|  |
| --- |
| KeyBarricade – De Ontwikkeling |
| Door Matthias Aarnoutse, Jeffrey Dufour en Ewoud Vermeij |
|  |
| Begeleider: Okan Zor. Groep: 2. Klas: SE-OO-2. |
|  |

Inhoud

1. Inleiding.
2. Eisen aan het systeem.
3. Het analyse klassendiagram.
4. Het design klassendiagram.
5. JUnit.
   1. Code coverage.
   2. Decision coverage.
6. Conclusie.

1. Inleiding

In opdracht van de opdrachtgever hebben wij een spel moeten ontwikkelen. In dit document vindt u informatie over de ontwikkelingen hiervan in stappen terug. Zo wordt eerst vastgesteld wat de precieze eisen zijn aan het systeem. Er wordt een analyse klassendiagram opgesteld voor de business. Vervolgens wordt er een design klassendiagram opgesteld voor de ontwikkelaars en als laatste treft u een aantal JUnit testen aan. Uiteraard vind u een uitgebreide uitleg per onderdeel aan.

Verder willen wij vermelden dat wij dit spel met de waterfall stijl hebben ontwikkeld.

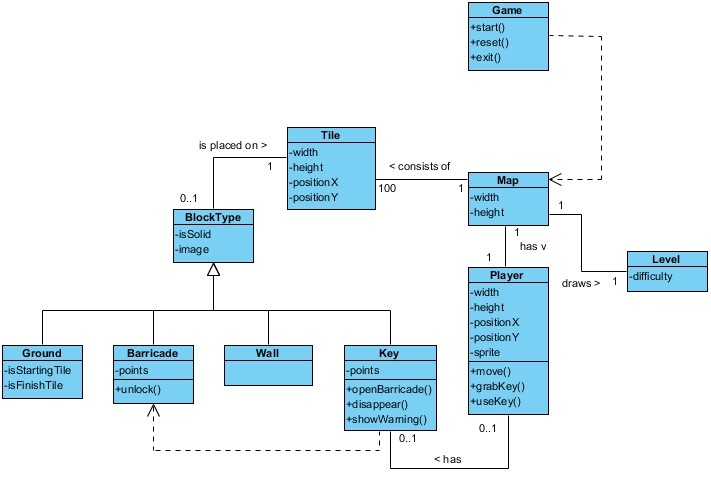
2. Eisen aan het systeem

Om het spel op een correcte wijze de ontwikkelen is het van belang om de eisen van de opdrachtgever bij elkaar te verzamelen. Uit die eisen moet een compleet beeld ontstaan over hoe het spel moet werken en hoe de gebruiker het spel in gebruik zal nemen. Om voor een degelijk, overzichtelijk en compleet beeld te zorgen zult u hieronder alle eisen nauwkeurig geformuleerd terug vinden.

1. Het spel bestaat uit een vierkant speelveld.
2. Het doel van het spel is om van punt a naar punt b te lopen.
3. In het speelveld bevinden zich muren.
4. In het speelveld bevinden zich barricades die elk een aantal punten bezit.
5. In het speelveld bevinden zich sleutels die elk een aantal punten bezit.
6. Een sleutel kan een barricade openen als het dezelfde aantal punten bezit als de barricade.
7. Bij het openen van een barricade zal deze verdwijnen van het speelveld.
8. De speler kan een sleutel oppakken.
9. De speler mag maar 1 sleutel bij zich houden.
10. Als de speler een sleutel oppakt, verdwijnt deze van het speelveld.
11. Als de speler een sleutel pakt terwijl de speler een sleutel bij zich had, verdwijnt deze. De speler bezit nu de nieuw opgepakte sleutel.
12. De speler kan de sleutel niet terugzetten op het speelveld.
13. Sleutels kunnen oneindig op barricades gebruikt worden, mits het aantal punten van de sleutel en barricade overeenkomen.
14. Bij het niet passen van een sleutel verschijnt een melding op het scherm.
15. De speler beweegt zich door het speelveld doormiddel van de pijltjestoetsen.
16. Het moet mogelijk zijn voor de gebruiker om exact hetzelfde speelveld opnieuw te spelen.
17. Als de speler het eindpunt heeft bereikt, komt er een melding in het scherm dat de gebruiker het spel heeft gewonnen.
18. Muren, barricades en sleutels zullen willekeurig in het speelveld worden gegenereerd.
19. Niet ieder speelveld heeft een oplossing.
20. Het spel moet verschillende moeilijkheidsgraden bevatten.
21. Het ontwerp van het spel moet geschikt zijn voor uitbreidingen.

3. Het analyse klassendiagram

Om voor de business een beeld te schetsen hoe het spel in elkaar steekt en hoe “objecten” met elkaar om gaan hebben wij een analyse klassendiagram opgesteld. Figuur 1 laat een afbeelding zien van het analyse klassen diagram. Op de volgende bladzijde wordt uitleg gegeven over alle klassen, attributen, operaties, associaties en dependencies.

