Ultimate Subset

Door Sijmen Huizenga



Structured Program Development  
Begeleiding door Mark van der Maas  
Nijmegen, woensdag 25 oktober 2014

Inhoud

[Inleiding 2](#_Toc402464683)

[Set 3](#_Toc402464684)

[Hoe werkt de game? 3](#_Toc402464685)

[Eigenschappen? 3](#_Toc402464686)

[Wat is een set? 3](#_Toc402464687)

[Requirements 4](#_Toc402464688)

[Menu 4](#_Toc402464689)

[Game 4](#_Toc402464690)

[Scorebord 5](#_Toc402464691)

[About 5](#_Toc402464692)

[Software Ontwerp 6](#_Toc402464693)

[Schermen 6](#_Toc402464694)

[Knoppen 6](#_Toc402464695)

[Scorebord 7](#_Toc402464696)

[Kaarten 8](#_Toc402464697)

[Selectie 10](#_Toc402464698)

[Game 10](#_Toc402464699)

[Timing 11](#_Toc402464700)

[Bestandtype 12](#_Toc402464701)

[Test Cases 13](#_Toc402464702)

[isSet 13](#_Toc402464703)

[GetPossibleSets 13](#_Toc402464704)

[getCardStack 13](#_Toc402464705)

[shuffleStack 14](#_Toc402464706)

[Bijlage 15](#_Toc402464707)

[Schermontwerp menu 15](#_Toc402464708)

[Schermontwerp game 16](#_Toc402464709)

[Schermontwerp scoreboard 17](#_Toc402464710)

[Schermontwerp about 18](#_Toc402464711)

# Inleiding

SET is een leuke en bekende game. Ik heb deze game geanalyseerd en nagebouwd in Processing. De ontwikkeling van dit programma is gestructureerd uitgevoerd naar aanleiding van de course Structured Programming Development. In dit verslag wordt achtereenvolgens aandacht gegeven aan de volgende punten.

* Spelregels: Hoe werkt het spel?
* Requirements: Wat moet het programma kunnen?
* Software ontwerp: Hoe ga ik het programma maken?
* Test Cases: Hoe ga ik het programma testen?

Tot slot zal ik in de reflectie aandacht geven aan de voor en nadelen van Processing ten opzichte van Java.

Als aanvulling op dit verslag is er een lijst met alle gebruikte functies. Deze geeft aan wat de argumenten en return waarden van de functies zijn. Dit bestand heet “functies tabel.xlsx”.

# Set

## Hoe werkt de game?

In beeld heb je een aantal kaarten. Dit aantal kaarten varieert tussen 9 en 12 afhankelijk of je het originele of een versimpelde versie speelt. De bedoeling is om drie kaarten te selecteren die een ‘set’ vormen. Als deze kaarten een set vormen, worden deze kaarten van het bord verwijderd en komen er nieuwe kaarten uit de stapel.

De stapel bevat in het begin van het spel alle mogelijke kaarten(27 of 81). Vervolgens pakt de speler een set totdat er geen sets meer op tafel liggen of totdat de stapel leeg is. Dit gaat door totdat alle kaarten op zijn of totdat er geen set meer mogelijk is.

## Eigenschappen?

Elke kaart heeft een aantal eigenschappen. In de verkleinde versie zijn er 3 eigenschappen:

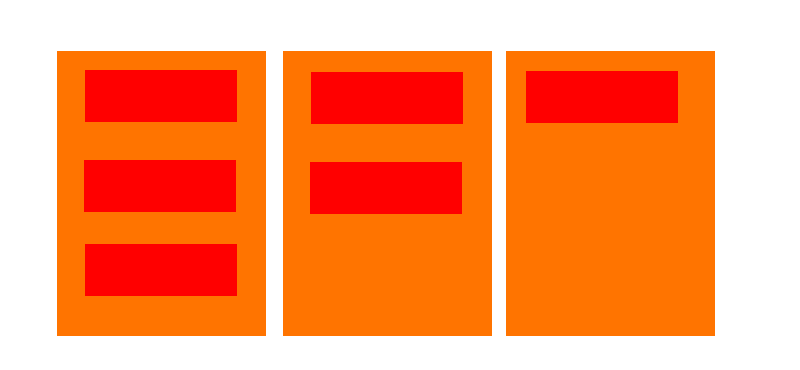
* De hoeveelheid (1, 2, 3)
* De vorm (rechthoek, driehoek, ovaal)
* De kleur van de vorm (rood, geel, blauw)

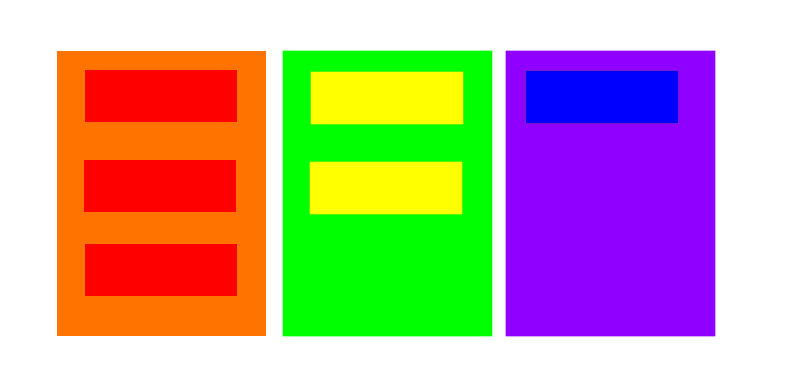
In de originele versie is er nog een extra eigenschap:

* De kleur van de kaart (oranje, groen, paars)

Alle mogelijke kaarten zijn online te vinden: [27 kaarten](https://github.com/SijmenHuizenga/SubSet/blob/master/afbeeldingen/27Kaarten.png)[[1]](#footnote-1) en [81 kaarten[[2]](#footnote-2).](https://github.com/SijmenHuizenga/SubSet/blob/master/afbeeldingen/81Kaarten.png)

