SubSet door Sijmen Huizenga

# Systeem

Je kan het programma verdelen in twee onderdelen: Component en Game. Met component bedoel ik alle knoppen, checkboxes, invoervakjes en verschillende schermpjes. Als er bijvoorbeeld een knop op het hoofdmenu staat ‘start game’, en je klikt op deze knop, dan wordt deze klik gehandeld door het component gedeelte van het programma. De game zijn de game mechanieken die het spel verzorgen. Zo houdt de game bijvoorbeeld de punten in de gaten.

## Component

Laten we bij het begin beginnen. Elk scherm(menu, game, scores, about) heeft een nummer. Door middel van dit nummer wordt bijgehouden welk scherm op dit moment wordt weergegeven. Om de schermen aan de nummer te linken, zijn er vier variabele die aangeven welk nummer bij welk scherm hoort:

**int SCREEN\_MENU** = 0; **int SCREEN\_GAME** = 1; **int SCREEN\_SCORES** = 2; **int SCREEN\_ABOUT** = 3;

Om aan te geven welk scherm nu wordt weergegeven gebruik je: **int selectedScreen** = SCREEN\_MENU;

Als dan de het scherm getekend moet worden, wordt naar de ‘selectedScreen’ variabele gekeken en wordt deze gebruikt om de juiste draw functie aan te roepen.  
**swich(selectedScreen){  
 case:** SCREEN\_MENU: drawMenu(); **break;  
 case:** SCREEN\_GAME: drawGame(); **break;  
 case:** SCREEN\_SCORES: drawScores(); **break;  
 case:** SCREEN\_ABOUT: drawAbout(); **break;  
}**

Naast schermen zijn er knoppen. Zoals te zien is in de ‘future screenshots’ zijn er erg veel verschillende knoppen(ongeveer 12). Het is onmogelijk om voor elk van deze knoppen een aparte x, y, breedte en hoogte bij te houden. Daarom gebruik ik en 2D array:

Int buttonAmount = 3;  
**String[][] buttonData = new buttonData[**buttonAmount**][**8**];**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Screen | X | Y | Width | Height | Background Color | Font Color |
| 1 | **SCREEN\_MENU** | 100 | 300 | 300 | 200 | # FFFFFF | # 00 |
| 2 | **SCREEN\_MENU** | 500 | 300 | 300 | 200 | # FFFFFF | # 00 |
| 101 | **SCREEN\_MENU** | 200 | 600 | 500 | 150 | # A00000 | # FFFAFF |

Omdat het verwarrend kan zijn welke locatie in het array corresponderen met welke eigenschap, zijn er een aantal variabelen die dit aangeven:

int **BUTTON\_ID** = 0;int **BUTTON\_SCREEN** = 1;int **BUTTON\_X** = 2;int **BUTTON\_Y** = 3;int **BUTTON\_WIDTH** = 4;int **BUTOTN\_HEIGHT** = 5;int **BUTTON\_BGCOLOR** = 6;int **BUTTON\_FGCOLOR** = 7;

Om handig met dit array te kunnen werken is er een functie in het leven geroepen die een button locatie te vinden aan de hand van ID nummer:

**int getButtonLocation(int id){  
 int i = 0;  
 for(int[] but :** buttonData**){  
 if(but[**BUTTON\_ID**] == id)  
 return but;  
 i++;  
 }  
 return -1;  
}**Zoals te zien is, wordt bij een invalide id -1 teruggegeven.

Om te zorgen dat elke knop ook een tekst heeft, moet er een String array komen:

**String[] buttonText = new String[**buttonAmount**];**

In dit array is de locatie gelijk aan de locatie in het buttonData array. Dus als je de locatie van een button in dat array hebt, heb je ook de locatie van de tekst in dit array. Voorbeeldje:

*buttonText[getButtonLocation(2)];*

Dit geeft de tekst op de knop met id nummer 2.

Op het moment dat ergens in het scherm geklikt wordt, worden bij alle knoppen die in het huidige scherm aanwezig zijn gekeken of er op die knop gedrukt wordt. Als een knop ingedrukt wordt, wordt de juiste functie aangeroepen. Dit ziet er dan ongeveer zo uit: **for(int[] but :** buttonData**){  
 if(but[**BUTTON\_SCREEN**] == selectedScreen){  
 if(mouseX > but[**BUTTON\_X**] && mouseX < (but[**BUTTON\_X**]+but[**BUTTON\_WIDTH**])  
 && mouseY > but[**BUTTON\_Y**] && mouseY < (but[**BUTTON\_Y**] + but[**BUTOTN\_HEIGHT**])){  
 doButtonAction(but[BUTTON\_ID];  
 }  
 }  
}**

En dan nu de functie doButtonAction met als argument één integer die het id nummer aangeeft:  
**swich(id){  
 case:** 0: goToMenu(); **break;  
 case:** 1: startGame(false); **break;  
 case:** 2: startGame(true); **break;  
 case:** 3: showScoreBoard(); **break;  
 case:** 5: showAbout(); **break;  
 case:** 6: loadLastGame(); **break;  
 case:** 7: saveGame(); **break;  
}  
if(id >= 100){** onCardClicked**(id-100);  
}**

## Game

Het game gedeelte bevat alle informatie over het spel. Tijdens dit hoofdstuk ga ik er vanuit dat duidelijk is hoe het spel werkt, en wat de verschillen zijn tussen simple en original versie.

|  |  |  |
| --- | --- | --- |
| 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) |

Ten eerste zijn er kaarten. Om een kaart met zijn eigenschappen weer te geven wordt een String gebruikt. In deze analyse wordt een string die een kaart voorstelt een Kaart genoemd. Een Kaart bevat ten alle tijden 4 karakters. Elk karakter geeft een eigenschap aan. Hiernaast is in een tabel weergegeven wat de karakters betekenen. Een Kaart bevat dus één karakter uit het bovenste vak, een uit het tweede vak enzovoort. Een voorbeeld van een Kaart zou kunnen zijn “ROQ1” of “BPQ3”.

Om een stapel kaarten weer te geven wordt een String array gebruikt. Hier is een schets van een functie die een array maakt met alle mogelijk kaarten:

**String[]** getCardSet**(boolean** simple**){  
String[] out = new String[simple ? 27 : 81];  
 int counter = 0;  
 for (int i = 1; i <=3; i++) {  
 for (int j = 1; j <=3; j++) {  
 for (int k = 1; k <=3; k++) {  
 if(simple){  
 out[counter] = makeCardString(i, j, k, 4);  
 counter++;  
 }else{  
 for(int l = 1; l <=3; l++){  
 out[counter] = makeCardString(i, j, k, l);  
 counter++;  
 }  
 }  
 }  
 }  
 }  
return out;  
}**

Zoals hierboven te zien is bestaat er een functie makeCardString(int i, int j, int k, int l). Het enigge wat deze functie doet is door middel van die vier nummers een string maken met de bijpassende karakters. Dus i=2 refereert naar B, i = 3 naar Yellow, j=3 T. Dit is gewoon een aantal swich statments die de getallen omzetten naar karakters.

Om te zorgen dat de speler niet alle kaarten op volgorde voorgeschoteld krijgt, moet de stapel geschud worden. Dit gebeurt door de functie shuffleArray. Deze neemt als argument een String array, en geeft een geschud array terug.