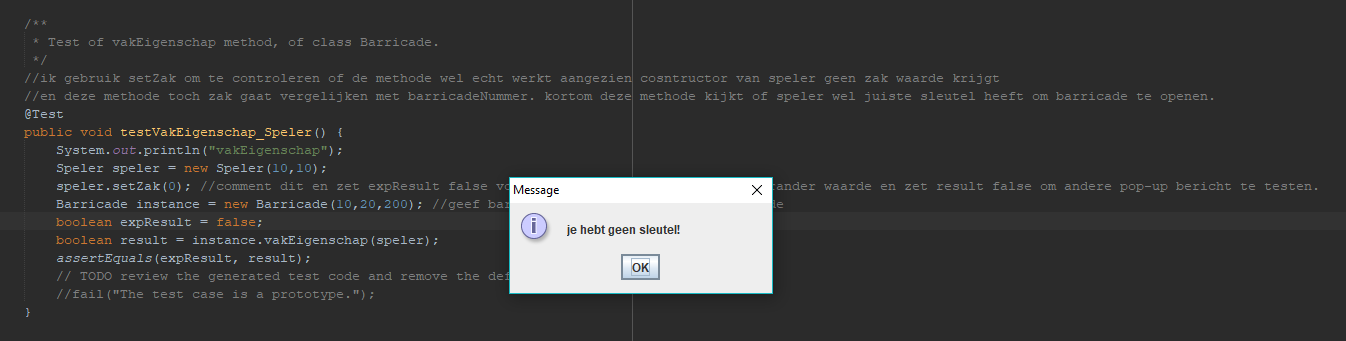


De methode hierboven Fig.1. De test case hieronder Fig.2.



Figuur 3 laat hieronder zien dat er ook false uit kan komen. Wanneer de methode ziet dat de waarden van Zak en barricadeNummer niet overeenkomen, komt er een pop-up message. Als we expectedResult 'false' hadden gezet, was de methode geslaagd zonder pop-up melding te geven. De methode werkt correct door een melding te geven dat sleutel niet overeenkomt met barricade.





Figuur 4. Hier zien wij wat er gebeurt al Zak waarde 0 heeft. De methode doet wat er verwacht wordt een geeft de melding dat speler geen sleutel heeft.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| methode | Test stappen | Werkelijke resultaat | Verwachte resultaat | Resultaat geslaagd / gefaald |
| vakEigenschap | Fig.1en 2 | true | true | geslaagd |
| vakEigenschap | Fig.1 en 3 | false | false | geslaagd |
| vakEigenschap | Fig1. En 4 | false | false | geslaagd |

p.7

6. JUnit decision coverage

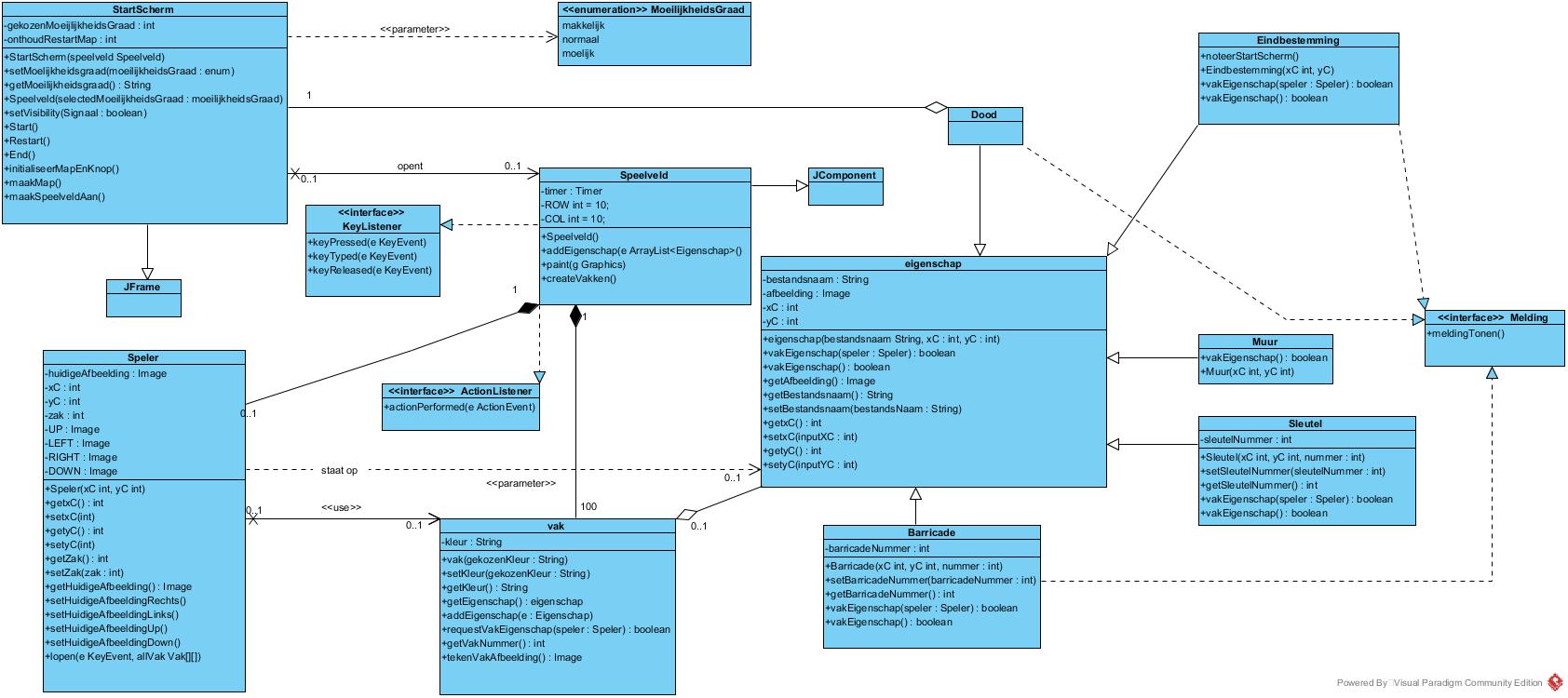
In methode lopen staan 4 voorwaarden voor iedere pijlen. Als een gebruiker op een links pijltoets drukt, dan gaat speler naar links lopen. Hieronder weergeeft een test case in een tabelvorm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pijlen van toetsenbord |  | Actual Result | Expected Result | Results fail/pass |
| beweegLinks |  | Links | links | geslaagd |
| beweegRechts |  | Speler loopt naar rechts | Speler loopt naar rechts | geslaagd |
| beweegBoven |  | Speler loopt naar boven | Speler loopt naar boven | geslaagd |
| Omlaag |  | Speler loopt naar beneden | Speler loopt naar beneden | geslaagd |

In een methode vakEigenschap(Speler speler) van de Barricade heeft 3 voorwaarden:

1. Als een sleutel van een speler gelijk is aan het nummer van een barricade, dan mag barricade geopend worden.
2. Als de sleutel niet gelijk is aan een nummer van een barricade

7. Design Diagram uitbreiding



Een nieuwe subclass “dood” is toegevoegd. “Dood” is subclass van eigenschap. Bovendien maakt “dood” gebruikt van interface “Melding”. Dood heeft “startscherm” als een instantie variable nodig, om zijn vakeigenschap te kunnen uitvoeren. Daarnaast is een nieuwe operatie van Startscherm “heropenStartscherm” ook toegevoegd. Wanneer een speler op het vak Dood zal staan, dan zal hij doodgaan. Daardoor herstart het spel.