Ontwikkeldocument

Object Orientation Project

De Haagse Hogeschool

Periode 3

**MazeGame JLR**

**Namen:** Roland Dovendinas, Leslie Lauw, Juliën van der Niet

**Studentennummers:** 18090257, 18151167, 18069681

**Klas:** h-1-se.1.b

**Groep:** 4

**Datum:** 01/04/2019

# **Inhoudsopgave**

[**Inhoudsopgave** 1](#_Toc5319530)

[**System Requirements** 2](#_Toc5319531)

[Requirements 2](#_Toc5319532)

[**Extra toelichting code** 3](#_Toc5319533)

[Methodes voor latere uitbreidingen 3](#_Toc5319534)

[Opbouw code 3](#_Toc5319535)

[JavaDoc commentaar 3](#_Toc5319536)

[**Analyse Klassendiagram** 4](#_Toc5319537)

[Beschrijving 4](#_Toc5319538)

[**Design Diagram** 5](#_Toc5319539)

[Beschrijving 5](#_Toc5319540)

[In het kort 5](#_Toc5319541)

[Toelichting Constants 5](#_Toc5319542)

[**Unit Tests Code Coverage** 7](#_Toc5319543)

[Code Coverage van Inventory Contains Key() method test 7](#_Toc5319544)

[**Unit Test Decision Coverage** 8](#_Toc5319545)

[Decision Coverage Player move() method test 8](#_Toc5319546)

[**Program Extensions** 9](#_Toc5319547)

[Dark Mode 9](#_Toc5319548)

[Requirements van Dark Mode 9](#_Toc5319549)

[Texture Packs 9](#_Toc5319550)

[Requirements van Texture Packs 9](#_Toc5319551)

[Level Statistics 10](#_Toc5319552)

[Requirements van Level Statistics 10](#_Toc5319553)

Analyse klassendiagram 11

Design diagram 12

# **System Requirements**

## Requirements

* Doolhof heeft vaste muren en barricades.
* Barricades kun je openen met een passende sleutel.
* Speler is in staat sleutels op te pakken.
* Speler kan bewegen d.m.v. pijltjestoetsen.
* Speler mag maximaal 1 sleutel op zak hebben.
* Huidige sleutel verdwijnt bij het oppakken van een nieuwe sleutel.
* Melding bij het proberen van een sleutel die niet past.
* Er moet een mogelijkheid zijn om het speelveld te herstarten.
* Het spel moet geschikt zijn voor latere uitbreidingen.
* Het speelveld moet bestaan uit vierkante blokken.
* In de GUI van de game moeten knoppen worden toegevoegd om te starten, herstarten en een spel te kiezen.
* Bij het bereiken van de finishline krijgt de speler een melding dat hij/zij het doolhof heeft opgelost.

# 

# **Extra toelichting code**

## Methodes voor latere uitbreidingen

In onze code vindt u bij de classes extra methodes die wij zelf niet gebruiken in onze code, maar die andere mensen wellicht wel kunnen gebruiken mochten ze onze code gebruiken.

## Opbouw code

Onze code is zodanig opgebouwd dat het altijd begint met de package ‘source’ > imports > class declaration > class constructor > getters & setters > custom methods. Dit hebben we gedaan om het zo overzichtelijk mogelijk te maken.

## JavaDoc commentaar

Voor de JavaDoc commentaar hebben we specifiek gekozen om de default getters & setters niet de commenten. Wel hebben we ervoor gezorgd dat we de Constructors van de classen en de custom methods wel hebben gecomment.

# **Analyse Klassendiagram**

## Beschrijving

Ons analyse diagram is geschreven voordat we begonnen zijn met programmeren, het is geschreven om een idee te krijgen voor onze programmastructuur, welke attributen en operaties een class kan bevatten en hoe de relaties liggen tussen de classes. Dit diagram is expres heel wat versimpeld en stelt daardoor onze basis visualisatie voor van het begin van dit project.

Zoals u kan zien heeft de speler, in het engels genaamd “player”, 0 of 1 key, hier proberen wij ons te houden aan de casus (zie requirements) van dit project. Onder de map ziet u dat de map uit honderd vakjes (in het engels genaamd “tiles”) bestaat, er zijn zes soorten tiles die in het engels staan (Start, Empty, Wall, End, Barricade, Key). Met elk hun eigen karakteristieken. Elke tile heeft ook een ImageIcon, een positie (x, y) op de map, en een standaard grote van 10x10 map.

