

**Faculteit Bedrijf en Organisatie**Valentin Vaerwyckweg 1  
9000 GENT

ACADEMIEJAAR 2020-2021

OPLEIDING TOEGEPASTE INFORMATICA

# PROJECTWERK Opleidingsonderdeel ‘Software Development Project I’

*Rummikub*

**GROEP: 29**

**STUDENTEN: BEGELEIDER:**

*Victor Vanhoren Sonia Vandermeersch,*

*Adem Ayari Liesbeth Lewyllie*

*Quinten September*

*Milan Van Opstal*

Inhoudstafel

[Use case 1: aanmelden van bestaande gebruikers uit een databank 2](#_Toc72002342)

[Analyse 2](#_Toc72002343)

[Activiteitsdiagram 2](#_Toc72002344)

[Domeinmodel 3](#_Toc72002345)

[Ssd-diagram 4](#_Toc72002346)

[Databank gedeelte 5](#_Toc72002347)

[Java gedeelte 6](#_Toc72002348)

[DcD 6](#_Toc72002349)

[JAVA-code 6](#_Toc72002350)

[Use case 2: Speel spel 8](#_Toc72002351)

[Analyse 8](#_Toc72002352)

[Activiteitsdiagram 8](#_Toc72002353)

[Domeinmodel 9](#_Toc72002354)

[Ssd-diagram 10](#_Toc72002355)

[Java gedeelte 11](#_Toc72002356)

[DcD 11](#_Toc72002357)

[JAVA-code 12](#_Toc72002358)

[Use case 3: Speel beurt 13](#_Toc72002359)

[Analyse 13](#_Toc72002360)

[Activiteitsdiagram 13](#_Toc72002361)

[Domeinmodel 14](#_Toc72002362)

[Java gedeelte 16](#_Toc72002363)

[DcD 16](#_Toc72002364)

[JAVA-code 17](#_Toc72002365)

[Use case 4: Maak overzicht 20](#_Toc72002366)

[Analyse 20](#_Toc72002367)

[Activiteitsdiagram 20](#_Toc72002368)

[Domeinmodel 21](#_Toc72002369)

[Ssd-diagram 22](#_Toc72002370)

[Java gedeelte 23](#_Toc72002371)

[DcD 23](#_Toc72002372)

Figurenlijst

[*Figure 1: activiteitsdiagram use case 1* 3](#_Toc72002373)

[*Figure 2: domeinmodel use case 1* 3](#_Toc72002374)

[*Figure 3: ssd use case 1* 4](#_Toc72002375)

[*Figure 4: toevoegen van 4 test gebruikers* 5](#_Toc72002376)

[*Figure 5: overzicht van gegevens in databank* 5](#_Toc72002377)

[*Figure 6: verwijderen van kolom idSpeler* 5](#_Toc72002378)

[*Figure 7: voorlopig dcd use case 1* 6](#_Toc72002379)

[*Figure 8: testapplicatie use case 1* 7](#_Toc72002380)

[Figure 9: activiteitsdiagram use case 2 8](#_Toc72002381)

[Figure 10: domeinmodel use case 2 9](#_Toc72002382)

[Figure 11: ssd-diagram use case 2 10](#_Toc72002383)

[Figure 12: operation contract startSpel() 10](#_Toc72002384)

[Figure 13: DcD use case 2 op basis van het domeinmodel en ssd-diagram 11](#_Toc72002385)

[Figure 14: applicatie klasse use case 2 Bron: Visual paradigm 12](#_Toc72002386)

[Figure 15: activiteitsdiagram use case 3 13](#_Toc72002387)

[Figure 16: domeinmodel use case 3 14](#_Toc72002388)

[Figure 17: ssd diagrammen use case 3 15](#_Toc72002389)

[Figure 18: OC voor SSD use case 3 15](#_Toc72002390)

[Figure 19: DCD diagram use case 3 16](#_Toc72002391)

[Figure 20: overzicht gui klassen 17](#_Toc72002392)

[Figure 21: overzicht gui klasse StartUp 18](#_Toc72002393)

[Figure 22: voorbeeld gui use case 3 18](#_Toc72002394)

[Figure 23: voorbeeld gui spelbord en stenen 19](#_Toc72002395)

[Figure 24: Activiteitsdiagram use case 4 20](#_Toc72002396)

[Figure 25: domeinmodel use case 4 21](#_Toc72002397)

[Figure 26: ssd diagrammen use case 4 22](#_Toc72002398)

[Figure 27: DCD diagram use case 4 23](#_Toc72002399)

[Figure 28: code voor databank 24](#_Toc72002400)

[Figure 29: deel 2 code voor databank 24](#_Toc72002401)

[Figure 30: code voor GUI overzicht 24](#_Toc72002402)

Use case 1: aanmelden van bestaande gebruikers uit een databank

Analyse

Activiteitsdiagram

*Omschrijving*

Het activiteitdiagram heeft twee alternatieve verlopen. Het eerste alternatief bevindt zich in het begin. Er wordt namelijk gevraagd naar het aantal gebruikers indien dit niet het correcte aantal is wordt dit steeds opnieuw gevraagd tot het correcte aantal in ingegeven. Het tweede alternatief verloop speelt zich af op het einde van de use case, hier wordt gevalideerd of de gebruikersnaam en het wachtwoord voldoen aan de domeinregels. Wanneer een gebruiker aan de domeinregel voor aanmelden voldoet, is er een tweede decision node zodat ook aan DR\_aantal voldaan wordt. Binnen deze use case is het de bedoeling gebruikers aan te melden dit proces wordt steeds herhaald tot alle gebruiker zijn ingelogd.

*Praktijk*



*Figure 1: activiteitsdiagram use case 1*

Bron: Visual paradigm

Domeinmodel

*Omschrijving*

Het domeinmodel bestaat slechts uit één klassen. Een klasse gebruiker met twee parameters, de parameter wachtwoord en een parameter gebruikersnaam.

*Praktijk*



*Figure 2: domeinmodel use case 1*

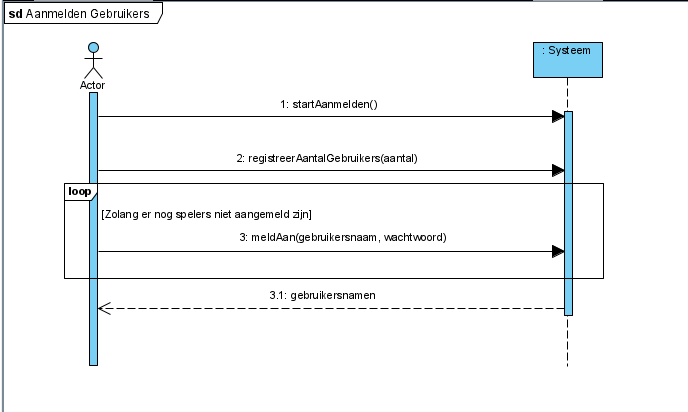
Bron: Visual Paradigm

Ssd-diagram

*Omschrijving*

Het ssd-diagram begint met een pijl startAanmelden() deze pijl wordt niet geïmplementeerd in de cui. Deze staat enkel op het ssd ter informatie. De tweede pijl registreerAantalGebruikers(aantal) stelt de registratie voor van het aantal gebruikers. Het aantal gebruikers wordt opgeslagen. Verderop is er een pijl meldAan deze is voorzien van een loop met de voorwaarde zolang er nog spelers niet aangemeld zijn. Op deze stap wordt volgens het geregistreerd aantal het correcte aantal gebruikers ingelogd. Verderop is er nog een terugkeer pijl, deze pijl heeft een lijstje van ingelogde gebruikers terug.

*Praktijk*



*Figure 3: ssd use case 1*

Bron: Visual Paradigm

Databank gedeelte

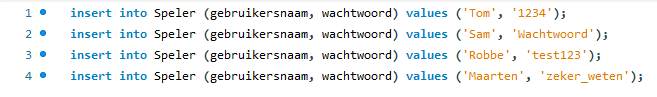
*Omschrijving*

Om deze test applicatie te runnen zijn een aantal default gebruikers nodig. Deze worden toegevoegd aan de Speler tabel. De attributen wachtwoord en gebruikersnaam worden voorzien van een test waarde die gebruikt kan worden in de applicatie.

*extra*

In een verder iteratie kan er nagedacht worden over het beveiligen van de gegevens die in de databank geplaatst worden. Dit kan door middel van een encryptie algoritme en door het toevoegen van een salt zodat iedere waarde in de databank uniek blijft.

*praktijk*



*Figure 4: toevoegen van 4 test gebruikers*

Bron: Mysql queries om gebruikers toe te voegen aan de tabel Speler



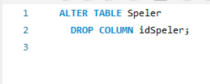
*Figure 5: overzicht van gegevens in databank*

Bron: Mysql overzicht van gebruikers die zijn toegevoegd

*Omschrijving*

De kolom “idSpeler” is verwijderd uit de databank aangezien deze niet gebruikt wordt.

