# Ontwerp

Het programma is veel te groot om alles te ontwerpen. Als ik dit zou doen, zou het programma nooit af komen. Daarom wordt hier de algemene flow uitgelegd. Daarna worden de vier belangrijkste functies verder uitgelegd.

## Schermen

Er zijn meerder schermen. Het huidigge 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 gaan, 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. Het tekenen van schermen zijn ook een aantal functies voor: *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

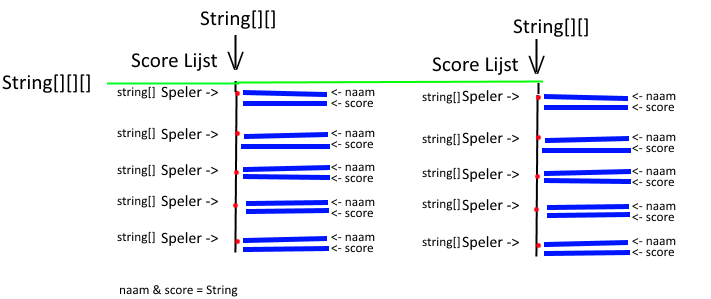
Door het hele programma worden knoppen gebruikt. Alle informatie over knoppen is opgeslagen in twee arrays: *buttonData[][]* en *buttonTxt[]*. Als je het identificatie nummer van een knop weet, kan je met de functie *buttonIdToLocation* het identificatie nummer omzetten naar een locatie nummer. Met dit 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 ‘BUTON\_’. Een nieuwe knop toevoegen kan met de functie *addButon*.

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

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 wat 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*. Als laatste is er de *drawScoreList* functie. Hierin wordt een score lijst getekend.

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.

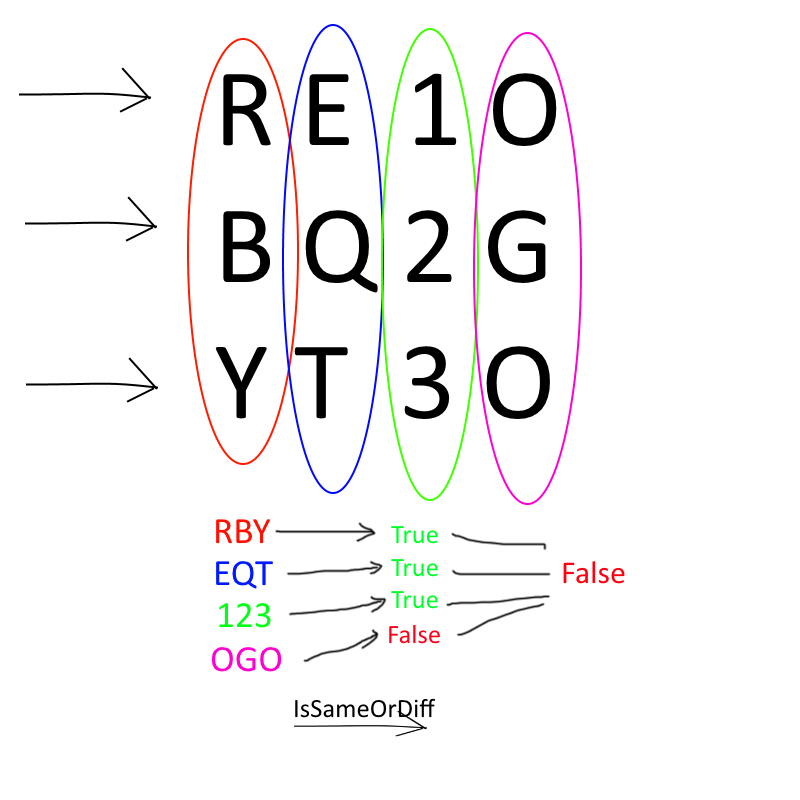
Om nieuwe scores toe te voegen is er de functie *addScoreEntry*. Deze doet niks anders dan een row toevoegen in het

## 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 word 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.

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 moet 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 gaat zij dat doen? 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 gebeurd voor elke eigenschap. Als één of meer van de eigenschapen de keuring niet doorkomen, dan is het geen set.

De keuring of een String uit helemaal de zelfde 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 de zelfde karakters, en de andere kijkt of de string alleen maar uit verschillende karakters bestaat. Als een van de twee waar is, moet waar terug gegeven worden.

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 stapel van de stapel kaarten (*stack*), en voegt deze toe aan het scherm. Dit doet hij door een speciale knop toe te voegen: Een knop met een identificatie nummer boven de 100. Alle knoppen met identificatie nummers boven de 100 worden behandeld als fysieke kaarten: 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 teken functie 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.