Quebble

Software Design Description

v2.0
10 juni 2021

ITA-OOSE-A-f 2020
Rowan Paul Flynn 637588
Alex Cheng 634967

OOSE-OOAD
Docent: Marco Engelbart

Inhoudsopgave

Inleiding

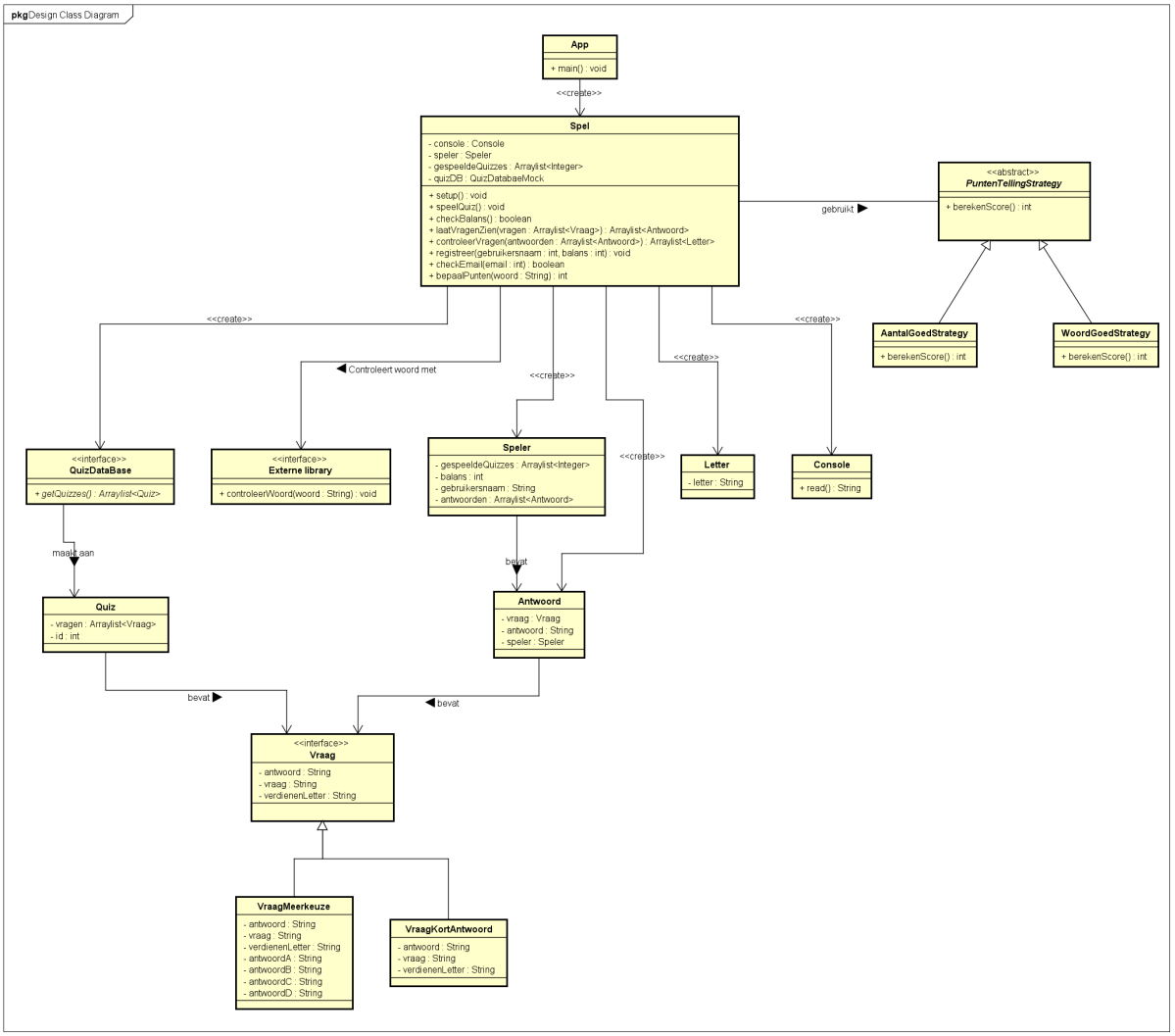
Het bedrijf Solid Games wil een quiz-applicatie Quebble ontwikkelen. Quebble bevat een grote verzameling quizzen die elk uit acht vragen bestaan. Elke quiz is een mix van meerkeuzevragen (met steeds vier alternatieven) en kort-antwoord-vragen. Als alle vragen zijn beantwoord dan worden de vragen gecontroleerd en voor elk correct antwoord wordt er een letter gegeven. Met deze letters maakt de speler een woord, waarvoor punten correct worden toegekend.