****

Figuur

Wij hebben ervoor gekozen op om het speelveld te laten bestaan uit honderd vakjes. De klasse **Map** omvat het speelveld. De klasse **Tile** omvat het vakje. Dit verklaard ook de associatie tussen Map en Tile. De Map “consist of” (bestaat uit) honderd Tiles. Een Tile zal zich altijd bevinden op een Map. Zowel Map als Tile heeft een breedte (width) en een hoogte (height). Tile heeft echter ook een positionX en een positionY om de locatie op de map te bepalen.

In het spel komen muren, barricades, sleutels en grond voor om op de lopen. Deze potentiele klassen hebben allemaal de overeenkomst dat ze op een Tile komen te liggen, maar hebben verschillende eigenschappen. Daarom hebben wij ervoor gekozen om de klassen **Ground, Barricade, Wall** en **Key** een overerving te maken van de klasse **Blocktype**. Alle klassen die overerven van Blocktype zijn solid of niet. Hiermee wordt bedoelt of de player er doorheen kan lopen of niet. Zullen alle klassen die overerven van Blocktype een image hebben waarmee ze goed herkent kunnen worden.

Ground heeft als unieke attributen isStartingTile en isFinishTile. Hiermee wordt bedoelt dat er een Ground voorkomt in het spel waar de player begint en waar de player moet eindigen om het spel te beëindigen.

Barricade heeft als uniek attribuut points en als unieke operatie unlock. Points zorgen er voor dat de Barricade alleen geopend (unlock) kan worden met een in punten overeenkomende sleutel. Unlock spreekt voor zichzelf.

Wall heeft geen unieke attributen of operaties. Toch hebben wij besloten er een klasse van te maken. Een Wall heeft namelijk een unieke image en zal altijd solid zijn. Wall overerft dit van Blocktype. Vandaar dat het niet wordt aangegeven in de klasse Wall zelf.

Key heeft als uniek attribuut points en als unieke operaties openBarricade, disappear en showWarning. De points dienen voor het openen van barricades met hetzelfde aantal punten. Met de operatie openBarricade kan een barricade worden geopend, mits de punten overeenkomen. Komen de punten niet overeen dan zal de operatie showWarning een melding geven dat de sleutel niet past. Als de player de sleutel oppakt zit deze in zijn zak en zal de operatie disappear ervoor zorgen dat de sleutel van het veld verdwijnt. Key heeft een dependency met Barricade, omdat de sleutel bepaalde barricades kan openen. De sleutel hoeft niet te weten welke barricade het is, alleen het aantal punten. Vandaar de keuze van de dependency;

Vanwege het feit dat Ground, Barricade, Wall en Key overerven van Blocktype, heeft Tile alleen een associatie met Blocktype nodig. Voor een Tile is het namelijk alleen belangrijk te weten of de Blocktype solid is en welke afbeelding die met zich mee heeft. Een Tile bevat ook altijd 1 Blocktype en een Blocktype is altijd geplaatst op een Tile.

De klasse **Player** heeft de attributen width, height, positionX, positionY en sprite. Width en height bepalen de hoogte en de breedte van de speler. PositionX en positionY geven aan waar de speler zich in de Map bevind. Sprite is de afbeelding van de player die de player herkenbaar moet maken.

Player bevat de operaties move, grabKey en useKey. Met de operatie move kan de player zich voortbewegen in de Map. Met de operatie grabKey kan een sleutel in het speelveld worden opgepakt. Met de operatie useKey kan de player een sleutel gebruiken op een barricade.

Player heeft een associatie met Key, omdat deze wordt opgepakt en de player altijd moet weten welke sleutel hij in zijn zak heeft. Een Key hoort bij 0 of 1 player en een player kan 0 of 1 Key bij zich houden.

Player heeft ook een associatie met Map. Een Map bevat altijd 1 player en een player bevind zich altijd op een Map. Vanwege de coördinaten is een associatie nodig en voldoet een dependency niet.

De klasse **Level** bevat als enige attribuut difficulty en een associatie met de klasse Map. Met de difficulty wordt de moeilijkheidsgraad van een Level bepaald. Doordat Level een associatie met Map heeft kan de map met de difficulty van level bepalen hoe alle Tiles met Blocktypes moeten worden neergezet.

Als laatste de klasse Game. Deze heeft de operaties start, reset en exit en heeft een dependency naar Map. De operaties van Game geven aan Map door of er een spel moet stoppen opnieuw moet starten of dat het spel moet worden verlaten.

Nu het analyse klassendiagram is behandelt zal de volgende stap het design klassendiagram zijn. Hierbij wordt alle technische benodigdheden voor het spel gemodelleerd. Dit zult u aantreffen in het volgende hoofdstuk.