Connections

Appen min “Connections” er inspirert av New York Times sin spill med samme navn. I dette spillet får man 16 ord, det er totalt 4 grupper med 4 ord som tilhører sammen i en kategori efks: “musikalske intrumenter” eller “måter å forberede egg på”. Spilleren kan gjette hvilket ord som hører sammen ved å velge ut 4 ord og trykke submit. Dersom man gjetter riktig vil navnet på kategorien bli avslørt til spilleren. I min versjon har jeg valgt å gi spilleren muligheten til å velge om de vil ha 2, 3 eller 4 ord per kategori. Hvis de velge 2, vil kun 8 ord bli vist der 2 og 2 hører sammen i en gruppe. De kan også legge til egne kategorier nederst på skjermen. I det originale spillet har spilleren kun 4 liv, men i min versjon har jeg bestemt å ikke begrense antall forsøk man får. Spilleren kan også velge å starte et nytt spill når som helst ved å trykke på “new game” knappen.

Svar på spørsmål

Pensum

Mye av appen baseres på bruken av interface og delegering. Den har en en interface som heter WorkerInterface som som blir implementert av TwoWordWorker, ThreeWordWorker og FourWordWorker. Disse klassene er ansvarlig for å velge ut ordgruppene som skal brukes i spillrunden, stokke alle ordene og sjekke om ordgruppen spilleren har gjettet er riktig. Får å sørge for at spillrunden har riktig antall ord har jeg brukt en Delegator klasse som delegerer ulike forespørsler til riktig WorkerInterface klasse avhengig av hva spilleren velger.

Metodene til Delegate klassen:

delegateAccordingToGroupSize: tar inn antall ord spilleren har valgt og lager tilsvarende ny “WorkerInterface” klasse.
delegateWordAction: tar inn ordet spilleren har trykket på og sorger for at den riktig WorkerInterface klassen legger den i ord listen.
delegateSubmission: Kaller på riktig WorkerInterface for å sjekke om ord listen er riktig.

Jeg bruker også en del streams i Appkontrolleren for å utføre samme funksjon på flere objekter. For eksempel forEach(checkBox -> checkBox.setVisible(true/false) å vise / gjemme “checkbox” knappene der ordene vises.

For å lagre alle ord-kategoriene som kan brukes i spillet har jeg en fil som heter “WordBank.txt”. Når en WorkerInterface klasse blir opprettet vil den hente alle kategoriene gjennom Save klassen. Den gjør dette ved å lese fra filen og returnere kategoriene i en List<List<String>>. Jeg har også gjort det mulig å lagre spilleren sin egen kategori i filen gjennom Save klassen.

Jeg har ikke brukt arv eller abstrakte klasser i appen. Arv kunne ha blitt brukt som en erstatning til interfaces. Alle WorkerInterface klassene er veldig like utenom de metodene der antallet av ord spillerene har valgt spiller en rolle. Jeg kunne ha lagt en abstrakt klasse som heter Worker som alle de andre arver fra og gjøre metodene: pickNewCategories() og

setupNewCategories om til abstrakte metoder som må bli definert i de klassene som arver fra den.

Model-View-Controller-prinsippet

Koden min forholder seg til Model-View-Controller-prinsippet ved å ha en AppController klasser som tar seg av alt det visuelle og inputen fra spilleren. Model-komponenten består av alle klassene som implementerer WorkerInterface. De gjør alle hånterer informasjonen programmet får fra brukeren og velger ut ordene man trenger til spillet. En stor svakhet er åpenbart at både View og Controller konponentene er styrt med app kontrolleren og ikke slik at det ikke går ann å lett skille funskjonene.

Test

testFileReading() og testFileWriting() tester om Save klassen klarer å lese og skrive fra filen: “WordBank.txt”, som har 25 linjer med informasjon. Save klassen har ansvar om å importere disse linjene og returnere en List<List<String>>. testFileReading() sjekker om Listen som blir returnert har 25 elementer. testFileWriting() bruker Save klassen til å skrive en til linje med informasjon som skal representere inputen fra spilleren. Så sjekker den om antall linjer har gått opp med 1. Hvis ikke har spillet mistet en viktig funksjon, som er at spilleren kan legge til sine egne kategorier.

testWordBank() tester om tilstandsvariabelen List<List<String>> WordGroup er tomt når man initialiserer en WorkerInterface klasse og om den blir oppdatert etter man kaller getWorkBank() metoden. Dette er igjen viktig for at spilleren får opp de ordene de trenger for å spille.

testWordTouched() tester også om ordene lagt til og fjernet fra WordGroup på riktig måte men denne gangen gjennom metoden wordTouched(String word). Dersom ordet allerede er i WordGroup listen skal den bli fjernet da dette samsvarer med om checkBox-ene i spillskjermen er merket eller ikke.

testWordGroup() tester om tilstandsvariabelen List<String> WordGroup er tomt når man initialiserer en WorkerInterface klasse og sjekker om listen blir riktig når man legger til ord. Dette er viktig for å sørge for å programmet har kontroll over det spilleren ønsker å gjette som sin ordgruppe.

testPickCategories() tester om pickNewCategories returnerer lister med riktig lengder. Listene skal ha lengde 8,12 og 16 for henholdvis 2,3 og 4 i ordgruppe størrelsene.