*Praktijk*



*Figure 6: verwijderen van kolom idSpeler*

Bron: Mysql overzicht van gebruikers die zijn toegevoegd

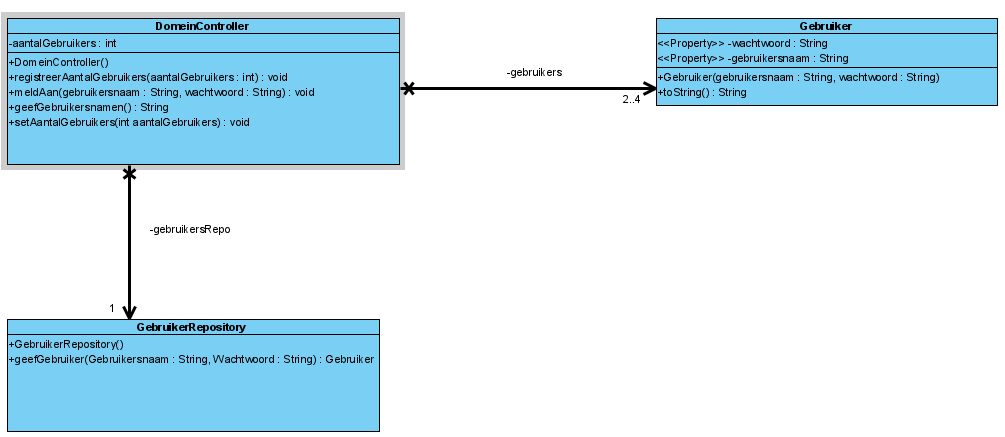
Java gedeelte

DcD

*Omschrijving*

De domeincontroller bevat een attribuut die het aantal gebruikers bewaart. Daarnaast worden slechts 3 methodes gebruikt. De registreer methode stelt de parameter aantal in via een setter. De andere methode meldAan vergelijkt de input van de gebruiker met de gegevens die in de tabel te vinden zijn. Ten slotte hebben we geefGebruikersnamen, deze methode zorgt ervoor dat de ingelogde gebruikers te zien zijn in een overzicht.

*Praktijk*



*Figure 7: voorlopig dcd use case 1*

Bron: Eclipse

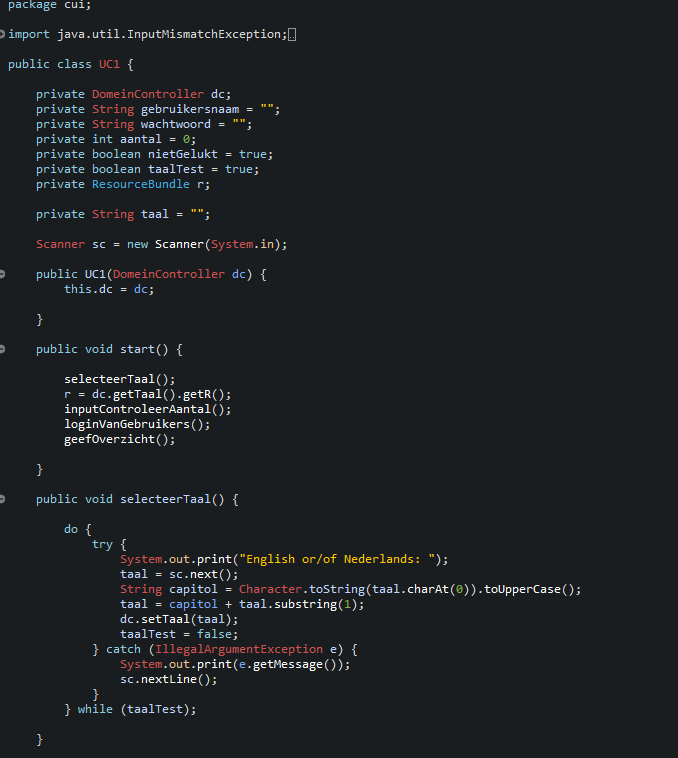
JAVA-code

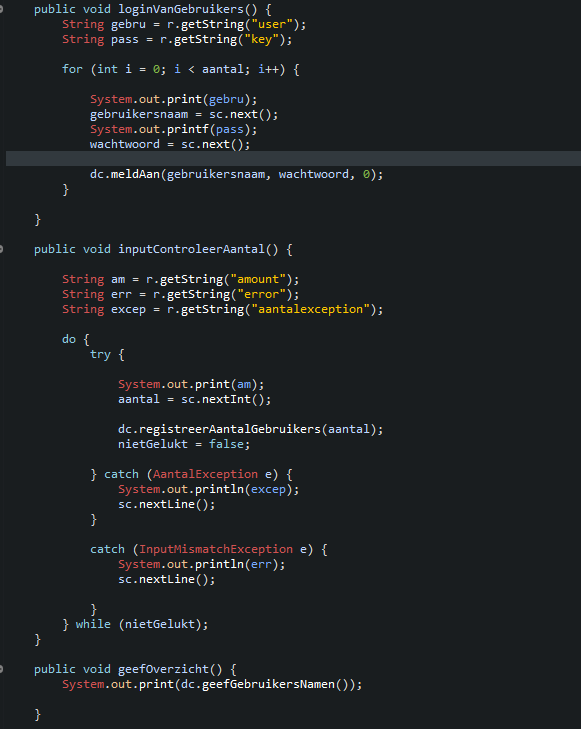
*Iteratie 1*

Omschrijving

De use case wordt getest met een stukje test code waar er gevraagd wordt naar het aantal gebruikers. Dit aantal wordt getest waarbij een exception gegooid wordt bij een incorrecte invoer. Een te groot, te klein en een combinatie van letters zijn allemaal incorrecte invoer bij deze use case. De methode die gebruikt wordt hiervoor is inputControleerAantal().Daarna wordt gevraagd om de logingegevens van elke gebruiker in te voeren dit totdat het correct aantal gebruikers is aangelogd aan de hand van de methode LoginVanGebruiker() In het het laatste deel wordt met de methode geefOverzicht() een overzicht verkregen van de ingelogde gebruikers.

Praktijk





*Figure 8: testapplicatie use case 1*

Bron: Eclipse code die gebruikt wordt om de applicatie te testen

# Use case 2: Speel spel

## Analyse

### Activiteitsdiagram

#### Omschrijving

Dit activiteitsdiagram heeft geen alternatieve verlopen maar wel een loop. Bij deze use case gaat een speler een speler starten en het systeem bepaalt de volgorde en de beurt van de spelers. Wanneer het systeem de volgorde heeft gegenereerd, registreert die de speler aan de beurt en toont de gebruikersnaam van deze speler. Als een speler speelt en er is geen winnaar (volgens de domeinregel “einde”), gaat de volgende speler aan de beurt. De spelers die om de beurt spelen wordt gezien als een loop. Als uiteindelijk er een winnaar is, registreert het systeem de score van elke speler en laat deze zien.

#### Praktijk

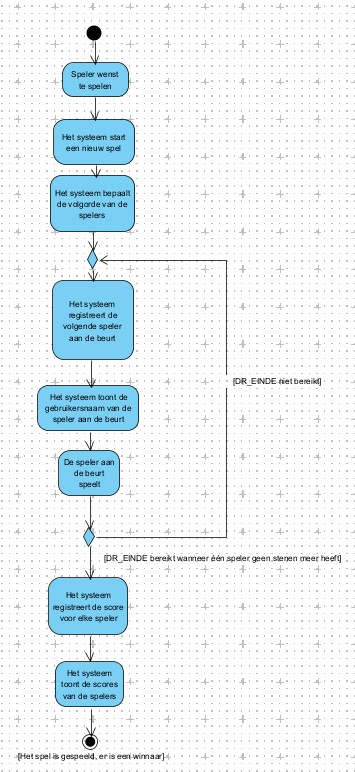


Figure 9: activiteitsdiagram use case 2

Bron: Visual paradigm

### Domeinmodel

#### Omschrijving

Het domeinmodel bestaat uit 3 klassen. Een klasse Speler, Spel en Steen. De klasse Speler heeft 3 parameters namelijk score, die de score van de speler aangeeft na elke speelronde en na het spel eindigd, gebruikersnaam en wachtwoord, die in de databank zijn opgeslagen en uniek is voor elke speler. De klasse Spel heeft een parameter spelerAanDeBeurt, die aanduidt welke speler nu effectief aan de beurt is in de speelronde en een klasse Steen heeft 2 parameters, kleur en getal, die zoals vanzelfsprekend de kleur en getal van de stenen die gebruikt worden in het spel bijhoudt.

#### Praktijk

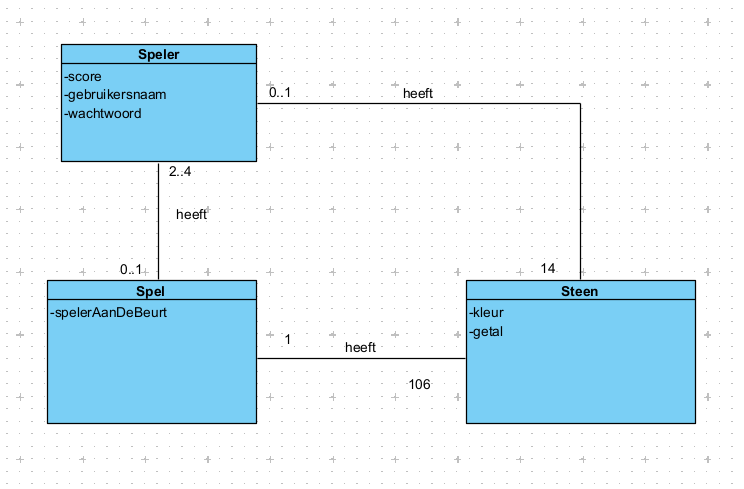


Figure 10: domeinmodel use case 2

Bron: Visual Paradigm

### Ssd-diagram

#### Omschrijving

Allereerst staat een pijl die wijst op het starten van het spel. In deze eerste stap wordt een nieuw spel aangemaakt waarbij de gebruikers die eerder ingelogd waren meegegeven worden. Vervolgens zijn er een aantal operaties die gebeuren in het systeem zelfs zoals het aanmaken van een pot, stenen uitdelen aan elke speler en het verwisselen van de volgorde waarin de spelers aangemeld zijn. De volgende stap staat niet op het diagram en is opnieuw een systeem operatie. De daaropvolgende operatie toont de gebruikersnaam van de gebruiker aan beurt. Na het tonen van de gebruikersnaam wordt de volgende use case opgeroepen en speelt iedere speler op zijn beurt een ronde totdat er een winnaar is dus tot er iemand geen stenen meer over heeft. Deze voorwaarde wordt met de terugkeer pijl scoreGebruikers getoond. Bij dit ssd is ook een operation contract namelijk bij de methode startSpel aangezien hier een nieuwe instantie van een klasse wordt aangemaakt.

