INLEDNING

Ivana Orz

22.12.2022

Grupp Fem – War Dolphines

https://github.com/rodercode/HangmanGruppFive.git

Hangman | Trello

Kundens vision

Kunden ville återskapa det klassiska Hangman-spelet där spelare gissar ord och om de gissar fel bokstäver blir deras gubben hängd. Hans vision var att ge alla spelare möjlighet att spela samtidigt genom att ta korta turer och inte vänta tills en spelare gissar hela ordet. Hans krav för den nya versionen av spelet var följande:

- minst två spelare som spelar samtidigt;
- möjlighet att utöka till 4 - 8 spelare;
- turbaserat system;
- familjevänligt tema;
- spelet måste spelas lokalt;
- avslutas när en spelare når en bestämd poäng;
- en spelare kan gissa ordet skapat av en annan spelare;
- Power Ups som innebär blockering av använda bokstäver.

REFLEKTION

1. Images of Cakes.

As a Player I want to see images of a cake, so that I can experience at full Hangman Game concept.

För att implementera denna user story först ritade jag elva bilder: den första innehåller representationen av en hel tårta. De efterföljande kommer med en bit mindre, och den sista bilden har ingen tårta alls. Denna sekvens av bilder representerar fel gissningar som en spelare gör. För att göra det visuellt mer effektivt och bekvämt för varje spelare att skilja mellan sin tårta och de andra spelarna, placerades varje uppsättning tårtor på en annan bakgrundsfärg. Så spelare 1 har tårtan på en rosa bakgrund, spelare 2 har tårtan på en blå bakgrund. Bilder skapades i ritprogrammet Adobe Illustrator och sparades som en png-fil. För att implementera denna user story i kod, skapades två listor med bilder: en för tårtan på rosa bakgrund och den andra för tårtan på blå bakgrund. Det skapades också en metod *addImagesToLists* som innehåller adressen till varje bild.

```
private List<String> listOfBlueCake;
3 usages
private List<String> listOfPinkCake;
```

```
listOfBlueCake = new ArrayList<>();
listOfPinkCake = new ArrayList<>();
addImagesToLists();
```

```
imageViewCakeOne.setImage(imageCakeBlue);
imageViewCakeTwo.setImage(imageCakePink);
```

```
public void addImagesToLists() {
    for (int i = 0; i < 11; i++) {
        listOfBlueCake.add("src/main/resources/cakeBlue/cakeBlue" + i + " cat.png");
        listOfPinkCake.add("src/main/resources/cakePink/cakePink" + i + " cat.png");
    }
}
```

2. Animation of the Cake.

As a player I want to be able to see pieces of cake disappear when I make wrong guesses, so I know how many guesses are left for me.

Den här user story är en fortsättning på den föregående. Dess syfte är att få bilder av tårtan att förändras och visa hur delar av tårtan försvinner beroende på de misstag som spelarna gör. Om spelaren gissar en bokstav fel försvinner en tårtbit. För att implementera denna user story skapades två identiska metoder: en för tårtan på rosa bakgrund för Player One och den andra för tårtan på blå bakgrund för Player Two. Den ena metoden kallades *importImagePinkCake,* den andra *importImageBlueCake*. Eftersom animering av tårtan sker när spelaren gör misstag, gjordes en hänvisning till variabeln PlayerMistakes från DataBase-klassen, och Player One(1) och Player Two(2) indikerades i sina respektive metoder. I början skapade jag animationsmetoden bara för en spelare. Senare, när ytterligare en spelare lades till fanns det ett behov av att göra ändringar i koden. Även om den första varianten av metoden fungerade bra, innehöll den för mycket kod och med hjälp av teammedlemmar gick det att göra det kortfattat.

```
public void importImagePinkCake(){
    try {
        imageCakePink = new Image(new FileInputStream(listOfPinkCake.get(database.getPlayerMistakes().get(1))));
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
}
```

```
public void importImageBlueCake(){
    try {
        imageCakeBlue = new Image(new FileInputStream(listOfBlueCake.get(database.getPlayerMistakes().get(2))));
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
}
```

3. “Player’s Turn” label.

