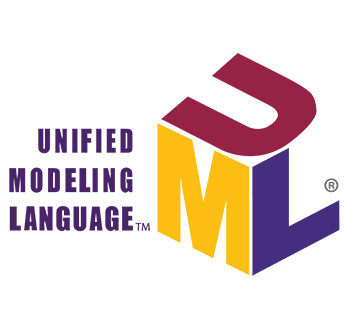
Ontwikkeldocument





Namen: Jasper Schinkel, Joost Simons en Daniel Zwamborn

Klas: SE 1D

Docenten: Cobie van der Hoek

Rob van der Krogt

Inleiding:

In dit document zal kort beschreven worden wat de eisen waren voor het te maken spel, hoe de opgestelde analyse en design klassendiagrammen eruit zien en waarom deze diagrammen ontworpen zijn. Tot slot komen de unittesten aan bod die gedaan zijn over de “Move” klasse.

**Beschrijving van het spel:**

Het spel *"Sleutelbarricade"* is een spel waar de speler zich in een doolhof van 10 bij 10 vlakjes bevindt. Er zijn verschillende soorten vakjes:

1. Grass, dit is een normaal vakje waar je altijd op kan lopen.

2. Wall, dit is een vakje waar je niet doorheen kan lopen.

3. Barricade, dit is een vakje dat met een sleutel opengemaakt kan worden.

4. Key, dit is een vakje dat je een sleutel geeft voor een barricade.

5. Finish, dit is het vakje waarop je moet gaan staan om te winnen.

Het doel van het spel is om de finish te bereiken.

Doordat er muren en barricades in het spel zijn, gaat dat niet zo makkelijk.

Muren zijn objecten waar de speler niet doorheen kan. Hij zal dus een andere route moeten vinden.

Barricades zijn objecten waar de speler doorheen kan mits hij een sleutel van dezelfde waarde heeft. Er zijn 3 soorten barricades:

1. Barricade (waarde 100), deze barricade heeft een sleutel (waarde 100) nodig.

2. Bush (waarde 200), deze barricade heeft een heggenschaar (waarde 200) nodig.

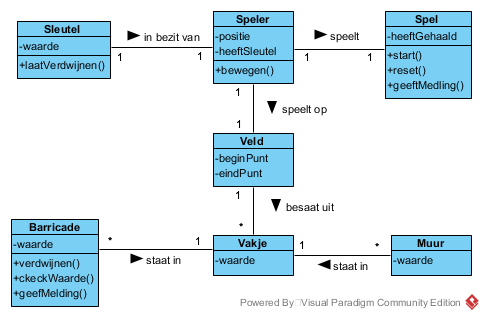
3. Stump (waarde 300), deze barricade heeft een kettingzaag (waarde 300) nodig.

Sleutels passen bij een bepaalde barricade en kunnen net zo lang gebruikt worden totdat er een nieuwe wordt opgepakt, de vorige sleutel verdwijnt dan uit het spel. Wees dus voorzichtig met sleutels want je hebt het risico om vast te lopen.

Als de speler naar een plek wilt waar hij niet naar toe kan, komt er onderaan het scherm een melding te staan.

Besturing: Om de speler de bewegen gebruik je de pijltjestoetsen.

Als je vast zit, druk je R in om het level te resetten.



Figuur 1. Analyse diagram voor het gemaakte sleutelspel

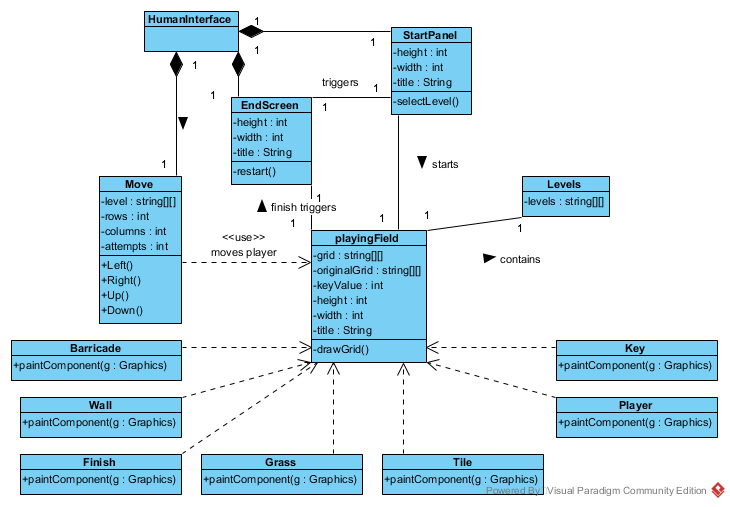
Toelichting:

In het ontwerp van de analyse klassendiagram is er in principe vanuit gegaan dat er maar een select aantal klassen gemaakt zouden worden. Een aantal van deze klassen zouden ook als objecten gedefinieerd worden, zoals de barricade, de sleutel, de muur en de speler. Deze objecten zouden dan hun eigen waardes krijgen om aan het speelveld aan te geven wat zij zijn.

De speler moest uiteraard kunnen lopen en een sleutel in zijn bezit kunnen krijgen. Voor de sleutels en barricades was het ook de bedoeling dat zij acties kregen. Sleutels moesten verdwijnen nadat zij opgepakt werden en barricades moesten ook verdwijnen nadat er een interactie plaatsvond tussen de speler en de barricade met de juiste sleutel.

Het speelveld zelf had nog een losse definitie wat simpelweg “Veld” genoemd werd. Hierin zou dan het speelveld gedefinieerd worden, met het beginpunt en het eindpunt voor de speler. In het uiteindelijke programma is dit veel uitgebreider geworden.

In het analyse diagram was het plan ook om alle knoppen in een klasse te stoppen, wat het spel zou worden, om zo het spel zo simpel mogelijk te houden. Ook was het de bedoeling dat de status van het spel bijgehouden zou worden in de klasse van het Spel, zodat het spel met deze informatie een melding kan geven om aan te geven dat het spel voorbij was.



Figuur 2. Design diagram voor het gemaakte sleutelspel

Toelichting:

Het uiteindelijke design diagram is heel anders geworden dan het analyse diagram. De klasses voor de objecten definiëren hier de graphics, dus hoe deze eruit zien, voor het spel. Deze worden nergens anders gedefinieerd dan in het playingField, dus deze zijn afhankelijk van deze klasse om te bestaan.

Het grid van het speelveld word gedefinieerd in playingField en gewijzigd in Move. Deze wijzigingen worden ook doorgegeven aan de klasse playingField, zodat de speler de wijzigingen kan waarnemen. De Levels worden geladen door het StartPanel, en de EndScreen gaat uiteindelijk terug naar het StartPanel.

In de klasse PlayingField word ook bijgehouden welke sleutel opgepakt is. Wanneer een andere sleutel opgepakt word, dan overschrijft deze de vorige sleutel.

Menselijk input kan de Move klasse beïnvloeden en de Move klasse beweegt de Speler op het Speelveld. Ook kiest de mens het level in het StartPanel en kan de mens terug gaan naar het StartPanel via de klasse van de EndScreen.

Dit diagram laat wel erg zien dat de klasses van PlayingField en Move veel attributen waarborgt. Voor een schonere design diagram had dit wellicht beter verspreid kunnen worden over meerderen klassen.

Unittesten:

Om te testen of de code wel goed werkt moet de code getest worden. Dit kan door middel van JUnit test. Een JUnit test wordt vaak toegepast bij Test Driven Development (TDD). Hier wordt er naar verschillende methodes uit het programma gekeken. het doel ervan is om te kijken als het programma naar behoren werk. We moesten Decision Coverage maken. Dit bestaat uit vele if en else statements. We hebben gekozen om de class “Move” te testen omdat dat onze belangrijkste class is. De speler kan de pijltjestoetsen gebruiken en hier zijn 4 methodes voor:

* public void left()
* public void right()
* public void up()
* public void down()

Voor elke methode wordt er gekeken als de speler naar het volgende vakje kan lopen. Eerst wordt er gekeken als het volgende vakje binnen het veld ligt en als het volgende vakje geen wall is. Daarna kijkt hij naar alle mogelijke zetten en als die mogelijk zijn kan de speler er naartoe. Om te testen als dit werkt is er een simulatie gemaakt waarbij de speler verschillende kanten opgaat en uiteindelijk op positie (4,2) moet uitkomen. In de simulatie wordt er vermeld dat de speler niet alle bewegingen kan uitvoeren en komt hij uiteindelijk op positie (4,2): test geslaagd!



Figuur 3. Code Coverage van de JUnit Test