De key tile heeft een pincode die het deelt met de tile class en bij key class als attribuut “pickedUp” zorgt ervoor dat de key van het tile afhaalt samen met de “removeKey()” operatie die player accessed. En volgens de casus (zie requirements) wanneer de player al een sleutel heeft en er nog een sleutel probeert op te pakken zal remove key worden aangeroepen samen met pickup Key zodat de nieuwe sleutel in de inventaris gaat van speler en oude sleutel wordt weggegooid.

De barricade heeft ook een waarde (pincode) en wanneer de speler op de positie van de desbetreffende barricade staat en als de waarde van barricade niet overeenkomt met de waarde van key pinCode dan wordt er vervolgens een waarschuwing vertoond in de console. Wanneer de key wel overeenkomt met de barricade wordt de methode remove barricade aangeroepen en zal de barricade openstaan/weg gaan.

Verder is er nog een start tile waar de speler begint. Een empty (een normaal vakje) waar de speler op kan lopen, wall tile waar de speler niet op kan staan. Een end tile die kijkt naar de positie van de speler en als de speler op deze tile bereikt heeft, wordt er een bericht getoond voor het halen van het level.

# **Design Diagram**

## Beschrijving

### In het kort

*Main kan bestaan uit 0 of meerdere Menu’s => Menu 1-1 met Board => Board 1-1 met Map => Map kan bestaan uit 64-144 met Tile => Tile heeft een generalisatie met de subklassen => Wall ,Empty, Barricade en Finish <= Item heeft dan weer een aggregaat relatie met Empty, ook is Item onafhankelijk van Inventory en heeft een Inventory normaal gesproken 0 of 1 Item => Player heeft dan vervolgens dan 1 Inventory en Inventory 1 Player. En tot slot heeft Player een interactie met het Board middels de interface “ArrowKeyListener” class. Deze roept de move() methode aan om mogelijk de Player om van Tile te verplaatsen.*

### Toelichting Constants

Alle resources die in dit project worden gebruikt komen uit de constants class denk hierbij aan: mappen, foto’s voor menu’s, textures voor tiles, kleuren en allerlei andere static waarden. De constants class wordt in elke class geïmplementeerd.

Zoals u kunt zien is ons design diagram redelijk van formaat, dus om het overzichtelijk te maken lopen wij het proces af hoe de computer het ook zou aflopen. We beginnen daarom met de *main* class waaruit de code begint.

De *main class* heeft nul of meerdere *menu’s* via de *menu class*, de *menu class* paintde achtergronden van het hoofdmenu aan de hand van de *inner class ‘JPanelMain’* de gifs/foto’s hiervoor worden gehaald uit de *constants class*.

De *menu class* heeft een één-op-één relatie met de *board class*. In de *board class* wordt altijd al een standaard tutorial map gegenereerd omdat in *menu* een nieuwe instance van board wordt aangemaakt. Dit wordt duidelijk als er op de *“play”* knop wordt gedrukt in het beginscherm en de main menu niet meer zichtbaar is. Deze standaard tutorial map wordt gehaald uit de *constants class*.

Wat ons brengt bij de *map class* deze class genereert de textures voor de map die deze zoals bij alle andere resources uit de *constants class* haalt.

De *map* class zelf heeft een relatie van 64 tot en met 144 met de *tile class* (dit is omdat we verschillende map groottes hebben). Elke tile in de *map class* krijgt zijn eigen corresponderende texture die ingelezen wordt via de *constants class*. Hij herkent de soort tile aan de enum naam van de soort tile. De verschillende soorten tiles hebben andere gedragsregels, bijvoorbeeld dat over een wall niet gelopen kan worden. Daarom is voor elke soort *tile* een subclass die dit afhandelt per tile soort. Elke soort tile heeft ook een eigen x en y als coördinaat/positie.