#### Praktijk

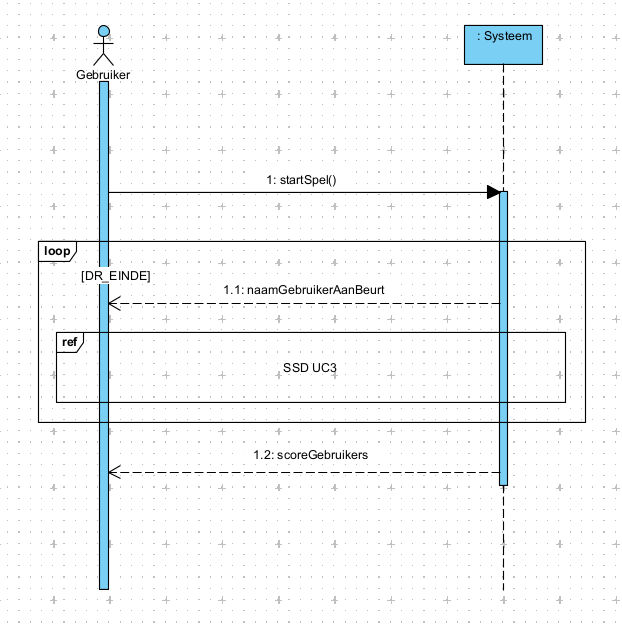


Figure 11: ssd-diagram use case 2

Bron: Visual Paradigm

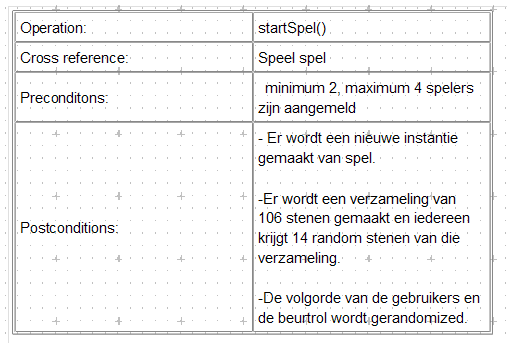


Figure 12: operation contract startSpel()

Bron: Visual paradigm

## Java gedeelte

### DcD

#### Omschrijving

De domeincontroller bevat een nieuwe setter genaamd “setAantalGebruikers()” en drie methodes, “geefGebruikersnamen()”, “meldAan()”, “registreerAantalGebruikers()”. De methode meldAan() zorgt ervoor dat een speler zich kan aanmelden voor er een spel wordt gestart. De setAantalGebruikers() stelt in hoeveel gebruikers er zijn ingegeven via de methode registreerAantalGebruikers(). Daarnaast is er ook een methode geefGebruikersnamen(), die op zijn beurt de gebruikersnamen weergeeft nadat er een winnaar is en het spel gespeeld is.

Het DCD heeft een nieuwe enumeratie bijgekregen voor de kleur van een steen, deze zijn nodig voor met waarden te werken die constant moeten blijven. De klasse “Gebruiker” heeft twee setters gekregen voor het wachtwoord en de gebruikersnaam (setWachtwoord(wachtwoord) en setGebruikersnaam(gebruikersnaam)). De methode maakStenenbordSpeler() en de getter getStenen() zorgen beide voor de stenen te genereren van de speler. Bij de klasse RummikubSpel is er een nieuwe methode geefNaamGebruikerAanBeurt() die de gebruikersnaam van de speler aan beurt uit een lijst haalt en weergeeft. De klasse steen heeft twee constructors gekregen, één voor de joker (Steen()) en één voor de gewone stenen (Steen(getal: Int, kleur: String)). De methode toString() zorgt ervoor dat de steen afgedrukt kan worden als een string.

#### Praktijk

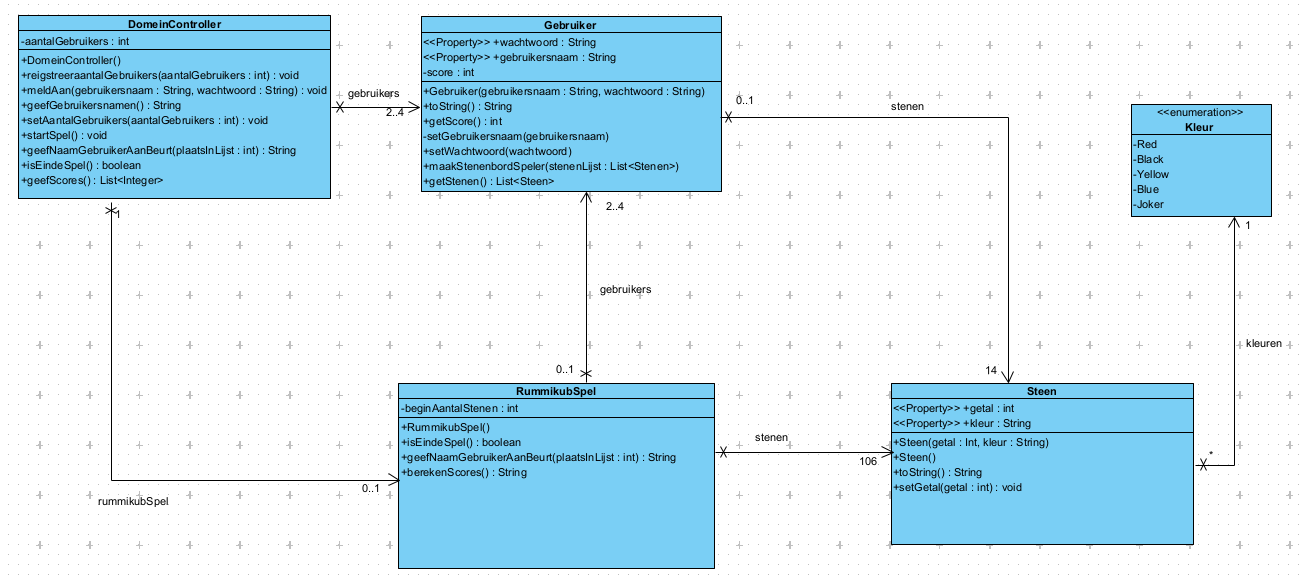


Figure 13: DcD use case 2 op basis van het domeinmodel en ssd-diagram

Bron: Visual Paradigm

### 

### JAVA-code

#### Omschrijving

De applicatie van deze use case is deze keer heel wat eenvoudiger aangezien de meeste code in de klassen zelf wordt uitgevoerd. De eerste methode die aangeroepen wordt na de vorige use case is de methode startSpel. In deze methode wordt de constructor van het spel aangeroepen en wordt alles dus klaargezet zodat iedere speler dan het spel kan spelen in de volgende use case. De registreerVolgendeGebruikerAanDeBeurt methode zorgt ervoor dat er een gebruiker gekozen wordt om een beurt te spelen en zorgt er tevens ook voor dat na de laatste gebruiker het opnieuw aan de eerste gebruikers zijn beurt is. De laatste methode toont het overzicht van de scores in een String.

#### Praktijk



Figure 14: applicatie klasse use case 2  
Bron: Visual paradigm

# Use case 3: Speel beurt

## Analyse

### Activiteitsdiagram

#### Omschrijving

Het activiteitsdiagram bij deze use case gaat over een speler die alle opties voor zich krijgt wanneer het zijn beurt is. Eerst toont het systeem de spelsituatie. De speler wenst daarna actie te ondernemen, het systeem toont al de mogelijke opties. Het grote aantal linkse decisions/verlopen zijn al de opties de speler kan uitvoeren (beurt resetten, steen verplaatsen, joker vervangen,...). Uiteindelijk valideert het systeem de spelsituatie, als dit ongeldig is, geeft het systeem een gepaste melding en toont terug de mogelijkheden. Als de speler een geldige beurt heeft gespeeld stopt zijn beurt. Als dit ongeldig is, toont het systeem een gepaste melding en geeft het systeem de speler een steen.