Met dit document wordt een overzicht van het design van Quebble en de keuzes die daarin gemaakt zijn. Voor meer informatie over wat de software gaat doen en hoe verwacht wordt dat het zich gedraagt zie het Software Requirements Specification document.

1) Design Description

In dit hoofdstuk staat een gedetailleerde omschrijving van alle softwarecomponenten aan de hand van diagrammen.

1.1 Design class diagram



Hieronder staat het design class diagram voor de Quebble applicatie. De applicatie start in de *App* klasse, die een nieuw *Spel* aanmaakt.

Als een gebruiker wilt registreren dan wordt de methode *registreren()* uitgevoerd die zelf weer de methode *checkEmail()* aanroept om de email te checken. Wanneer de email uniek is, dan wordt er een nieuwe speler aangemaakt de gebruikersnaam en balans van 1000.

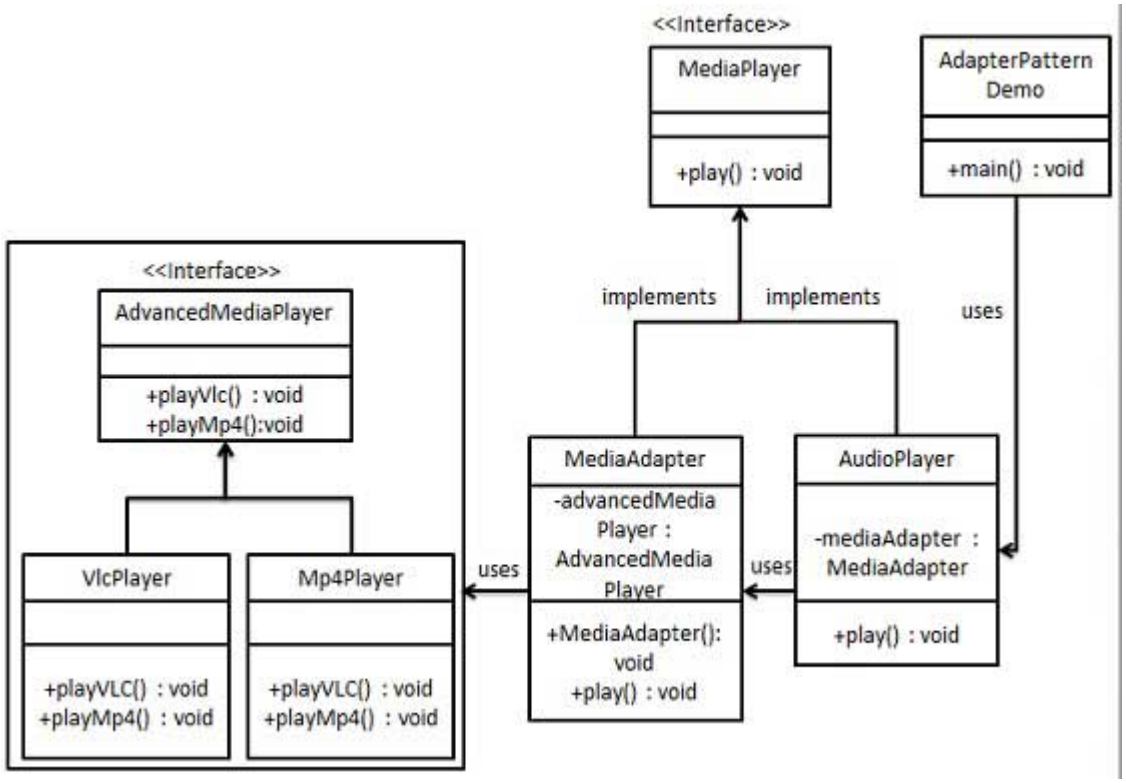
Wanneer een gebruiker een nieuwe quiz start wordt de balans van de speler gecheckt en wordt de *speelQuiz()* methode gestart. De *speelQuiz()* methode roept de *nieuweQuiz()* methode aan die de *Databaselaag* aangeroept die de quizzes uit de database ophaalt en Quiz maakt met de database data.