## Wat is een set?

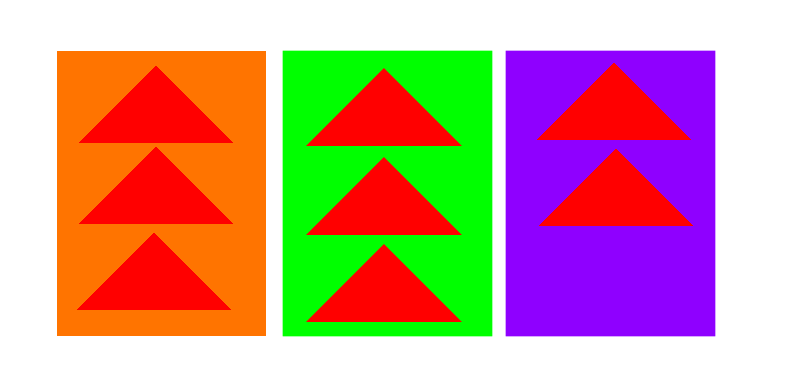
Een set wordt in dit spel pas een valide set genoemd als voor elke eigenschap geld dat die eigenschap voor elke kaart het zelfde of verschillende is. Hier zijn wat voorbeelden:

**voorbeeld 1**  
kleur: rood, rood, rood (overal zelfde)   
achtergrond kleur: oranje, oranje, oranje (overal zelfde)   
vorm: rechthoek, rechthoek, rechthoek (overal zelfde)   
hoeveelheid: 3, 2, 1 (overal anders)  
Dus dit is een valide set.

Voorbeeld 1

**voorbeeld 2**  
Kleur: rood, geel, blauw (overal anders)   
Achtergrond kleur: oranje, groen, paars (overal anders)   
Vorm: rechthoek, rechthoek, rechthoek (overal gelijk)   
Hoeveelheid: 3, 2, 1 (overal anders)   
Dus dit is een valide set.

Voorbeeld 2

**voorbeeld 3**  
kleur: rood, rood, rood (overal anders)  
achtergrond kleur: oranje, groen, paars (overal anders)  
vorm: driehoek, driehoek, driehoek (overal gelijk)  
hoeveelheid: 3, 3, 2 (*deels gelijk, deels anders*)  
Dus dit *niet* een valide set.

voorbeeld 3

# Requirements

Zoals te zien is in de schermontwerpen(zie Bijlage), komen er vier schermen. Menu, ScoreBbrd, About en Game. In deze vier hoofdstukken leg ik uit wat er binnen deze schermpjes gebeurt.

## Menu

Het menu is het beginscherm van het programma. Vanuit hier kan de speler een aantal verschillende acties ondernemen. Ten eerste kan de speler via de twee witte knoppen het spel starten. Door op een van deze knoppen te drukken word het ‘Game’ scherm geopend. Bij ‘Start Easy Mode’ wordt de game op makkelijke modus gestart, en bij ‘Sart Original Mode’ wordt de game op originele moeilijkheidsgraad gestart. Nog voordat de game gestart is, wordt een invoervakje weergegeven om de naam van de speler te noteren. Deze naam wordt aan het einde samen met de score in het score bord opgeslagen.

Door op de knop ‘Score Board’ te klikken wordt het scorebord geopendDoor op de knop ‘About and Rules’ te klikken wordt het scherm ‘About’ weergegeven.

Als laatste is er nog de knop ‘Load saved Game’. Door op deze knop te drukken wordt deze opgeslagen game gestart. Je komt dan uit in het Game scherm. Als er geen opgeslagen game is wordt een error weergegeven.

## Game

Het game scherm heeft een aantal onderdelen. Ten eerste zijn er statistieken. Deze zijn te vinden linksboven in het scherm. Deze statistieken geven bijvoorbeeld de huidige tijd, aantal kaarten op de stapel en de highscore weer.

Daarnaast heb je rechts het speelvlak. Hier liggen 9 of 12 kaarten, afhankelijk of je simpel of orgineel speelt. Door met je muis te klikken en te slepen, kan je de kaarten van locatie veranderen. Op deze manier kan je de kaarten op een volgorde leggen waarbij jij makkelijk een valide set kan vinden.

Als je denkt dat je een set hebt gevonden, dan kan je de kaarten een voor een aanklikken. Dan springen ze naar de vakken linksonder in het scherm. Als je een verkeerde kaart heb aangeklikt, kan je linksonder op de kaart klikken, en dan springt hij weer terug naar het scherm. Als er drie kaarten liggen, dan verschijnt er onder de drie hokjes een knop. Deze knop kan verschillende tekst hebben:  
 - Zijn de kaarten een valide set? Dan zegt de knop ‘Set! Hand in’.  
 - Zijn de kaarten geen set? Dan zegt de knop ‘No set.’  
In het eerste geval kan je de kaarten inleveren, en worden er drie nieuwe kaarten in het speelveld gelegd. Bij het tweede geval doet de knop niks.

Verder zijn er nog vier knoppen die een actie uitvoeren. Van boven naar beneden:  
‘**Save & Quit**’: Sla het spelletje op en ga terug naar het menu. Er kan altijd maar één spel zijn opgeslagen. Als er al een spel is opgeslagen wordt deze verwijderd.  
‘**Order Cards**’: Alle kaarten in het speelveld worden op volgorde gelegd zodat het speelveld weer overzichtelijk wordt.  
‘**Hint**’: Als de vakjes linksonder leeg zijn, dan worden er twee kaarten vanuit het speelveld gepakt en in de vakjes linksonder gelegd. Bij gebruik van een hint wordt de score met één strafminuut verhoogd.   
‘**Give Up**’: De speler geeft op. De game wordt afgesloten, de score wordt niet genoteerd omdat het spel niet is afgerond.