#### Praktijk

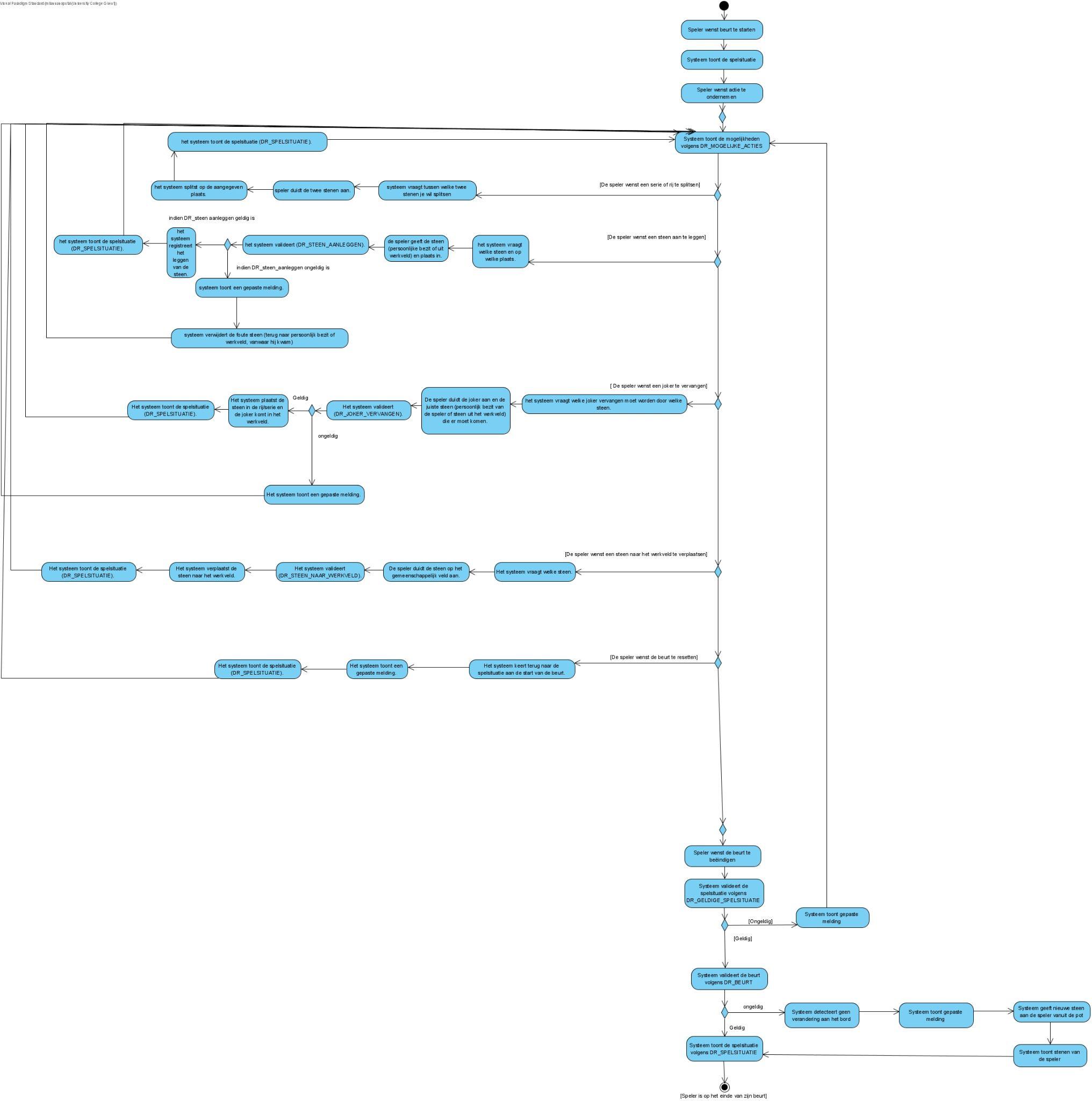


Figure 15: activiteitsdiagram use case 3

Bron: Visual paradigm

### Domeinmodel

#### Omschrijving

Aan het domeinmodel zijn enkel nieuwe klassen toegevoegd. Bij het spelen van een spel wordt gewerkt met een aantal beurten, vandaar de beurt klasse. Deze klasse kan aan het spel en heet zo automatisch toegang tot de gebruiker die aan beurt is. Daarnaast kan de beurt klasse ook aan het spelbord het zogenaamde gemeenschappelijk veld. Bij iedere beurt wordt ook telkens een werkveld aangemaakt om deze op te vullen is er een relatie tussen beurt en steen. De spel klasse is ook deels uitgebreidt. Deze heeft nu een gemeenschappelijk veld een spelbord waarop series of rijtjes worden uitgelegd.

#### Praktijk

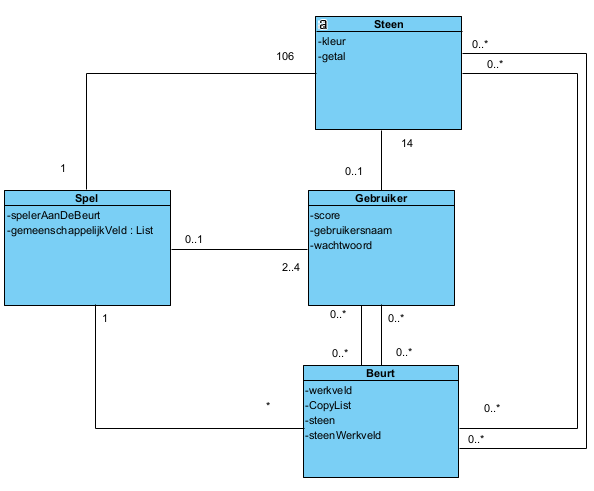


Figure 16: domeinmodel use case 3

Bron: Visual Paradigm

Ssd-diagram

#### Omschrijving

Het eerste SSD in het rijtje is het normaal verloop. Hierbij gaat de speler geen actie ondernemen maar gewoon zijn beurt beëindigen. De volgend vier SSD’s vindt plaats wanneer de speler een bepaalde actie wilt ondernemen. Uiteindelijk stopt de beurt altijd en geeft het systeem de spelsituatie. Voor de SSD’s is er ook een operation contract die beschrijft wat er gebeurd als er een nieuwe beurt wordt aangemaakt.

#### Praktijk

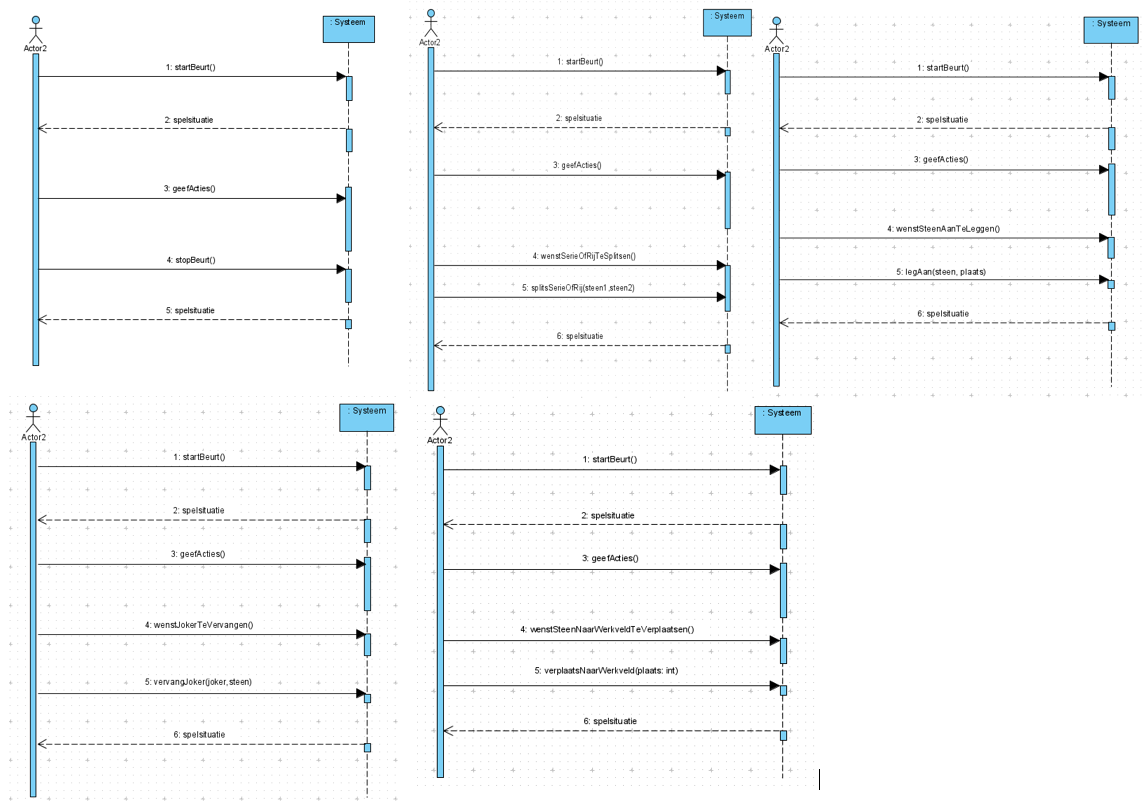


Figure 17: ssd diagrammen use case 3

Bron: Visual Paradigm

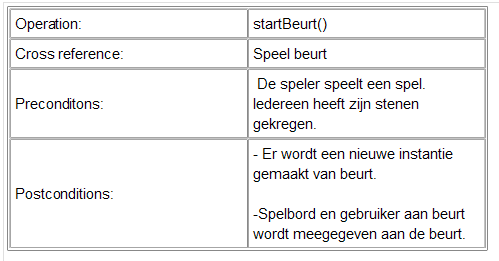


Figure 18: OC voor SSD use case 3

## 

## Java gedeelte

### DcD

#### Omschrijving

Het dcd is aangevuld met enkele nieuwe methodes. De eerste methode startBeurt() wordt aangeroepen in de domeincontroller deze maakt een nieuwe beurt klasse aan voor de gebruikerAanBeurt tegelijk wordt op de GUI de spelsituatie getoond.