Vervolgens wordt voor elke vraag de vraag getoond, een nieuw *Antwoord* gemaakt met het antwoord van de speler en opgeslagen in *Speler*. Wanneer alle vragen zijn getoond worden alle antwoorden gecontroleerd met *controleerVragen()*, voor de vragen die goed zijn, de te geven letter teruggegeven. De speler kan nu een woord opgeven die gecontroleerd wordt door de *Externe Library*. Als deze goed is dan wordt krijgt de speler een bepaald aantal punten.

De puntentelling maakt gebruik van een strategy pattern om meerdere implementaties van de puntentelling mogelijk te maken. Dit is gedaan omdat Solid Games aangegeven heeft gemakkelijk te willen overstappen op een andere puntentelling.

Voor de *Externe Library* is gebruik gemaakt van een Adapter Pattern. De applicatie spreekt een interface aan die ongeacht welke library erachter zit hetzelfde teruggeeft. Dit maakt het switchen van library een stuk gemakkelijker want er hoeft niet elke keer code veranderd te worden behalve in de interface.

Eigenlijk kan het gezien worden als een soort brug tussen twee libraries, met een Adapter library is er een hoofd interface waardoor je dus kan kiezen welke library je wilt gebruiken. Deze pattern is een handige oplossing voor wanneer een klasse een bepaalde interface verwacht voor een methode, maar deze service of klasse een andere interface gebruikt.