Op een gegeven moment is de stapel op en zijn alle set’s gevonden: Het spel is afgelopen. Er wordt een popup schermpje weergegeven met de mededeling dat het spel is afgelopen. De score wordt samen met de naam van de speler genoteerd op het scorebord. De score bestaat uit de tijd die de speler over het spel heeft gedaan in combinatie met de strafminuten. Op dit moment komt er in beeld de tekst ‘Game Over’ te staan. Met een klik ergens op het scherm wordt het menu weergegeven.

## Scorebord

Op het scorebord wordt de top vijf beste scores weergegeven. Er is een aparte ranking voor ‘Simple’ en ‘Original’ mode. Hoe dit er uit ziet is goed te zien in de schermontwerpen(Schermontwerp scoreboard). Verder zijn er nog twee knoppen. De eerste genaamd ‘Clear Scores’ verwijdert alle score lijsten. De tweede knop ‘Back to Menu’ verwijst terug naar het menu scherm.

## About

Op deze pagina is te vinden hoe het spel werkt, hoe je het spel moet spelen, welke dingen van belang zijn enzovoort. De enige knop die hier aanwezig is de knop ‘Back to Menu’, deze knop verwijst naar het menu.

# Software Ontwerp

Er is vooraf een ontwerp gemaakt op hoofdlijnen. Details zijn tijdens het programmeren ingevuld. In dit verslag zijn vooral de algemene flow van functies beschreven. Bij belangrijke functies zijn meer uitgelegd

## Schermen

Er zijn meerdere schermen. Het actuele scherm wordt bijgehouden in de variabele selectedScreen. Dit getal refereert naar een scherm. Welk scherm bij welk nummer hoort is te vinden in de variabele die beginnen met ‘SCREEN\_’.

Om tussen verschillende schermen te wisselen, zijn er een aantal functies die een ander scherm tonen. Deze zijn *backToMenu*, *showAboutScreen* en *showScoreScreen*. Deze functies spreken voor zich.

Al deze verschillende schermen moeten op enig moment getekend worden. Voor het tekenen van schermen zijn ook een aantal functies: *drawAbout*, *drawMenu*, *drawGame*, *drawScoresScreen*. De juiste teken-functie aanroepen aan de hand van *selectedScreen* wordt gedaan door de functie *drawScreen*.

Als laatste is er nog de variabele *popupTxt*. Deze String houdt bij of er een overlay over het scherm ligt. Als dat het geval is, wordt er een half doorzichtige blauwe overlay weergegeven met in het midden groot de tekst die in *popupTxt* staat. Het tekenen van deze overlay en tekst wordt gedaan door de functie *drawPopupScreen*.

## Knoppen

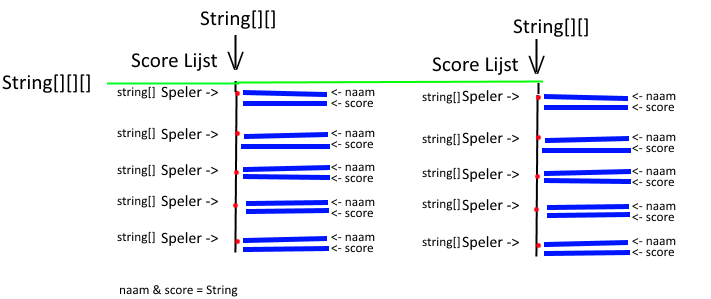
Door het hele programma worden knoppen gebruikt. Alle informatie over knoppen is opgeslagen in twee arrays: *buttonData[][]* en *buttonTxt[]*. Als je het identificatie nummer van een knop weet, kan je met de functie *buttonIdToLocation* het identificatie nummer omzetten naar een locatie nummer. Met dit locatie nummer kan binnen de twee arrays de juiste data gevonden worden.

Navigeren binnen het buttonData array kan door middel van acht numerieke variabelen die allemaal beginnen met ‘BUTTON\_’. Een nieuwe knop toevoegen kan met de functie *addBUTTON*.

Om elke knop een actie te geven, is er de functie *doButtonAction*. Deze functie roept door middel van het button identificatie nummer de juiste actie aan. Het zoeken naar een bepaalde knop met een x en y locatie, kan door middel van de functie *getButtonAtLocation*.

Het tekenen van de juiste knoppen wordt gedaan in de functie *drawButtons*.

## Scorebord

Het scorebord wordt opgeslagen in een drie dimensionaal array met tekst. In figuur 1 is te zien hoe dit array is ingedeeld.  


Figuur 1

Om met het score bord te werken, zijn er een aantal functies. Ten eerste heb je de *initScoreBoard* functie. Deze moet aan het begin van het programma worden aangeroepen. Deze functie laad het score bord.

Het laden van scores wordt gedaan door de functie *loadScoreBoard*. De tegenhanger is *saveScoreBoard*. Deze functie slaat de scores op in een bestand op de computer. Het opslaan wordt gedaan volgens het bestandsformaat dat in hoofdstuk Bestandsformaat is beschreven.

Om de scores in de goede volgorde weer te geven, moeten de scores geordend worden. Dit gebeurt in de functie *orderScoreBoard*.

Er mogen altijd maar maximaal vijf scores zijn opgeslagen, daarom is er een functie die het scores array inkrimpt tot vijf. Deze functie heet *maximizeScoreBoard*. De actie achter de knop “Clear Scoers” zit de functie *clearScores*. Deze functie verwijdert alle score entry’s.