De gebruikerAanBeurt zal acties moeten ondernemen. Hiervoor gaat de domeincontroller de methode aanroepen die, aan de hand van de gebruikerAanBeurt en de spel situatie, de geldige of mogelijke acties die valideerd worden door de opgestelde domeinregels in deze use case, tonen. De mogelijke acties zijn zichtbaar in de klasse Beurt. Deze zijn: splitsSerieOfRij(), vervangJoker(), legAan(), verplaatsNaarWerkVeld(), stopBeurt(). Aan de hand van welke actie er word uitgevoerd heeft deze enkele argumenten nodig.

#### Praktijk

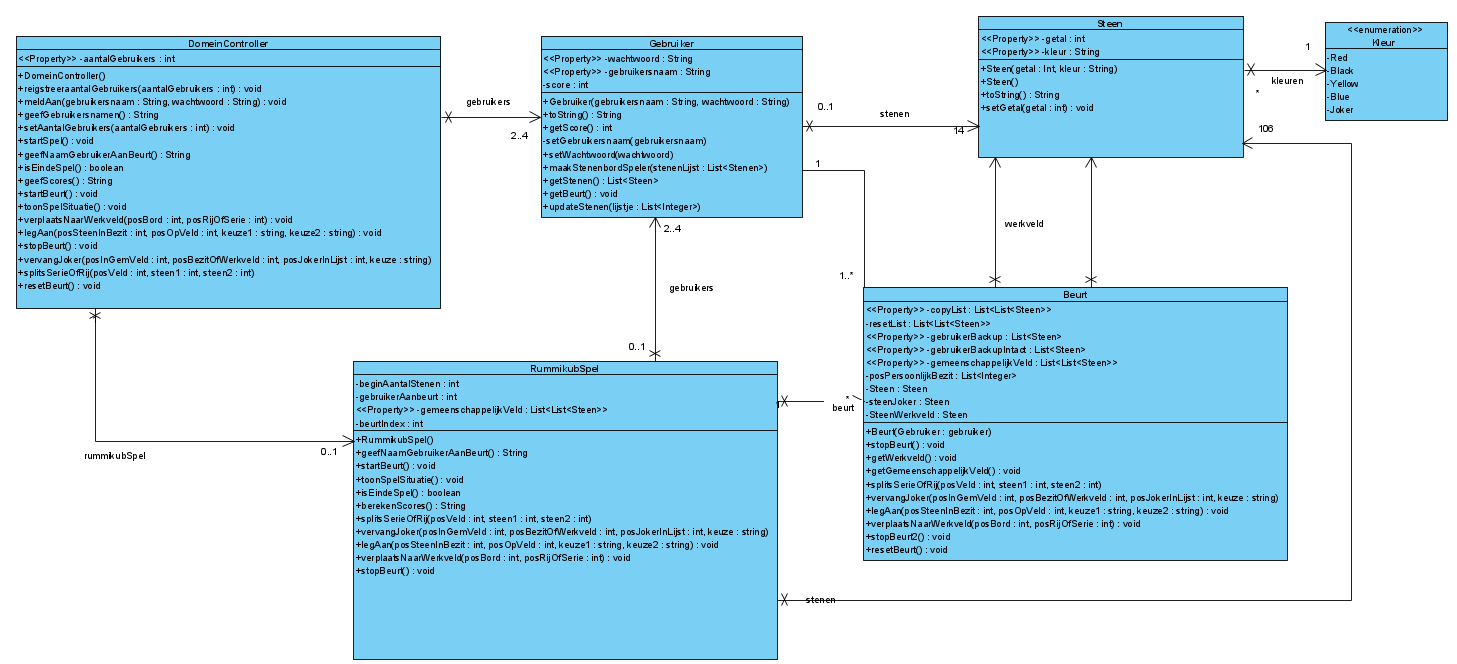


Figure 19: DCD diagram use case 3

Bron: Visual Paradigm

### JAVA-code

#### Omschrijving

De applicatie van deze use case is nog niet helemaal afgewerkt. Voor de methodes die wel al in de applicatie staan, namelijk start, geefSpelSituatie en startBeurt, is het vanzelfsprekend wat hun functie is. In de start methode worden 2 constructors opgeroepen, startBeurt en geefSpelSituatie. Deze gaan na de vorige use case is afgerond, een nieuwe beurt starten voor de gebruikerAanBeurt en gaat dan ook de spel situatie weergeven voor deze gebruikerAanBeurt, zoals aangegeven in deze use case.

De print statement in de geefSpelSituatie is tijdelijk om te testen of de methode het daadwerkelijk doet via de CLI.

Op de volgende afbeelding is de package GUI te zien met al enkele GUI klassen. De klassen verbonden aan use case één en twee werken al correct. In de GUI klasse StartUp is te zien dat bij het uitvoeren van de applicatie een scene word opgesteld (visueel) waarbij de taalselectie word opgeroepen. Ook word een nieuwe instantie gemaakt van de domeincontroller die gebruikt zal worden voor de andere GUI klassen correct te kunnen opstellen.

Maar bij het starten van een beurt en hoe men de velden moet maken voor elke gebruiker aan beurt, zijn stenen zichtbaar te maken en de spelsituatie te kunnen tonen is iets waar we momenteel geen idee over hebben, hoe we dat zouden moeten aanpakken is nog een raadsel voor ons.

Wat ons ook niet verder helpt is het feit dat het aangemaakte rummikub spel in de klasse RummikubSpel niet kan worden doorgegeven aan de klasse Beurt, waar grotendeels van de code in verband met de verschillende acties die de gebruiker kan uitvoeren in zijn beurt en al dan niet zijn beurt kan stoppen, als de voldoeningen overeenkomen met zijn wens deze actie te ondernemen, staat. Het is momenteel zeer moeilijk om dit probleem opgelost te krijgen tegelijkertijd adherend aan de opgedragen feedback.

De laatste figuren geven een overzicht van de verschillende fases van de GUI. Deze schetsen hoe de applicatie grafisch is opgebouwd.

Bron: Eclipse code cui use case 3

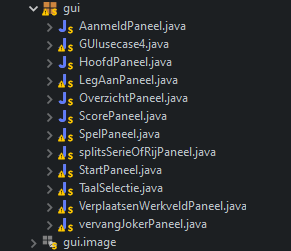


Figure 20: overzicht gui klassen

Bron: Eclipse overzicht gui

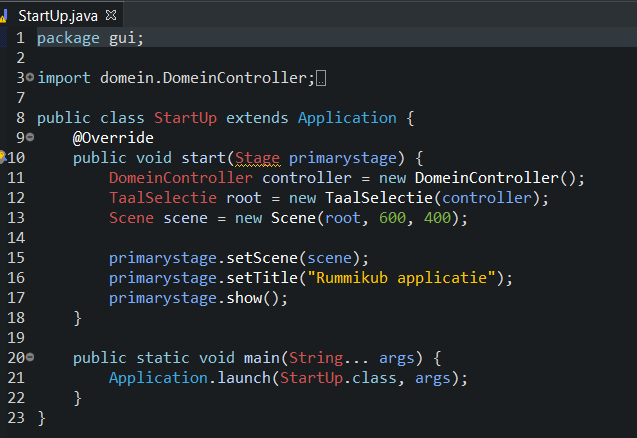
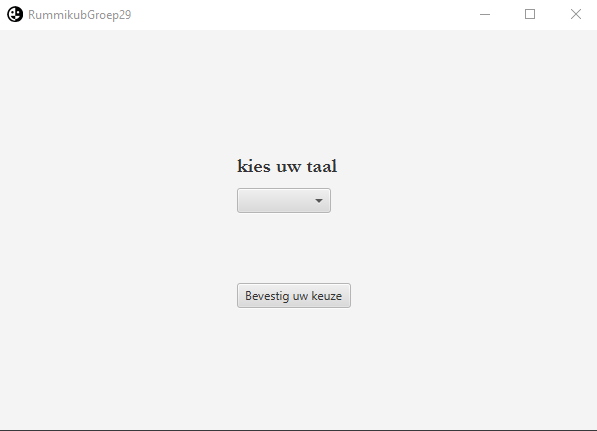