Dit is vooral belangrijk voor *finish class* en *empty class* waar de method *specialevent()* overschreven wordt. Bij *finish* wordt er victory scherm met statistieken getoond. Bij de *empty*

*tile* wordt de methode gebruikt om een key van de map af te halen (*removeItem()*) en een key oppakken wordt gedaan met de methode (*pickupItem()*) in de *Player class*.

De tile soort *barricade* heeft een value die nodig, die gebruikt wordt voor het openen van een *barricade* dit wordt gedaan door een interactie met de player, de methode die hiervoor gebruikt wordt om te kijken of dit mogelijk is, is *navigable()*. Deze wordt overschreven in de *barricade* class en returned *true* of *false* als dit *true* is dan verandert het icoontje van het *barricade*, bij *false* geeft het een melding dat de speler een verkeerde sleutel heeft in het console.

Dan komen we tot slot tot de laatste subclass van tile: de *wall class*. Deze class is vrij vanzelfsprekend, walkable staat altijd op false en verandert nooit.

Dan komen we aan bij de *Item class* de relaties van deze class is empty en inventory en is een aggregaat relatie. Dat betekent dat item kan bestaan zonder inventory of zonder een (empty) tile. Voor nu heeft de *item class* maar één item: een sleutel (genaamd *key*). Toch hebben we de class als *Item* genoemd, dit is gedaan zodat het mogelijk is om het spel later verder uit te breiden. Deze class kunnen we gebruiken om later andere soorten items aan te maken.

De *Item class* heeft een één op één aggregerende relatie met *Inventory class*. Zoals u kunt zien is de inventory verantwoordelijk voor het bijhouden van welke items de player verzamelt heeft. Dit inventory systeem is een uitbreiding op de standaard opdracht samen met de mogelijkheid dat de speler niet gelimiteerd is op het vasthouden van maar een item. Ook verwijdert de *inventory class* de oudere sleutel wanneer een andere sleutel is opgepakt (als deze optie niet geselecteerd is). *Inventory* voegt items van *player* toe via de method *addItem()****.***

*Inventory* heeft dan weer een 1 op 1 relatie met de *player* *class*. Zoals u al verwacht van de naam heeft de *inventory class* een arraylist die de items op slaat. Het houdt ook een counter bij van hoeveel *items* er verzamelt zijn wat wordt getoond in een label op het menu.

En tot slot de *player class*, *player* heeft interactie met het *board* middels een *actionlistener* die luistert naar de pijltjestoetsen, deze veranderen ook de richting waar het *Player* naar zou kijken aan de hand van welke pijltjes toets is ingedrukt.

# **Unit Tests Code Coverage**

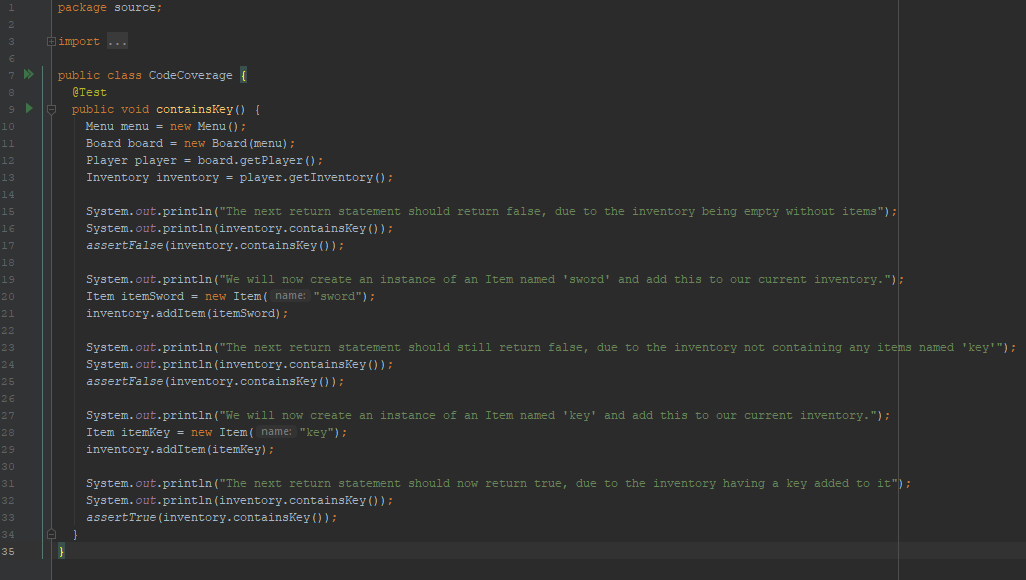
## Code Coverage van Inventory Contains Key() method test

Voor de unit test van code coverage hebben we de *inventory* contains key method getest.