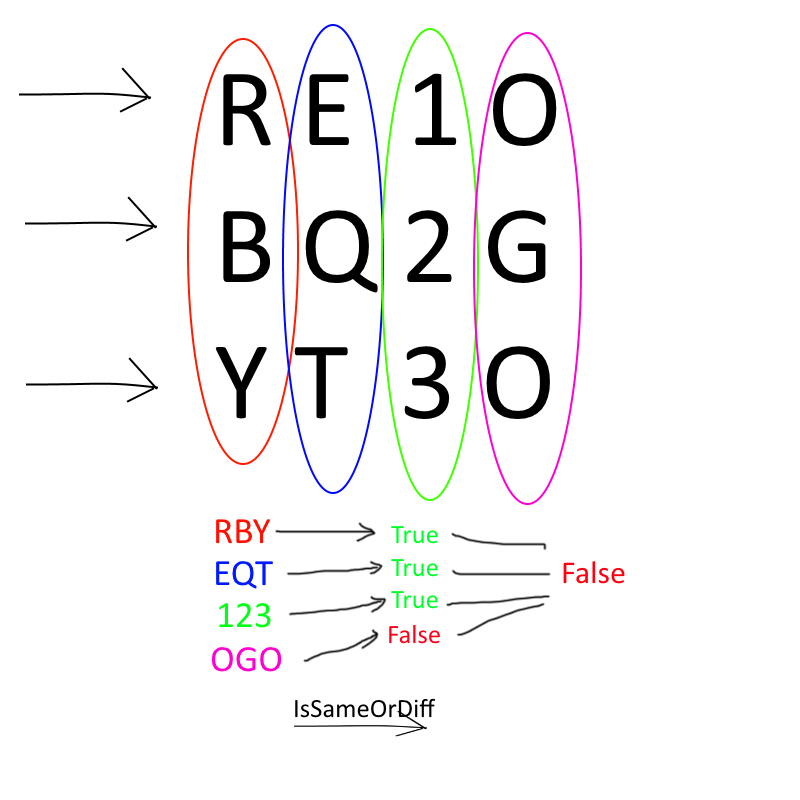
Om nieuwe scores toe te voegen is er de functie *addScoreEntry*. Deze doet niks anders dan een row toevoegen in het score bord array. Als laatste is er de *drawScoreList* functie. Hierin wordt een score lijst getekend.

## Kaarten

|  |  |  |
| --- | --- | --- |
| R | Red | De kleur van het figuur is rood. |
| B | Blue | De kleur van het figuur is blauw. |
| Y | Yellow | De kleur van het figuur is geel. |
| E | Ellipse | De vorm is een ovaal. |
| Q | Quad | De vorm is een vierkant. |
| T | Triangle | De vorm is een driehoek. |
| 1 | One | Er staat één vorm op het kaartje. |
| 2 | Two | Er staan twee vormen op het kaartje. |
| 3 | Three | Er staan drie vormen op het kaartje. |
| O | Orange | De kleur van de achtergrond is oranje. |
| G | Green | De kleur van de achtergrond is groen. |
| P | Purple | De kleur van de achtergrond is paars. |
| N | None | Er is geen achtergrondkleur. (simple version) |

In dit programma is er voor gekozen om een kaart opgeslagen als een String. Deze String bevat altijd vier karakters. De vier groepen karakters staan hiernaast. Een voorbeeld van een kaart zou dus “RE1O” kunnen zijn. Dit zou dan een kaart met 1 rood rondje en met een Oranje achtergrondkleur. Deze methode is erg overzichtelijk.

Als eerste moet er een stapel kaarten gemaakt worden. Dit wordt gedaan door de functie *getCardStack*. Deze methode maakt gebruik van de functie *makeCard*. In *makeCard* wordt een kaart gemaakt met de gegeven eigenschappen.

Voor gebruik moet deze stapel kaarten geschud worden. Het schudden van een stapel kaarten wordt gedaan door de methode *shuffleStack*. Dit schudden ga ik implementeren volgens het ‘fisher-yates’ algoritme.

Nu zijn we aangekomen bij de functie waar het hele programma om draait: *isSet*. Deze functie kijkt of een aantal kaarten samen een set vormen. Nu is de vraag, hoe moet dit gedaan worden? De afbeelding hiernaast laat het heel duidelijk zien.

De regels horizontaal stellen een kaart voor. Er zijn dus drie kaarten. Daarna worden verticaal de eigenschappen los gehaald. Deze eigenschappen worden door de gekleurde rondjes weergegeven. Volgens de regel: “Elke eigenschap moet of helemaal gelijk of helemaal ongelijk zijn.” worden de eigenschappen gecheckt. De eerste groep eigenschappen “RBY” bevat allemaal verschillende karakters. Deze eigenschap is dus goed gekeurd. Deze keuring gebeurt voor elke eigenschap. Als één of meer van de eigenschapen de keuring niet doorkomen, dan is het geen valide set.

De keuring of een String uit helemaal dezelfde of helemaal verschillende karakters bestaat wordt gedaan in de functie *isSameOrDiff*. Deze functie gebruikt twee regular expressies. De eerste kijkt of de string bestaat uit alleen maar dezelfde karakters, en de andere kijkt of de string alleen maar uit verschillende karakters bestaat. Als een van de twee ‘waar’ is, is het resultaat ‘waar’.

Nu zijn er kaarten, maar er is nog niks op het scherm. Om deze overstap te maken komt er een functie genaamd *addCardToScreen*. Deze functie pakt de laatste kaart van de stapel kaarten (*stack*), en voegt deze toe aan het scherm als een knop. De kaart wordt is geen knop, maar programmeer technisch wordt de kaart als een knop gezien: Een knop met identificatie nummers boven de 100. Bij een klik wordt de kaart geselecteerd, de kaart heeft gekleurde figuurtjes in plaats van tekst, door middel van slepen kan de kaart van locatie worden gewijzigd enzovoort. Al deze acties van kaarten worden in de volgende paragrafen beschreven.

Omdat de kaart-knoppen apart getekend moeten worden, moet er een aparte tekenfunctie zijn. Deze functie gaat *drawCard* heten. Deze functie hoeft alleen een locatie van de knop binnen het *buttonData* array te weten, zodat de functie zelf de benodigde data daaruit kan halen. De kleuren die deze functie gebruikt om te tekenen komen uit de variabelen die beginnen met “COL\_”.