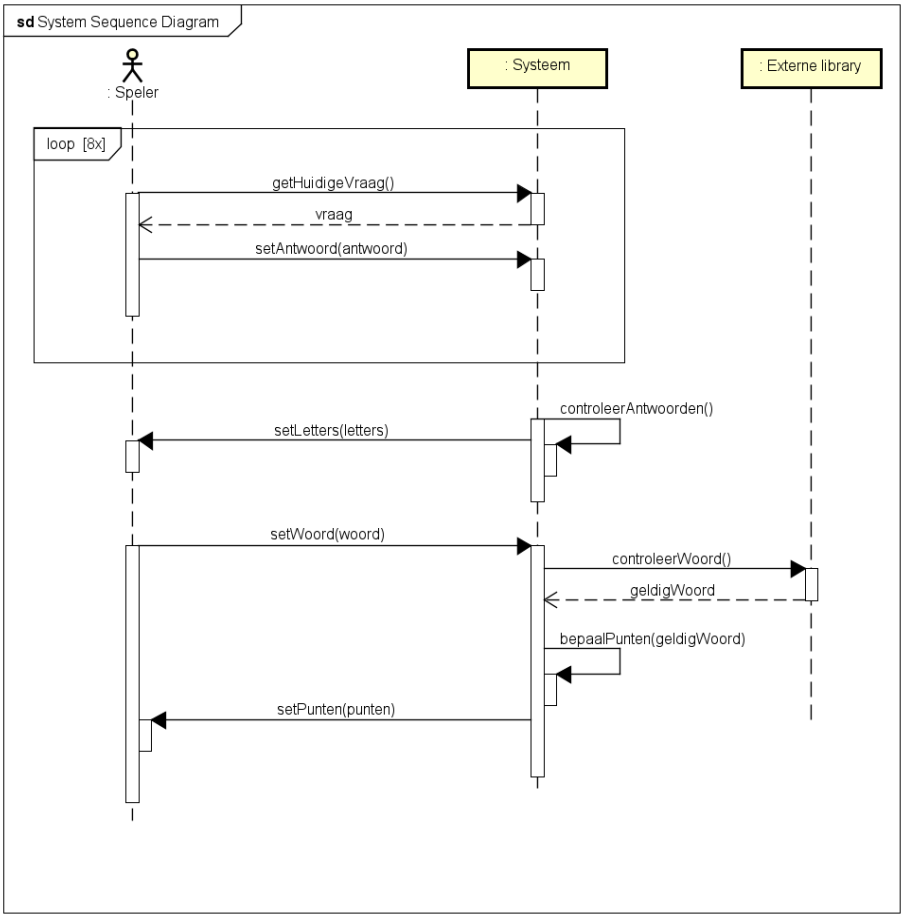


Voorbeeld van Adapter Pattern in Java

2) Sequence Diagrams

De sequence diagrams geven weer hoe het systeem omgaat met één bepaalde actie.

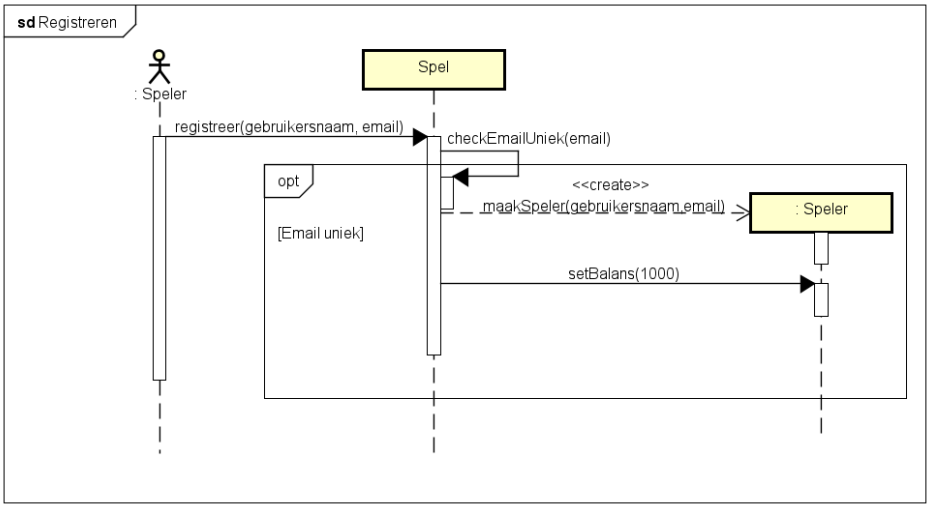
2.1 System sequence diagram



Het system sequence diagram laat de algemene flow van het spelen van de applicatie zien. De speler krijgt de vraag te zien, beantwoord die en geeft het antwoord terug aan de applicatie. Als dat 8 keer gebeurd is worden alle antwoorden gecontroleerd en krijgt de gebruiker letters terug voor de hoeveelheid antwoorden hij/zij goed had.