Hiervoor hebben we een nieuw *menu* met nieuw *board* en *player* aangemaakt. Bij het aanmaken van een *player* wordt er ook een *inventory* voor de *player* gemaakt. Eerst hebben we de methode containsKey() geprint die zou false moeten returnen want inventory bevat nog geen item.

Vervolgens hebben we een nieuwe item ‘sword’ aangemaakt en de addItem(item) method aangeroepen om deze toe te voegen aan de items arraylist van Inventory. Vervolgens gebruiken we de methode containsKey() en print return statement zou false moeten zijn want inventory bevat wel een item, maar niet genaamd “key”.

Vervolgens hebben we een nieuwe item ‘key’ aangemaakt en de addItem(item) method aangeroepen om deze toe te voegen aan de items arraylist van Inventory. Vervolgens gebruiken we de methode containsKey() en print return statement zou true moeten returnen.



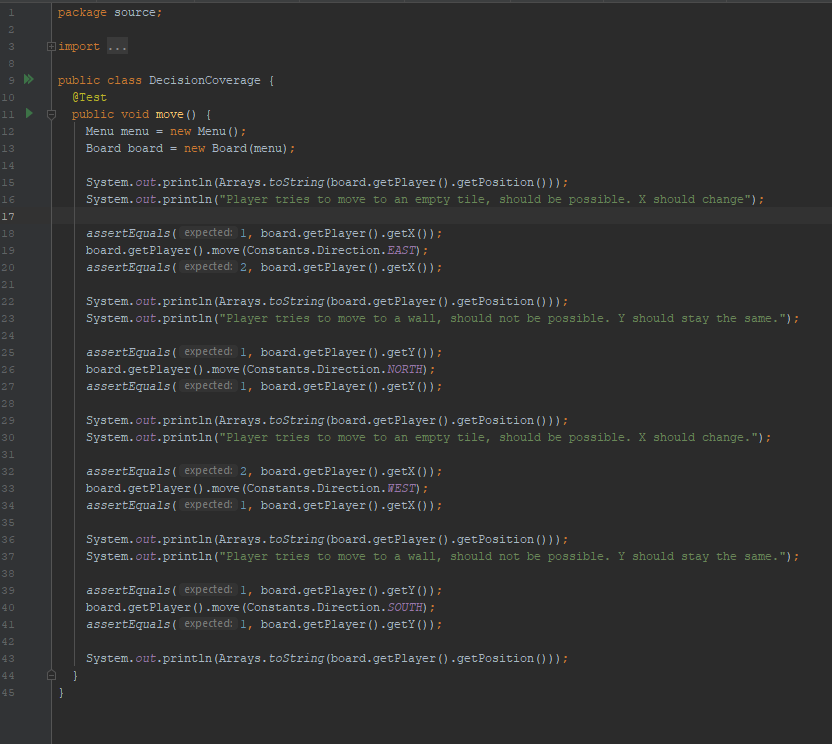
# **Unit Test Decision Coverage**

## Decision Coverage Player move() method test

Voor een unit test van decision coverage hebben we besloten om de player move() method getest. In de move functie maken we gebruik van de enum ‘Directions.’

NORTH = naar boven, EAST = naar rechts, SOUTH = naar beneden, WEST = naar links.

Zoals u kunt zien in de foto hebben we een nieuw *menu*, gemaakt met een nieuwe *board*. Met het *board* wordt de *player* gemaakt. Vervolgens gingen we de player move() functie testen door de player met methods te laten bewegen. We hebben *assertEquals* gebruikt om te kijken of de positie van de *player* is zoals wij dat verwachten. We wisten dat de *player* wel naar rechts kon, maar niet naar boven en naar beneden (want bij het creëren van een nieuwe board wordt bij default de tutorial map aangemaakt).