Dan komen er nog twee functies die de standaard locatie van kaarten op het scherm berekent. De eerste is *getDefaultCardLocation*. Deze functie kan aan de hand van de rij en column van een kaart de standaard x en y pixel locatie op het scherm terug geven. De functie *getSelectedCardLocation* geeft een x,y,breedte en hoogte van de schermlocatie van de selectie vakjes linksonder. Deze twee locatie functies kunnen gebruikt worden om de kaarten van locatie te verplaatsen.

Zoals eerder beschreven, heeft elke knop een identificatie nummer. Dit button id nummer kan worden omgevormd tot een kaart identificatie nummer. Dit nummer wordt gebruikt om onderandere de locatie op het scherm te bepalen. Deze omvorming wordt gedaan door *buttonIdToCardID*. Het terug vormen van kaart id naar knop id wordt gedaan door de functie cardIdToButtonId.

Wil je nu een kaart identificatie nummer vinden dat nog niet gebruikt is, dan moet er een zoekactie gestart worden. Deze zoekactie gaat verzorgd worden door de functie *getEmptyCardID*.

Nu liggen de kaarten op het scherm, maar nu kan er nog niks mee gedaan worden. Er zijn drie functies die de kaart acties verzorgen. De eerste is *cardPressAction*. Deze functie wordt aangeroepen als de muis wordt ingedrukt boven een kaart. Binnen deze functie moet de kaart locatie gelinkt worden aan de locatie van de muis.

Als de muis over het game scherm ingedrukt bewogen wordt, dan is er een kans dat er een kaart verschoven moet worden. Deze check en de daaropvolgende actie wordt geregeld door de functie *cardDragAction*.

Als laatste kan de muis losgelaten worden. Dan moet de vastgepakte kaart los gelaten worden. Als de kaart niet is verplaatst, dan moet de kaart naar een vrij selectiehokje worden verplaatst. Deze selectie wordt in het hoofdstuk Selectie beschreven.

Dan is er nog de knop “Order Cards”. Bij een druk op deze knop moeten de kaarten op de originele plek worden gelegd. Dit ordenen wordt gedaan door de functie *orderCards*. Deze functie gaat alle kaarten langs en zet ze op de standaard plek die wordt berekend door de functie *getDefaultCardLocation*.

## Selectie

In dit hoofdstuk wordt uitgelegd hoe het ‘selectie’ gedeelte van de game werkt. Met het selectie gedeelte worden de drie hokjes linksonder in het game scherm bedoeld. Als op een kaart wordt geklikt moet de kaart naar het selectie gedeelte worden verplaatst. Bij een klik op de geselecteerde kaart moet de kaart terugspringen naar de originele positie.

Er komen maar vier functies die dit gedeelte gaan beheren. Als eerste heb je *addSelectedCard* en *removeSelectedCard*. De functionaliteit van deze zijn vanzelfsprekend. Je geeft een kaart, en deze wordt aan de selectie toegevoegd of verwijderd.

Om nu alle geselecteerde kaarten in één keer weg te halen, komt er de functie *removeAllSelectedCards*.

Als laatste is er de functie *isSelectedSpotFree*. Deze functie geeft waar of onwaar terug, afhankelijk of er een lege plek bij de selectie is.

## Game

Met game wordt het spel zelf bedoeld dat gebruikt maakt van de onderdelen ‘kaarten’ en ‘selectie’. Ook valt hier het opslaan en laden van games onder.

Ten eerste komt er de functie *startGame*. Deze functie start het spel op. Als eerste wordt een popschermpje met een invoervak getoond met de vraag wat de naam van de speler is. De speler-naam wordt onthouden om aan het einde van de game de naam samen met de score toe te voegen aan het scorebord. Daarna worden alle variabelen geïnitialiseerd. Zo wordt ook een stapel kaarten gemaakt. Ook wordt hier de stapel geschud en wordt de game timer gestart. Deze functie heeft als argument een boolean die aangeeft of het spel op originele of simpele modus gestart moet worden.

Als tweede komt er de functie *updateGameInfo*. Deze update zo nodig de status van de knop “Hand In”. Als er een set geselecteerd is wordt deze knop groen en als er een verkeerde set geselecteerd is wordt deze knop rood. Daarbij verandert ook de tekst van de knop.

Daarnaast kijkt de *updateGameInfo* of de game over is. Als er geen mogelijke sets meer zijn, dan is de game klaar en moet de score worden toegevoegd aan het score bord. Daarna moet er een overlay met “Game Over” worden weergegeven. Dit wordt allemaal geregeld binnen de *updateGameInfo* functie.

Maar hoe weet de *updateGameInfo* functie hoeveel mogelijke sets er zijn? Dit wordt onderzocht door de functie *getPossibleSets*. Deze functie gaat alle groepjes van drie binnen de kaarten langs door middel van drie loopjes. Al deze groepjes worden door de functie *isSet* gehaald. Als het groepje een set is, dan wordt het aantal verhoogd met één. Het totale aantal wordt teruggegeven. Deze functie hoeft alleen aangeroepen te worden als de kaarten op het scherm veranderen. Dit is alleen als er kaarten worden ingeleverd. Waar deze update precies ziet wordt tijdens de ontwikkeling uitgevogeld.

Dan moet er een set ingeleverd worden. Dit inleveren mag alleen gebeuren als er een set geselecteerd is. Dit uitzoeken is een mooie taak voor de functie genaamd *isSetSelected*. Deze functie gebruikt de *isSet* functie om te vinden of een set geselecteerd is.

Dan is er nog de knop “Give Hint”. Bij een druk op deze knop moeten twee kaarten geselecteerd worden. De functie achter deze functionaliteit is *giveHint*. Deze functie moet een set vinden tussen alle kaarten. Het mooie is dat er al een functie bestaat die dit doet: *getPossibleSets*. De functie *getPossibleSets* en *giveHint* gaan een gezellige toekomst tegemoet, ze gaan namelijk samen werken.