Figure 21: overzicht gui klasse StartUp

Bron: Eclipse code gui use case 3



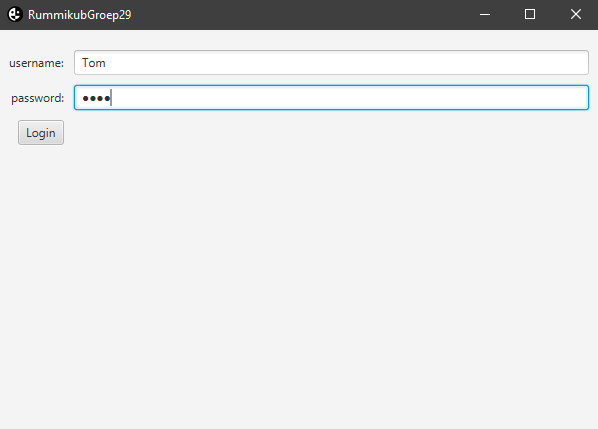
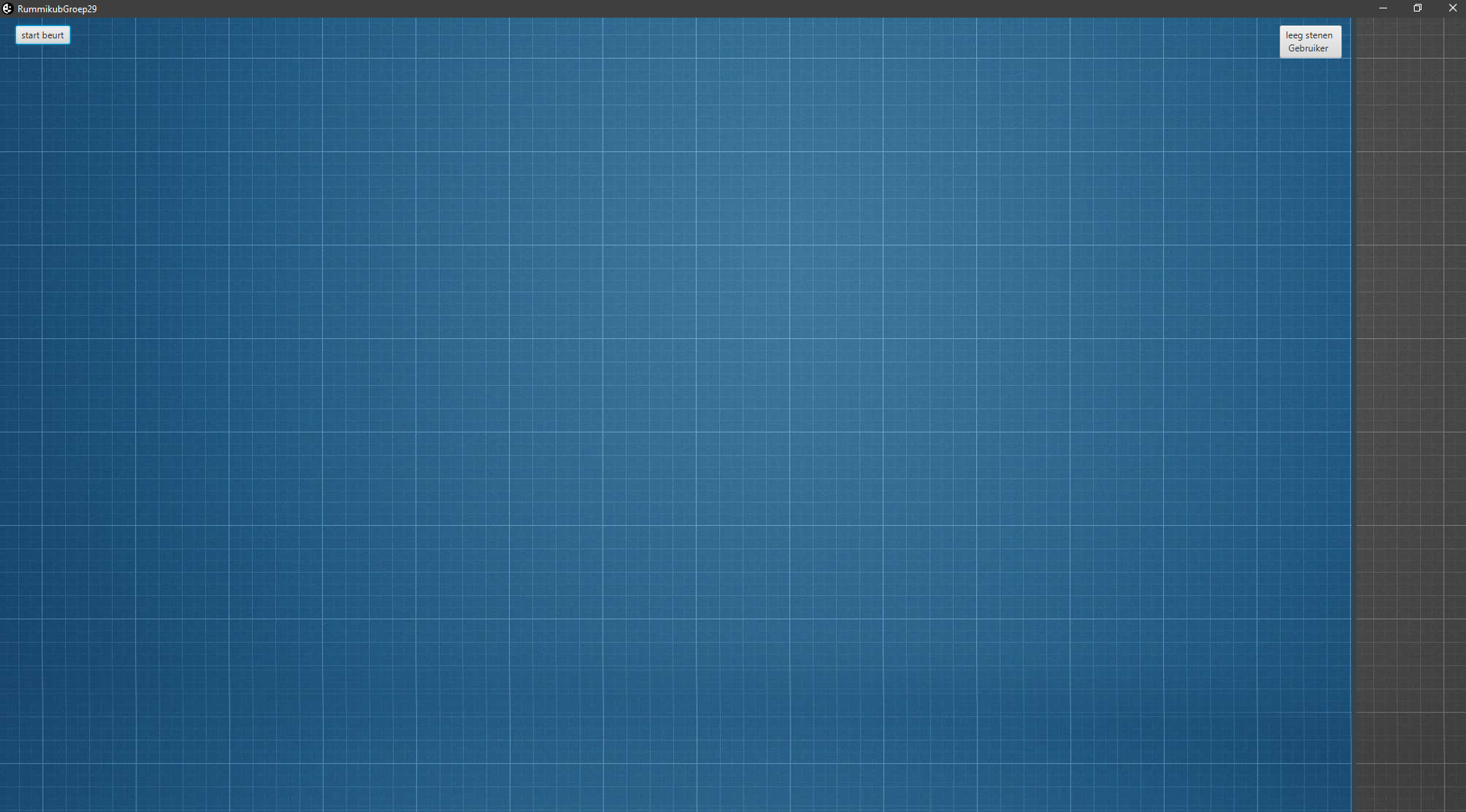
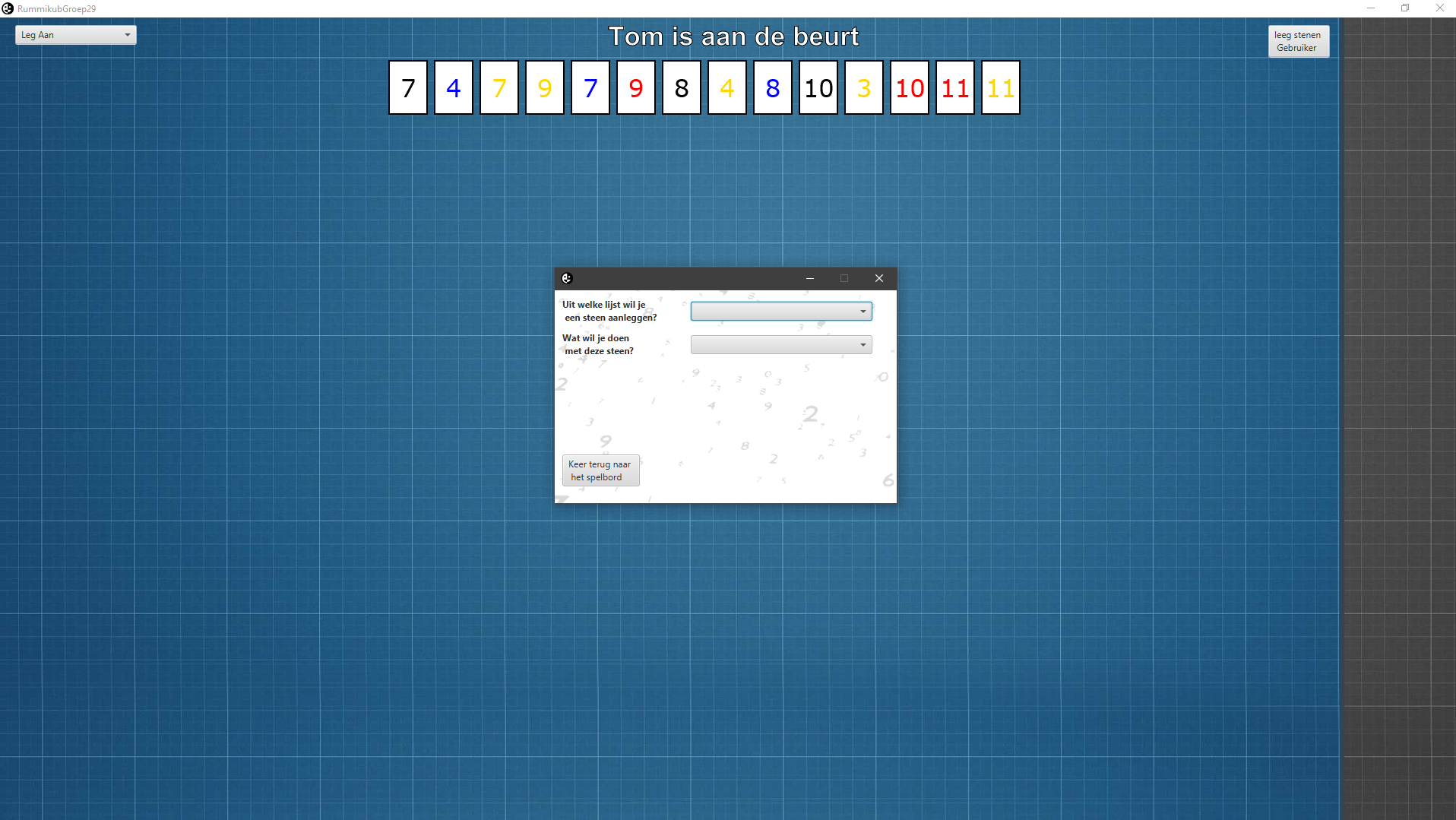
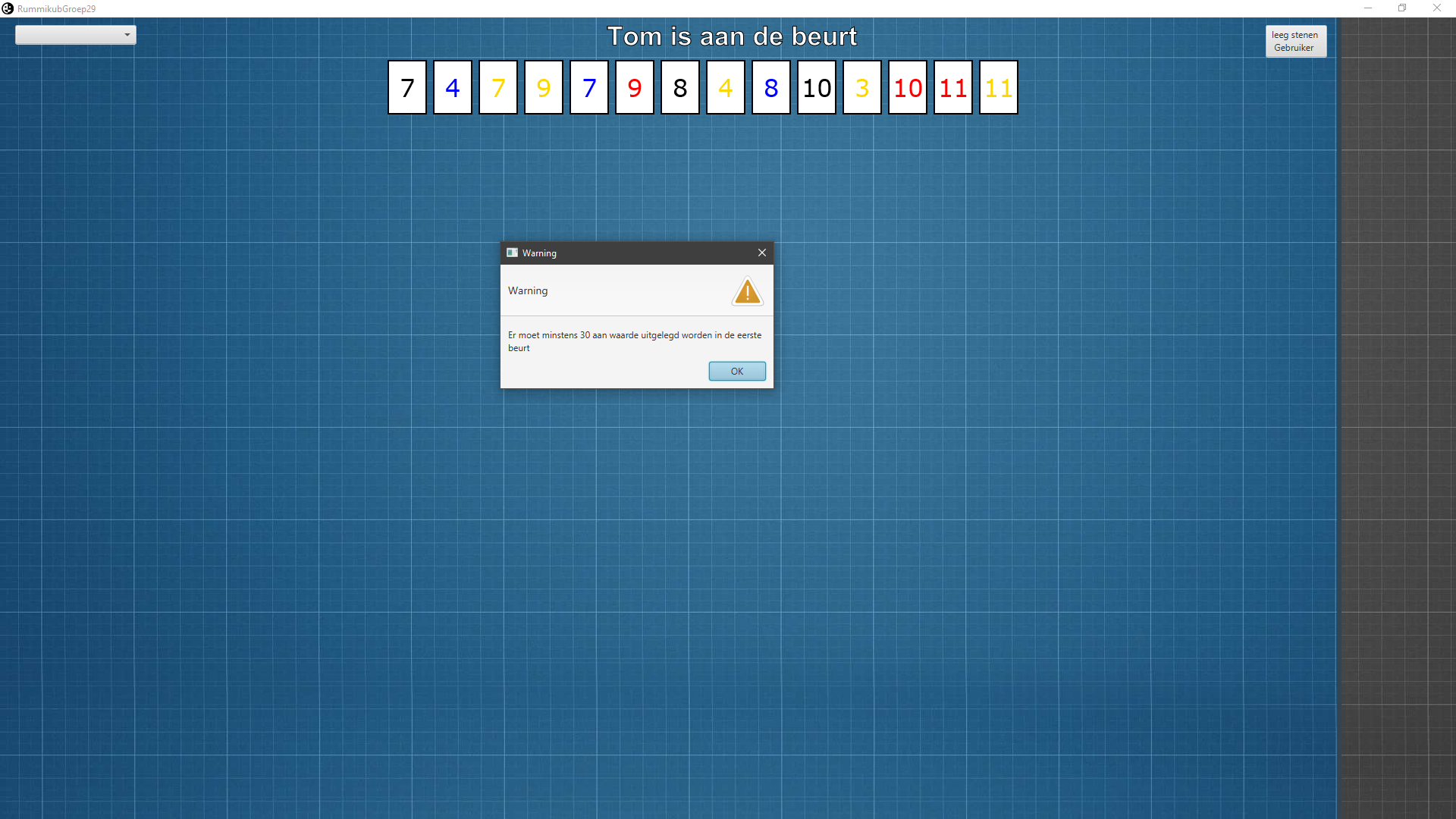