# **Program Extensions**

## Dark Mode

Het eerste dat bij ons op kwam qua uitbreidingen was een dark mode. Dit zorgt ervoor dat mensen die in het donker spelen minder last krijgen van hun ogen. Ook vinden veel mensen het prettig als er een knopje is om het Thema van de frame donkerder te hebben (gezamenlijk hebben wij besloten om liever een Dark Mode aan te maken).

### Requirements van Dark Mode

* Er moet een checkbox beschikbaar zijn om te switchen tussen Light en Dark Mode.
* Alle Panels moeten een ‘Re-color’ krijgen bij het checken en unchecken van de Dark Mode Checkbox.
* Alle Labels moeten een ‘Re-color’ krijgen bij het checken en unchecken van de Dark Mode Checkbox.
* De Dark Mode moet geen invloed hebben op het speelveld (dit kan namelijk ongewenste gevolgen hebben over de zichtbaarheid van het speelveld).
* Het Main Menu van de game moet ook een Dark Mode functie hebben.

## Texture Packs

Nu we toch bezig waren met aanpassen van het Thema van onze game, waarom niet een texture pack functie toevoegen. Dit zorgt voor diversiteit in hoe het speelveld eruit zal zien.

Hierdoor kijk je niet de hele tijd hetzelfde thema.

### Requirements van Texture Packs

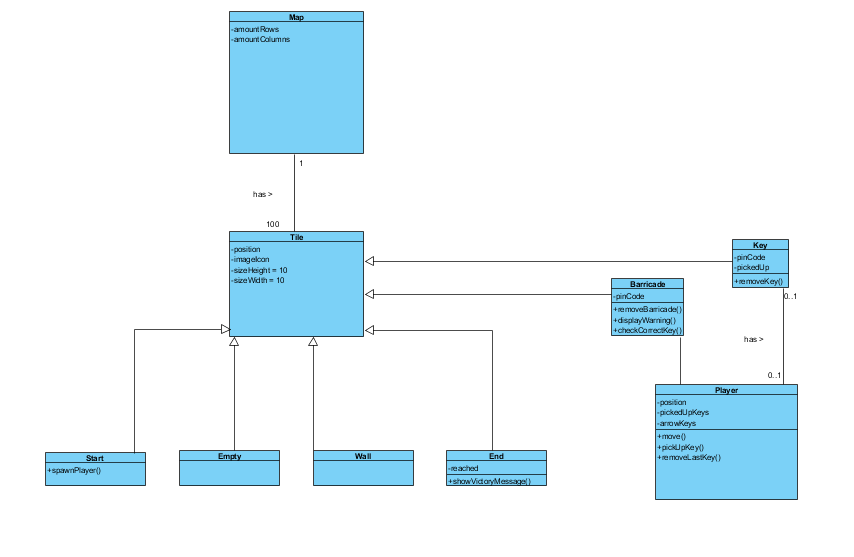
* Empty, Barricade, Wall en Player moeten een andere image krijgen bij het veranderen van de daarbij behorende texture pack.
* De speler kan zelf een Texture Pack kiezen.
* Texture Packs moeten met 1 klik op de knop aanpasbaar zijn (Combobox).
* Texture Packs moeten geen invloed hebben op de werking van het doolhof, ze dienen alleen voor visuele aspecten te werken.
* At launch willen we minimaal 3 verschillende Texture Packs.

## Level Statistics

Om het spel wat uitdagender te maken hebben we voor elk level statistics toegevoegd. Hierbij kun je denken aan een stappenteller, de hoeveelheid sleutels opgepakt etc. Dit zorgt ervoor dat de speler het best uit zichzelf kan halen door zijn aantal stappen bijvoorbeeld te verminderen.

### Requirements van Level Statistics

* De statistics die we minimaal werkend willen hebben zijn: een moveCounter, een pickedUpKeysCounter en een barricadesOpenedCounter.
* Ook moeten de Level Statistics terwijl het spel bezig is moeten kunnen updaten, bijvoorbeeld als de speler een stap zet moet de moveCounter met 1 omhoog gaan.
* Als de speler op restart klikt moeten de Level Statistics gereset worden.
* Als de speler de finishline heeft bereikt moet er een scherm komen die de Level Statistics laat zien van het bijbehorende doolhof.



## 