Deze samenwerking wordt verzorgd door één variabele genaamd *hintSet*[]. Op het moment dat er door de functie *getPossibleSets* een eerste set gevonden wordt, dan wordt deze set opgeslagen in het *hintSet* array. Op het moment dat *giveHint* wordt aangeroepen kijkt *giveHint* in het *hintSet* array en gebruikt deze data om de eerder uitgezochte set te selecteren. Hierdoor hoeft de giveHint functie niet zelf opzoek naar sets, maar kan *giveHint* meeliften op het werk van *getPossibleSets*.

Daarna is er de knop “Hand in Set”. Deze knop moet in het geval van een geselecteerde set de set inleveren en drie nieuwe kaarten op het scherm plaatsen. De functie die dit doet is *handInSet*.

Dan zijn er nog twee functies die het opslaan en laden van een game verzorgen. Deze functies heten *loadGame* en *saveGame*. De *loadGame* functie moet een nieuwe game starten met de geladen data terwijl de *saveGame* de data uit de variabelen haalt en deze opslaat in een bestand. Deze functies werken volgens het bestandsformaat zoals dat is beschreven in het hoofdstuk Bestandstype.

Als laatste is er de functie *quitGame*. Deze functie sluit het game scherm af en geeft het menu weer. Deze functie doet niks met het score bord. Bij aanroepen van deze functie wordt het spel abrupt afgebroken en is alle data verloren!

## Timing

Om bij te houden hoe lang de speler aan het spelen is binnen de game, moet een tijd bijgehouden worden. De makkelijkste manier is het gebruik van de Unix Time Stamp. In het begin van het spel wordt de huidige timestamp opgeslagen. Daarna kan op elk moment de verlopen tijd gevonden worden door de huidige timestamp af te trekken van de begin timestamp. De drie functies die dit regelen zijn *getUnixTime*, *getTimeSeconds* en *getTimerString*. Deze drie functies spreken voor zich.

## Bestandtype

Er worden op twee plekken in het programma data opgeslagen. Hiervoor zijn ook twee bestandstypes voor nodig.

**game**time:12312312354454  
gameType:0  
wrongSets:5  
name:Sijmen  
cardsOnScreen:RON1;RON2;RON3;BQN1;RQN2;RQN3;PTY1;PTY2;PTY3cardsInStack:RON1;RON2;RON3;BQN1;RQN2;RQN3;PTY1;PTY2;PTY3

Dit is het bestandstype dat een game opslaat. Niet eerder genoemd is ‘time’; de speelduur in seconden tot nu toe gespeeld. Het enige nieuwe is de time: dit is de begin tijd in seconden tot nu toe gespeeld. Verder staat gameType 0 voor simple mode, en 1 voor Original mode.

**scores**  
Sijmen;0:40;Bram;2:50;David;3:00;Max;4:20;Ramon;5:50  
Sijmen;2:40;Bram;2:50;David;3:00;Max;4:20;Ramon;5:50

In een .scores bestand staan altijd twee regels. Elke regel bevat een even aantal stukjes. Deze stukjes zijn gescheiden door een puntkomma. Dit betekent dat de tekst die je opslaat nooit een puntkomma mag bevatten. Je kan steeds groepjes maken van twee stukjes. In het eerste stukje staat een naam, en in het tweede stukje staat een score. In het derde stukje staat weer een naam en in het vierde stukje weer een score.

In elke regel in het scores bestand staat één scorelijst. Bij dit programma bevat de eerste regel de scores van de simpele versie, en de tweede regels is de scores van de originele versie.

# Test Cases

In dit hoofdstuk wordt een plan gemaakt voor het testen van het programma. Dit testen gaat op twee manieren gebeuren. Als eerste worden de losse functies getest. Hierbij wordt gekeken of de beschreven invoerwaarde overeen komt met de verwachte uitvoerwaarde. Daarna wordt een algemene test gedaan waarbij aan de hand van de eisen gekeken wordt of alles werkt.

## isSet

De belangrijkste functie van het programma is de functie *isSet*. Deze moet dus goed getest worden. Hieronder is een tabel te vinden met de invoerwaarden en de verwachte uitkomst.

|  |  |
| --- | --- |
| **Invoerwaarde** | **Uitvoerwaarde** |
| RE1O, BQ2G, YT3O | False |
| AAAA, BBBB, CCCC | True |
| ABC, ABC, ABC | True |
| CCC, CCC, CCC | True |
| CAC, CCC, CCC | False |
| CCC, AAA, CCC | False |

Deze tests zijn uitgevoerd op 30-10-2014 en zijn allemaal geslaagd.

## GetPossibleSets

Deze functie is erg fout-gevoelig sinds hier een veel loopjes en statements gebruikt worden. Om dit te testen moet bij elke gevonden set de set code en het kaartnummer in de console geprint worden. De verwachte uitvoer is dat het aantal regels gelijk is aan het getal, dat er geen dubbele combinaties van kaartnummers geprint zijn en dat alle genoemde sets te vinden zijn op het scherm. Deze test moet meerdere keren uitgevoerd worden.

Deze test is drie keer uitgevoerd op 30-10-2014 en is drie keer geslaagd.

## getCardStack

De *getCardStack* functie moet getest worden omdat bij een fout in deze functie, het spel niet speelbaar is. Er zijn twee mogelijke uitvoeren. De eerste is bij simpele modus, waarbij een uitvoer van 27 unieke kaarten is waarbij elke kaart een ‘N’ bevat en elke kaart vier karakters lang is. De tweede mogelijke heeft 81 kaarten bij originele modus. Ook hier zijn alle kaarten uniek.