Figure 22: voorbeeld gui use case 3

Bron: Eclipse code gui use case 3







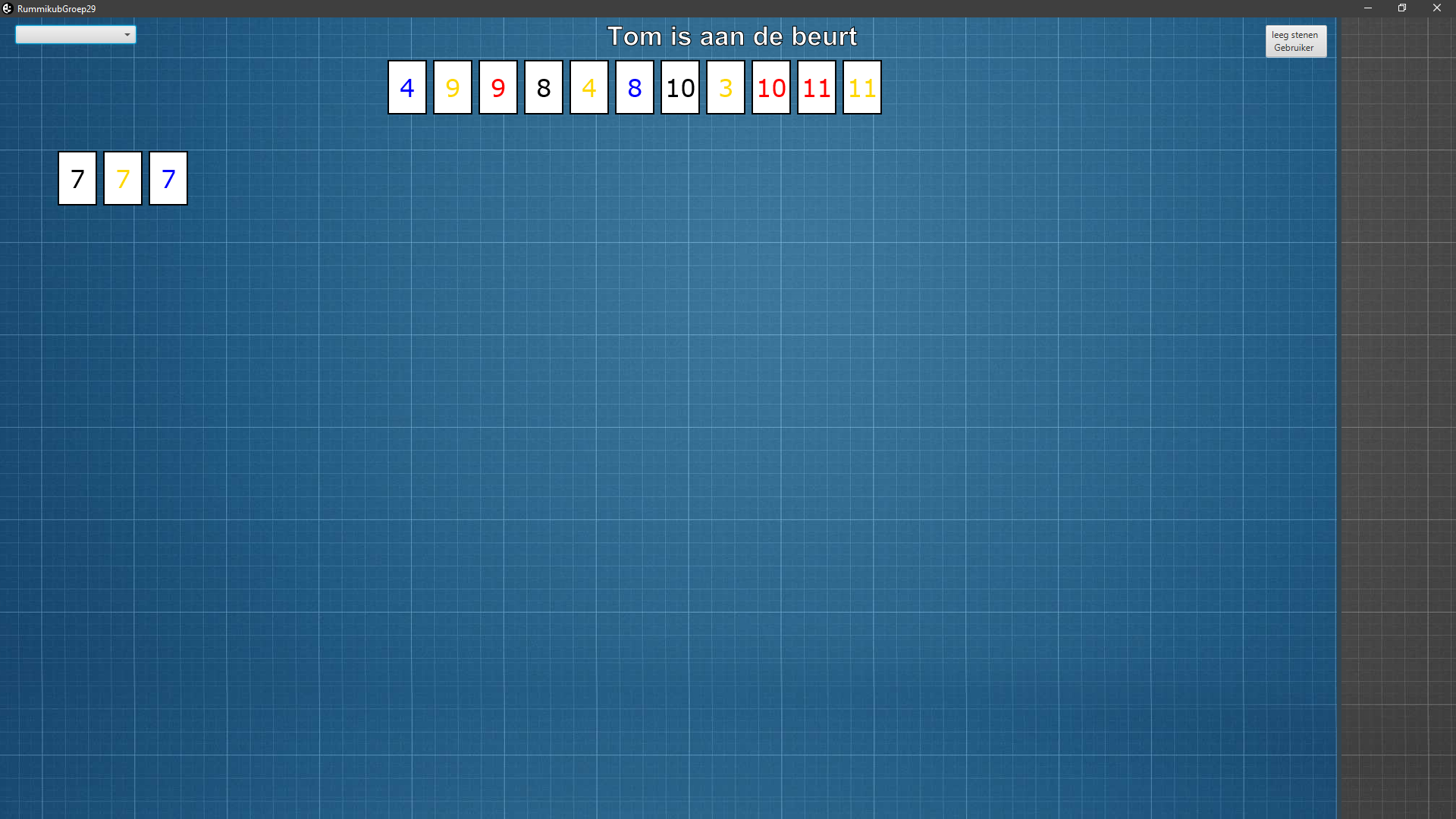


Figure 23: voorbeeld gui spelbord en stenen

Bron: Eclipse code gui use case 3

# Use case 4: Maak overzicht

## Analyse

### Activiteitsdiagram

#### Omschrijving

Een heel eenvoudige en korte use case heeft ook een klein AD, SSD en domeinmodel. De speler wenst een overzicht van de aangemelde spelers van al de gespeelde spelletjes. Het systeem toont dan een overzicht (na het drukken op de knop).

#### Praktijk

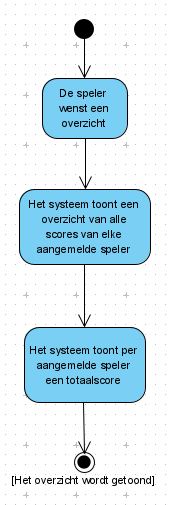


Figure 24: Activiteitsdiagram use case 4

Bron: Visual paradigm

### Domeinmodel

#### Omschrijving

In het domeinmodel zijn twee klassen, scores en speler. De klasse scores houdt de score van alle gespeelde wedstrijden bij en de klasse speler houdt de totaalscore en de gebruikersnaam bij. Een speler kan nul of meerdere spelletjes gespeeld hebben.

#### Praktijk



Figure 25: domeinmodel use case 4

Bron: Visual Paradigm

### Ssd-diagram

#### Omschrijving

In dit SSD zit geen alternatief verloop want er kan eigenlijk niet anders lopen. De speler wenst een overzicht via een knop in de GUI (wenstOverzicht()). Het systeem geeft en tabel met de gebruikersnamen en hun totaalscores (overzichtAangemeldeSpelers, totaalscoreAangemeldeSpeler).

#### Praktijk

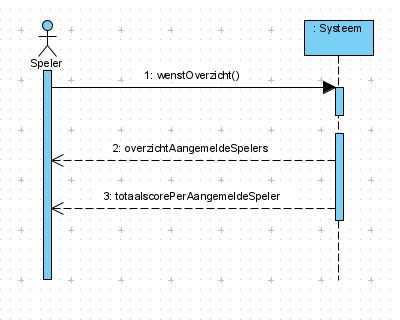


Figure 26: ssd diagrammen use case 4

Bron: Visual Paradigm

## 

## Java gedeelte

### DcD

#### Omschrijving

Het DCD is aangevuld met methodes voor de spelerscores. De methode “getScore()” geeft een lijst weer met de scores van de aangemelde spelers. Dit wordt enkel weergegeven als je op een knop drukt nadat het spel is gestopt. Dan kan je met de knop de scores in de databank resetten. Dit kan gedaan worden door de methode “scoreReset()”.

