

1 - Beskrivelse av appen

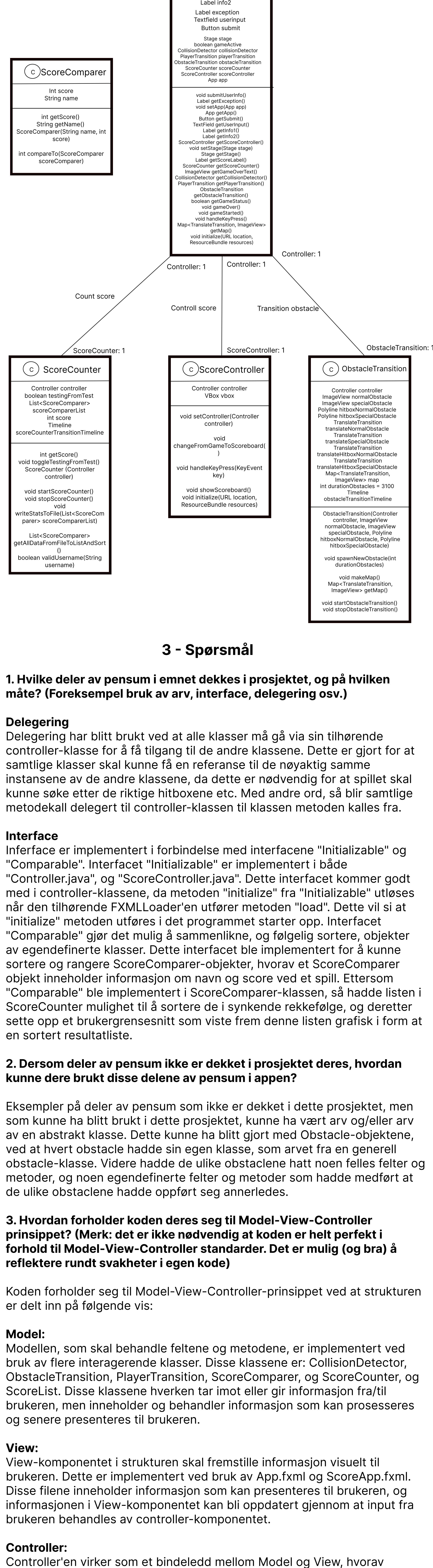
Appen jeg har utviklet er en videreutvikling av spillet som google chrome gir ut når det ikke finnes internettforbindelse. Appen er lik i den forstand at spillet går ut på å hoppe over hindre som kommer mot deg i økende fart, såvel som at scoren til spilleren øker så lenge spillet er aktivt. Spilleren har kun ett kontrolelement mens spillet er aktivt; spilleren får spillkarakteren til å hoppe ved å trykke "mellomrom".

Fundamentalt forskjellig fra spillet til google chrome, så eksisterer det et scoreboard, der alle spillere har mulighet til å legge inn navnet sitt for å bli presentert på en samlet, rangert (comparable) resultatliste som leses fra, og skrives til fil.

Videre fra scoreboardet har spilleren mulighet til å velge mellom hvorvidt de ønsker å spille igjen(ved å trykke "P"), eller om de ønsker å avslutte applikasjonen(ved å trykke "R").

Rent kosmetisk skiller applikasjonen seg ved at det er benyttet andre illustrasjoner for spillkarakteren, hindrene, og bakgrunnen.

2 - Diagram



3 - Spørsmål

1. Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (Foreksempel bruk av arv, interface, delegering osv.)

Delegering

Delegering har blitt brukt ved at alle klasser må gå via sin tilhørende controller-klasse for å få tilgang til de andre klassene. Dette er gjort for at samtlige klasser skal kunne få en referanse til de nøyaktig samme instansene av de andre klassene, da dette er nødvendig for at spillet skal kunne søke etter de riktige hitboxene etc. Med andre ord, så blir samtlige metodekall delegert til controller-klassen til klassen metoden kalles fra.

Interface

Interface er implementert i forbindelse med interfacene "Initializable" og "Comparable". Interfacet "Initializable" er implementert i både "Controller.java", og "ScoreController.java". Dette interfacet kommer godt med i controller-klassene, da metoden "initialize" fra "Initializable" utløses når den tilhørende FXMLLoader'en utfører metoden "load". Dette vil si at "initialize" metoden utføres i det programmet starter opp. Interfacet "Comparable" gjør det mulig å sammenlikne, og følgelig sortere, objekter av egendefinerte klasser. Dette interfacet ble implementert for å kunne sortere og rangere ScoreComparer-objekter, hvorav et ScoreComparer objekt inneholder informasjon om navn og score ved et spill. Ettersom "Comparable" ble implementert i ScoreComparer-klassen, så hadde listen i ScoreCounter mulighet til å sortere de i synkende rekkefølge, og deretter sette opp et brukergrensesnitt som viste frem denne listen grafisk i form at en sortert resultatliste.

2. Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?

Eksempler på deler av pensum som ikke er dekket i dette prosjektet, men som kunne ha blitt brukt i dette prosjektet, kunne ha vært arv og/eller arv av en abstrakt klasse. Dette kunne ha blitt gjort med Obstacle-objektene, ved at hvert obstacle hadde sin egen klasse, som arvet fra en generell obstacle-klasse. Videre hadde de ulike obstaclene hatt noen felles felter og metoder, og noen egendefinerte felter og metoder som hadde medført at de ulike obstaclene hadde oppført seg annerledes.

3. Hvordan forholder koden deres seg til Model-View-Controller prinsippet? (Merk: det er ikke nødvendig at koden er helt perfekt i forhold til Model-View-Controller standarder. Det er mulig (og bra) å reflektere rundt svakheter i egen kode)

Koden forholder seg til Model-View-Controller-prinsippet ved at strukturen er delt inn på følgende vis:

Model:

Modellen, som skal behandle feltene og metodene, er implementert ved bruk av flere interagerende klasser. Disse klassene er: CollisionDetector, ObstacleTransition, PlayerTransition, ScoreComparer, og ScoreCounter, og ScoreList. Disse klassene hverken tar imot eller gir informasjon fra/til brukeren, men inneholder informasjon om navn og score ved et spill. Ettersom "Comparable" ble implementert i ScoreComparer-klassen, så hadde listen i ScoreCounter mulighet til å sortere de i synkende rekkefølge, og deretter sette opp et brukergrensesnitt som viste frem denne listen grafisk i form at en sortert resultatliste.

View:

View-komponentet i strukturen skal fremstille informasjon visuelt til brukeren. Dette er implementert ved bruk av App.fxml og ScoreApp.fxml. Disse filene inneholder informasjon som kan presenteres til brukeren, og informasjonen i View-komponentet kan bli oppdatert gjennom at input fra brukeren behandles av controller-komponentet.

Controller:

Controller'en virker som et bindeledd mellom Model og View, hvorav controlleren gir informasjon til Model, Model behandler informasjonen, og gir eventuelt informasjon tilbake til Controller'en, som igjen eventuelt oppdaterer View-komponentet. Controller'en implementert for hver av View'ene, i form av "Controller.java" for "App.fxml", og "ScoreController.java" for "ScoreApp.fxml". Disse Controller'erne har forøvrig tilgang til hverandre, for å virke som et bindeledd mellom de ulike View-komponentene denne applikasjonen benytter seg av.

Etter å ha fullført prosjektet, så innser jeg at det burde ha blitt opprettet en separat klasse som hadde orden på relasjonene mellom de ulike klassene, slik at controlleren ikke hadde trengt å gjøre dette. Controlleren endte opp med å inneholde relativt mange getter-metoder som de andre klassene kunne benytte seg av, og selv om det kanskje ikke regnes som fullverdig funksjonalitet, så hadde det passet bedre overens med Model-View-

Controller prinsippet dersom controller-klassen hadde sluppet dette. De viktigste aspektene med Model-View-Controller prinsippet er imidlertid på plass ved at Model-aspektet har ansvaret for selve funksjonaliteten bak applikasjonen, og at controlleren virker som et bindeledd mellom modellen og view'et (bruksgrensesnittet).

4. Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke, hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at dere skal reflektere rundt egen bruk av tester)

Ved testing av appen, så har det blitt lagt vekt på å teste den metodene som kan testes uten å starte selve applikasjonen. Fortrikkvis har det blitt prioritert å teste metoden som sjekker av bruker-input'et av brukernavn fungerer som ønsket, da dette er det eneste konseptet ved applikasjonen som tar inn variabelt bruker-input. Videre har det blitt laget en test for å sjekke at "compareTo"-metoden i "ScoreComparer" klassen har fått rett utforming. Ettersom denne applikasjonen er et spill med et grafisk brukergrensesnitt der flere av variablene avhenger av verdier fra FXML-filen, så er det upraktisk å teste disse metodene. Disse metodene er dessuten ikke avhengig av input fra brukeren, så det heller ikke så mye rom for feil der.