De uitvoer bij testen is:   
StringList size=27 [ 0: "REN1", 1: "REN2", 2: "REN3", 3: "RQN1", 4: "RQN2", 5: "RQN3", 6: "RTN1", 7: "RTN2", 8: "RTN3", 9: "BEN1", 10: "BEN2", 11: "BEN3", 12: "BQN1", 13: "BQN2", 14: "BQN3", 15: "BTN1", 16: "BTN2", 17: "BTN3", 18: "CEN1", 19: "CEN2", 20: "CEN3", 21: "CQN1", 22: "CQN2", 23: "CQN3", 24: "CTN1", 25: "CTN2", 26: "CTN3" ]

StringList size=81 [ 0: "REO1", 1: "REG1", 2: "REP1", 3: "REO2", 4: "REG2", 5: "REP2", 6: "REO3", 7: "REG3", 8: "REP3", 9: "RQO1", 10: "RQG1", 11: "RQP1", 12: "RQO2", 13: "RQG2", 14: "RQP2", 15: "RQO3", 16: "RQG3", 17: "RQP3", 18: "RTO1", 19: "RTG1", 20: "RTP1", 21: "RTO2", 22: "RTG2", 23: "RTP2",, 24: "RTO3", 25: "RTG3", 26: "RTP3", 27: "BEO1", 28: "BEG1", 29: "BEP1", 30: "BEO2", 31: "BEG2", 32: "BEP2", 33: "BEO3", 34: "BEG3", 35: "BEP3", 36: "BQO1", 37: "BQG1", 38: "BQP1", 39: "BQO2", 40: "BQG2", 41: "BQP2", 42: "BQO3", 43: "BQG3", 44: "BQP3", 45: "BTO1", 46: "BTG1", 47: "BTP1", 48: "BTO2", 49: "BTG2", 50: "BTP2", 51: "BTO3", 52: "BTG3", 53: "BTP3", 54: "CEO1", 55: "CEG1", 56: "CEP1", 57: "CEO2", 58: "CEG2", 59: "CEP2", 60: "CEO3", 61: "CEG3", 62: "CEP3", 63: "CQO1", 64: "CQG1", 65: "CQP1", 66: "CQO2", 67: "CQG2", 68: "CQP2", 69: "CQO3", 70: "CQG3", 71: "CQP3", 72: "CTO1", 73: "CTG1", 74: "CTP1", 75: "CTO2", 76: "CTG2", 77: "CTP2", 78: "CTO3", 79: "CTG3", 80: "CTP3" ]  
De test is dus geslaagd.

## shuffleStack

Om te zorgen dat de kaarten niet altijd de zelfde plek krijgen, moet de stapel geschud worden. Dit schudden moet random zijn. Het is erg lastig om te testen of iets random is. Daarom ga ik drie keer een stapel schudden en het resultaat uitprinten. Als ik na vijf minuten zoeken geen structuren binnen de kaarten kan vinden, dan noem ik het schudden ‘random’.

De uitvoer is:  
StringList size=27 [ 0: "CQN1", 1: "RTN3", 2: "BEN1", 3: "RTN1", 4: "CEN2", 5: "BQN2", 6: "CEN3", 7: "BEN3", 8: "BQN1", 9: "BTN2", 10: "CQN2", 11: "RQN1", 12: "RQN3", 13: "CTN1", 14: "REN2", 15: "BQN3", 16: "REN1", 17: "BTN1", 18: "RTN2", 19: "CQN3", 20: "CTN2", 21: "BEN2", 22: "RQN2", 23: "REN3", 24: "CEN1", 25: "BTN3", 26: "CTN3" ]

StringList size=27 [ 0: "BTN3", 1: "REN3", 2: "REN2", 3: "RTN1", 4: "BEN2", 5: "BTN1", 6: "CEN3", 7: "BTN2", 8: "RTN2", 9: "BQN1", 10: "BQN3", 11: "CQN1", 12: "CQN2", 13: "BEN3", 14: "BQN2", 15: "CQN3", 16: "RTN3", 17: "CEN2", 18: "RQN2", 19: "CTN2", 20: "RQN3", 21: "REN1", 22: "RQN1", 23: "CEN1", 24: "CTN1", 25: "BEN1", 26: "CTN3" ]

StringList size=27 [ 0: "CEN1", 1: "CEN2", 2: "BQN1", 3: "BQN2", 4: "CEN3", 5: "REN3", 6: "RTN3", 7: "BEN3", 8: "BQN3", 9: "REN2", 10: "BTN1", 11: "RQN1", 12: "CTN2", 13: "RQN3", 14: "BEN1", 15: "RTN2", 16: "REN1", 17: xz"BTN2", 18: "RQN2", 19: "RTN1", 20: "CQN2", 21: "CTN1", 22: "CQN1", 23: "CQN3", 24: "BEN2", 25: "BTN3", 26: "CTN3" ]

De test is dus gefaald. Dit betekent dat de code aangepast moet worden.  
Vijf minuten later…..  
Een nieuwe test wordt uitgevoerd met volgende uitvoer:  
StringList size=27 [ 0: "REN1", 1: "BEN1", 2: "RTN1", 3: "CTN3", 4: "CQN1", 5: "CQN2", 6: "BEN3", 7: "BEN2", 8: "BQN3", 9: "CQN3", 10: "BQN1", 11: "RTN2", 12: "CEN1", 13: "BTN1", 14: "BTN2", 15: "CEN3", 16: "BQN2", 17: "RQN3", 18: "CTN1", 19: "RQN1", 20: "RTN3", 21: "CTN2", 22: "REN2", 23: "BTN3", 24: "REN3", 25: "CEN2", 26: "RQN2" ]