#### Praktijk

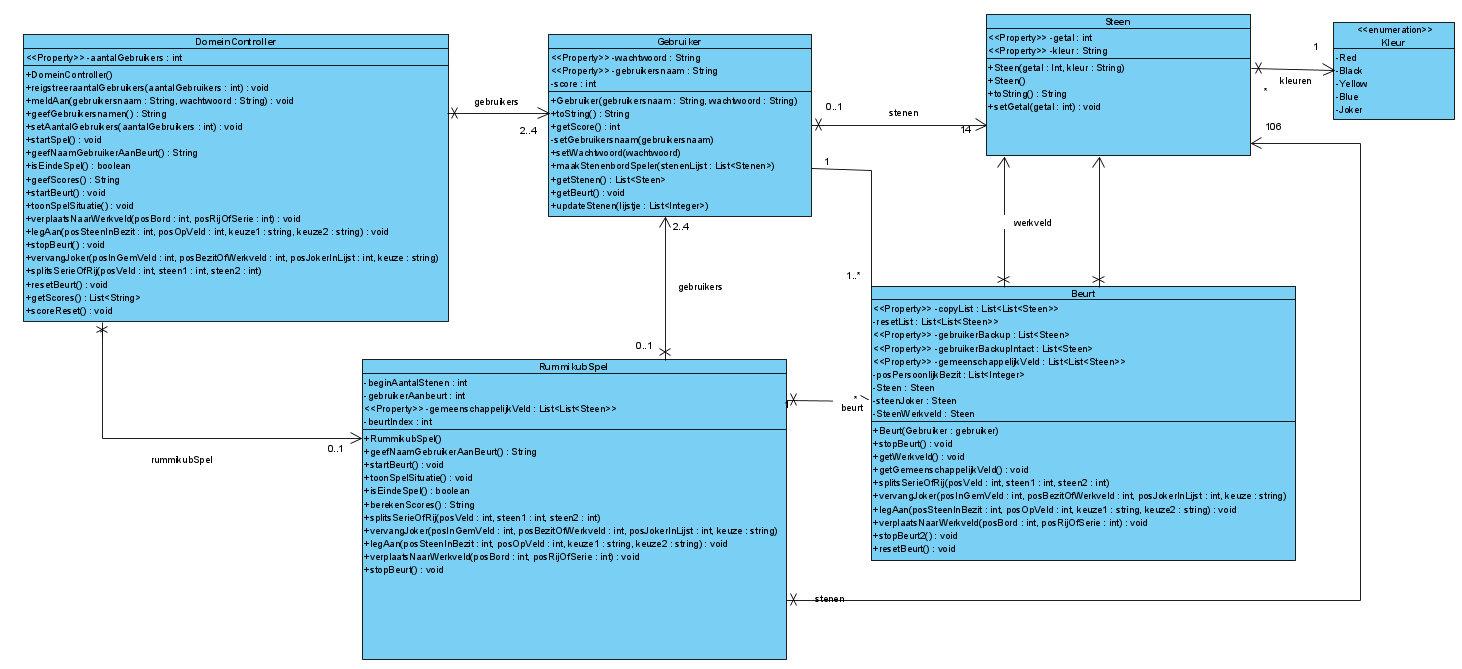


Figure 27: DCD diagram use case 4

Bron: Visual Paradigm

JAVA-code

#### Omschrijving

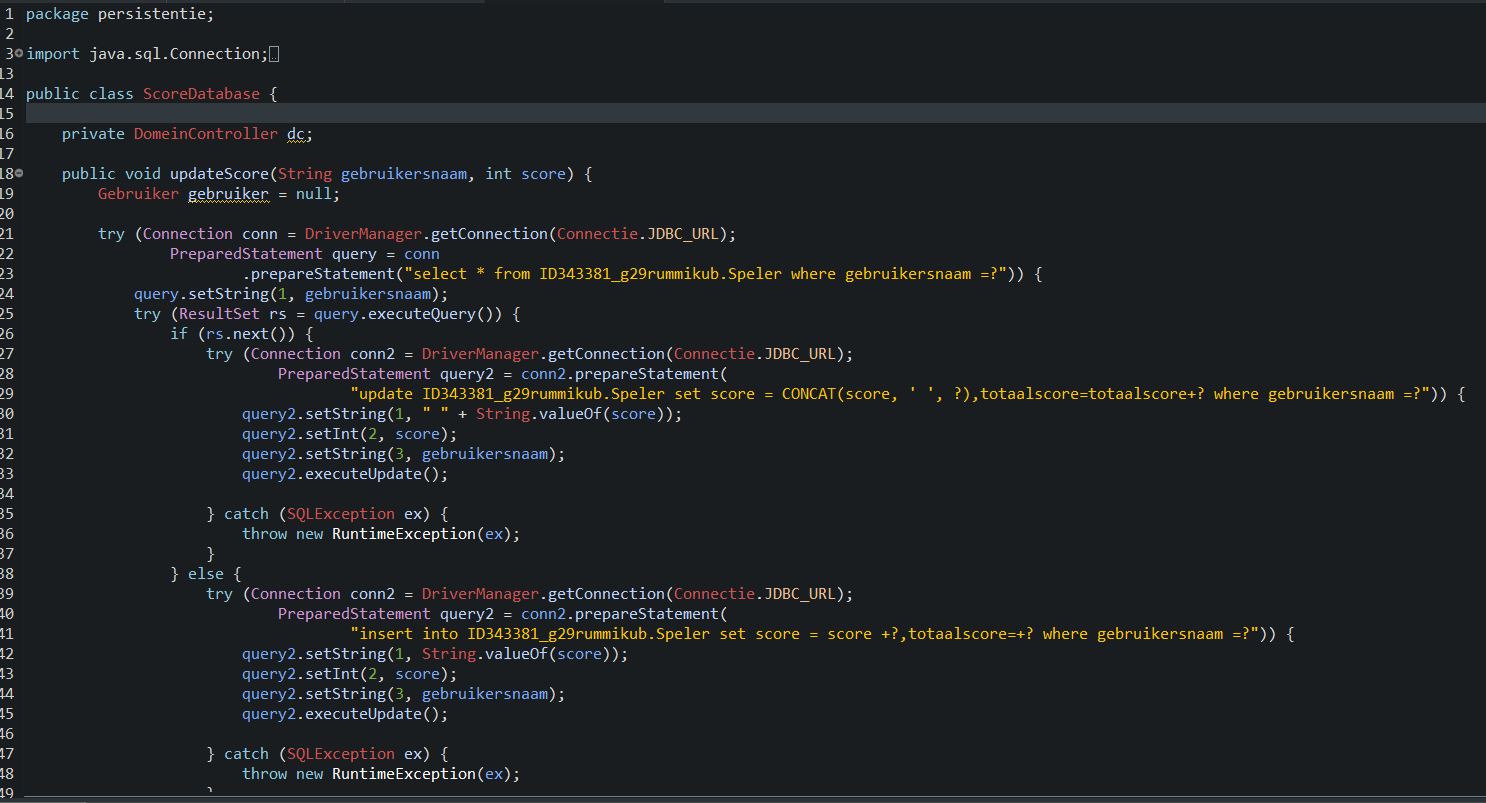


Figure 28: code voor databank

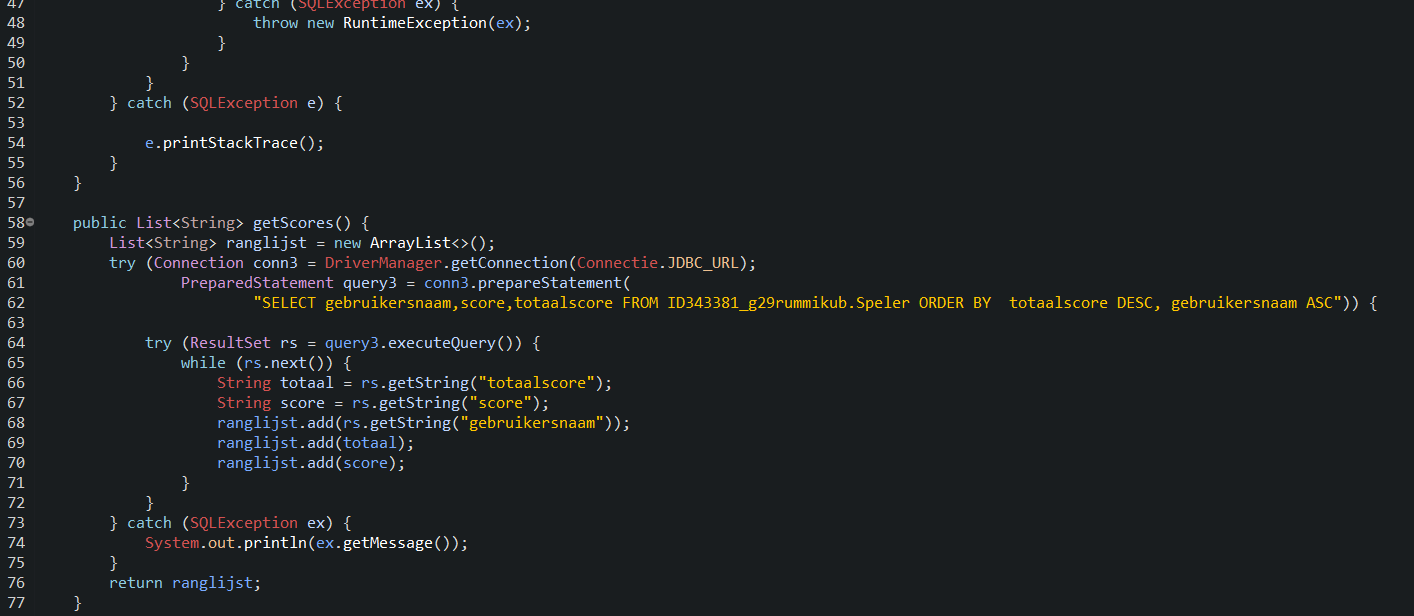


Figure 29: deel 2 code voor databank

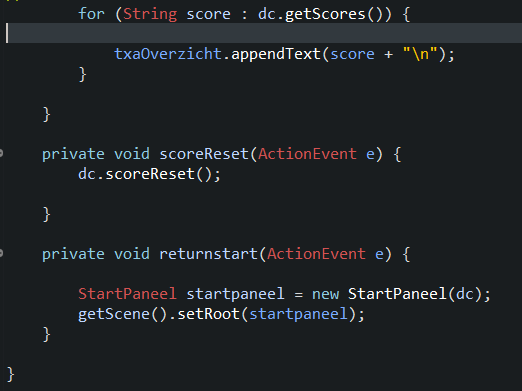


Figure 30: code voor GUI overzicht