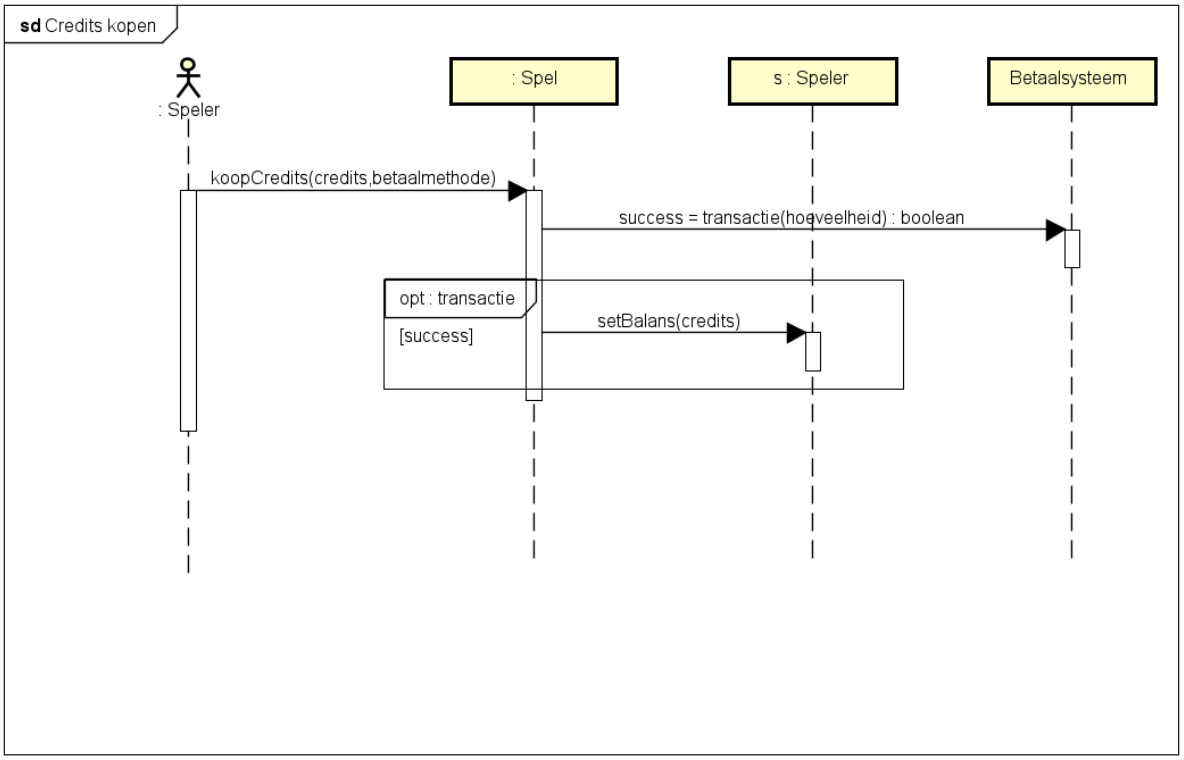
De gebruiker maakt een woord van die letters en het systeem controleert het woord met een externe library die aangeeft of het woord correct is. Op basis daarvan worden de punten bepaalt door het systeem en getoond aan de speler.