StringList size=27 [ 0: "BEN1", 1: "BTN1", 2: "BEN3", 3: "CQN1", 4: "BQN2", 5: "RQN2", 6: "CTN1", 7: "RTN1", 8: "CTN2", 9: "CEN2", 10: "CEN1", 11: "RTN3", 12: "BTN3", 13: "CQN2", 14: "RQN3", 15: "RQN1", 16: "CTN3", 17: "BQN1", 18: "REN2", 19: "CQN3", 20: "RTN2", 21: "BTN2", 22: "BEN2", 23: "BQN3", 24: "REN3", 25: "REN1", 26: "CEN3" ]

StringList size=27 [ 0: "BTN1", 1: "CTN1", 2: "BQN3", 3: "CEN2", 4: "CQN2", 5: "CQN3", 6: "BEN2", 7: "BTN2", 8: "BQN1", 9: "REN2", 10: "CQN1", 11: "BQN2", 12: "BEN1", 13: "RQN1", 14: "REN1", 15: "RQN3", 16: "RTN2", 17: "BEN3", 18: "RQN2", 19: "CTN2", 20: "BTN3", 21: "RTN1", 22: "CTN3", 23: "REN3", 24: "CEN3", 25: "RTN3", 26: "CEN1" ]

De test is geslaagd. Het probleem is dus verholpen in update ‘…ee7c’[[3]](#footnote-3).

# Reflectie

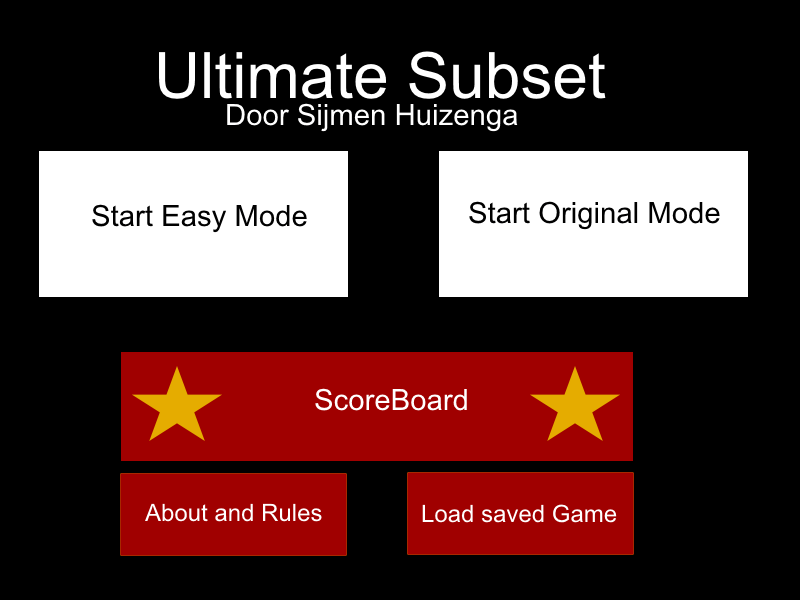
Ik vind SET een erg leuk spel. Niet om te spelen, maar wel om te analyseren. Ook de wiskunde achter de mogelijkheden en opties was erg interessant. Ik zou graag een verslag schrijven over alle wiskunde achter dit spelletje, maar dat is niet de opdracht.

Zelf heb ik al enige ervaring met object georiënteerd programmeren in Java. Omdat in deze opdracht geen objecten gebruikt mochten worden, moest ik terug schakelen naar een ‘lager’ niveau. In het begin leek dit geen probleem te zijn. Maar toen ik de duizend regels code naderde bleek dat mijn code toch beduidend minder overzichtelijk is geworden dan een denkbeeldig object georiënteerd programma in Java. Door het gebruik van veel functies en tabs, heb ik geprobeerd de code leesbaar te houden.

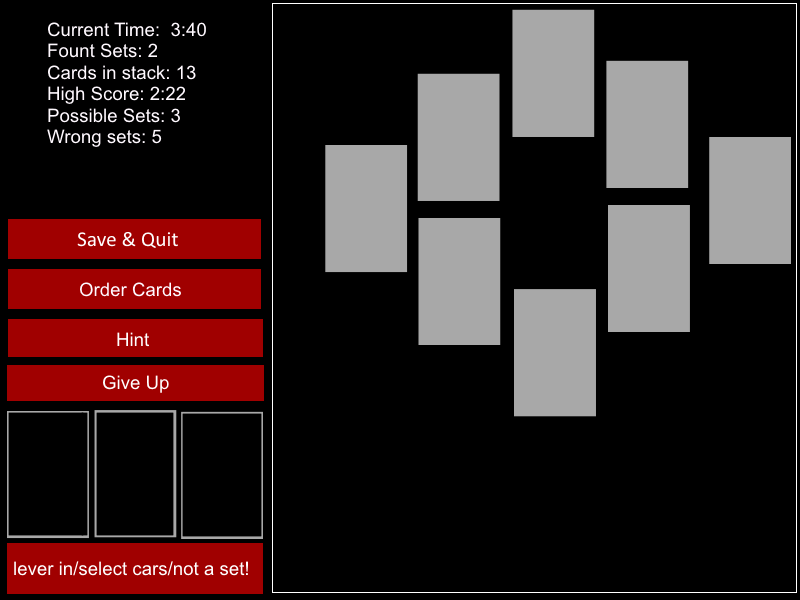
Ik vind het jammer dat ik niet helemaal heb kunnen laten zien wat ik in huis heb.

# Bijlage

## Schermontwerp menu



## Schermontwerp game



## Schermontwerp scoreboard



## Schermontwerp about



1. <https://raw.githubusercontent.com/SijmenHuizenga/SubSet/master/afbeeldingen/27Kaarten.png> [↑](#footnote-ref-1)
2. <https://raw.githubusercontent.com/SijmenHuizenga/SubSet/master/afbeeldingen/81Kaarten.png> [↑](#footnote-ref-2)
3. De link naar de update: <https://github.com/SijmenHuizenga/SubSet/commit/326695120db277fb20076796874fb75b4a62ee7c> [↑](#footnote-ref-3)