As a player I want to see an inscription that reads “Player’s Turn”, so to know when it is my turn to guess a letter.

För att implementera denna user story använde jag ett speciellt program som heter Scene Builder som har drag and drop principen. I den, valde jag ett verktyg som heter Label som gör det möjligt att skriva en text. Med teckensnittsstilen Comic Sans MS skrev jag "Player's Turn" för varje spelare i deras Label. Jag gav också varje Label ett unikt id-namn (playerPlate1, playerPlate2) så att det kan användas i koden. På .fxml filen koden implementerades så här:

```
<Label fx:id="playerPlate1" layoutX="77.0" layoutY="32.0" prefHeight="67.0" prefWidth="209.0" text="Player 1's turn" textFill="#f797c7">
   <font>
      <Font name="Comic Sans MS" size="28.0" />
   </font>
</Label>
<Label fx:id="playerPlate2" layoutX="710.0" layoutY="32.0" prefHeight="67.0" prefWidth="209.0" text="Player 2's turn" textFill="#6db8e4">
   <font>
   <Font name="Comic Sans MS" size="28.0" />
   </font>
</Label>
```

På .java filen skapade jag två variabler som skulle användas i koden.

```
@FXML
private Label playerPlate1;
3 usages
@FXML
private Label playerPlate2;
1 usage
```

AVSLUTNING

Efter mötet med kunden och brainstorming av idéer angående spelet skapade vår grupp en backlog med många user stories. De var alla bra och en lade till en ny funktionalitet till spelet. Naturligtvis hade de en hierarki av betydelse. Att implementera dem alla slumpmässigt var inte rimligt. Istället anpassade vi tillvägagångssättet att prioritera dem. Först implementerade vi user stories som gav den grundläggande funktionaliteten till spelet och implementerades gradvis resten efter deras betydelse. Var och en av oss skulle ta en user story och implementera den. När implementeringen var klar skulle vi ta en ny. Sättet som user stories delades upp mellan oss baserades, antar jag, på hur vi kände för dem när det gäller färdigheter att implementera dem, intresse och vilja att prova något nytt. Att dela upp user stories mellan oss gjordes på ett taktfullt sätt. Vi skulle ge friheten att välja vilken user story en av oss ville ha och skulle respektera valet.

Redan från början skapade vi en positiv och välkomnande atmosfär i gruppen som jag antar, bidrog mycket till vår produktivitet och fick mig personligen att känna mig användbar och en del av teamet. Som jag kände det, drevs vi av principen "Om jag misslyckas, misslyckas alla; om jag lyckas, lyckas alla". Under code reviews fanns inga känslor av kritik, tvärtom var code reviews en intressant och fängslande del där vi kunde lära oss. På grund av det kunde vi koncentrera oss på att göra det bästa vi kunde. Vi hjälpte varandra i varje steg av vårt grupparbete. Jag kan säga att detta hindrade oss från att vara fast under lång tid och vi lärde oss också mycket.

I efterhand av vårt grupparbete kan jag inte säga att det var problem. Det gällde bara att vänja sig vid rutinen vi etablerade. Jag kan säga att vi förstod och vande oss snabbt. Så, varje dag, under dagens sista möte, pratade vi om vad vi skulle göra nästa dag. Detta gav en tydlig bild av var vi var och var vi måste vara nästa dag. Trello board var förstås ett mycket användbart verktyg för att styra arbetet och implementera agila metoder. Vi visste vem som implementerade vilken user story, vad vi arbetade med just nu, vilka user story som skulle implementeras och vilka som implementerades. Vi gjorde korta och ofta commits av våra short live branches till develop branch. När en stor del av koden fungerade bra i develop branch, merged vi ihop den med den main branch.