2.2 Registreren



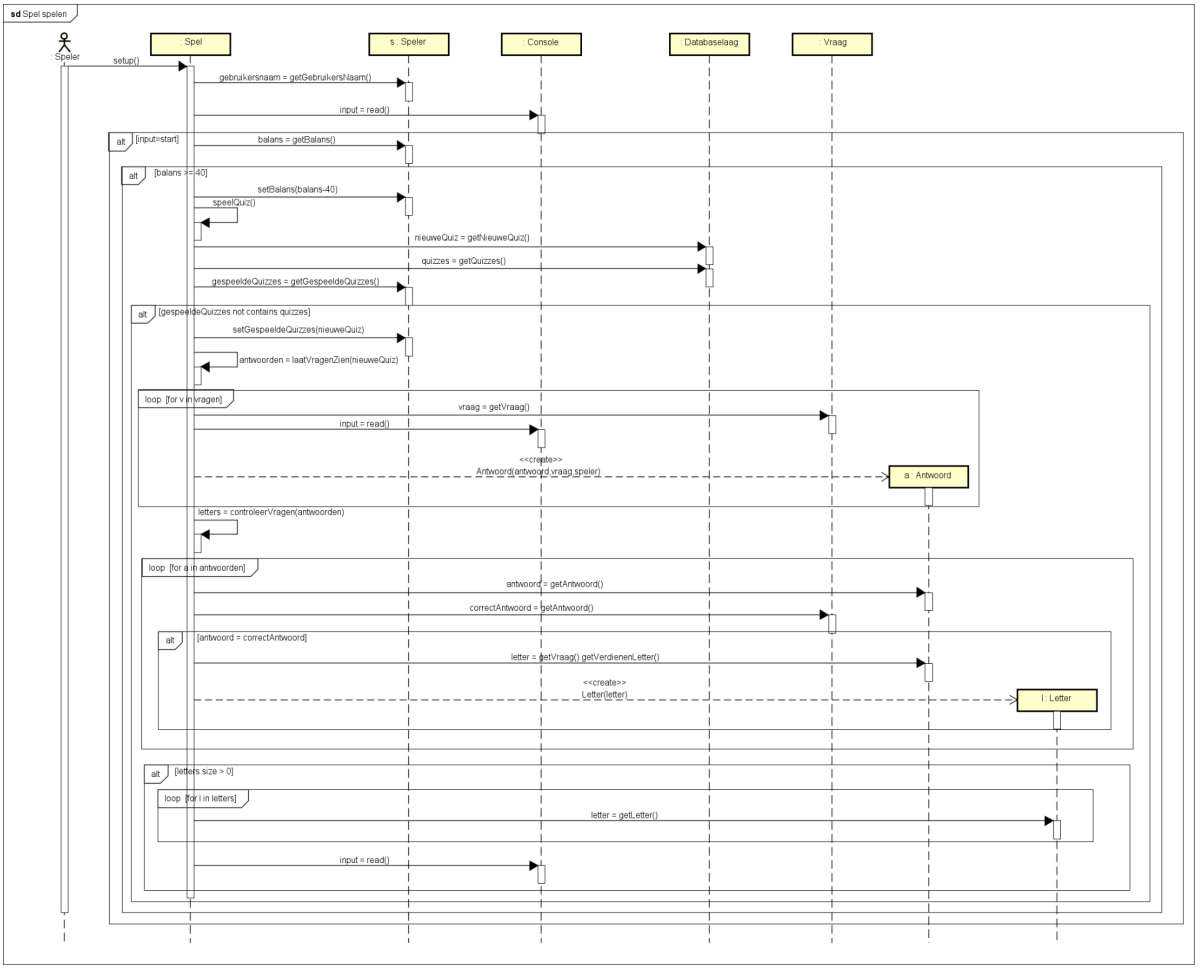
Een speler geeft aan te willen registreren met de email en gebruikersnaam die is opgegeven. Het systeem checkt of de email al in het systeem bestaat en maakt als dat het geval is een speler aan met de gegevens. Ook wordt het balans van de speler gelijk op 1000 gezet.

2.3 Credits bijkopen



Een speler kan ervoor kiezen om credits bij te kopen met echt geld. Dit wordt gedeeltelijk afgehandeld door een extern betalingssysteem.

2.4 Spel spelen



Hieronder staat het sequence diagram spel spelen. Deze sequence diagram bevat alles wat nodig is om het spel te spelen en is daarom ook zo groot. De methoden horen wel bij elkaar en het zou vreemd zijn om die op te splitsen.

Conclusie

In dit document zijn is het design van de Quebble applicatie besproken en uitgewerkt. Met een design class diagram en sequence diagrammen zijn de verschillende acties en klassen uitgelegd. Om erachter te komen wat de software gaat doen en hoe verwacht wordt dat het zich gedraagt kan je het software requirements specification document bekijken waarin dat wordt uitgelegd met behulp van domeinmodel en use cases.