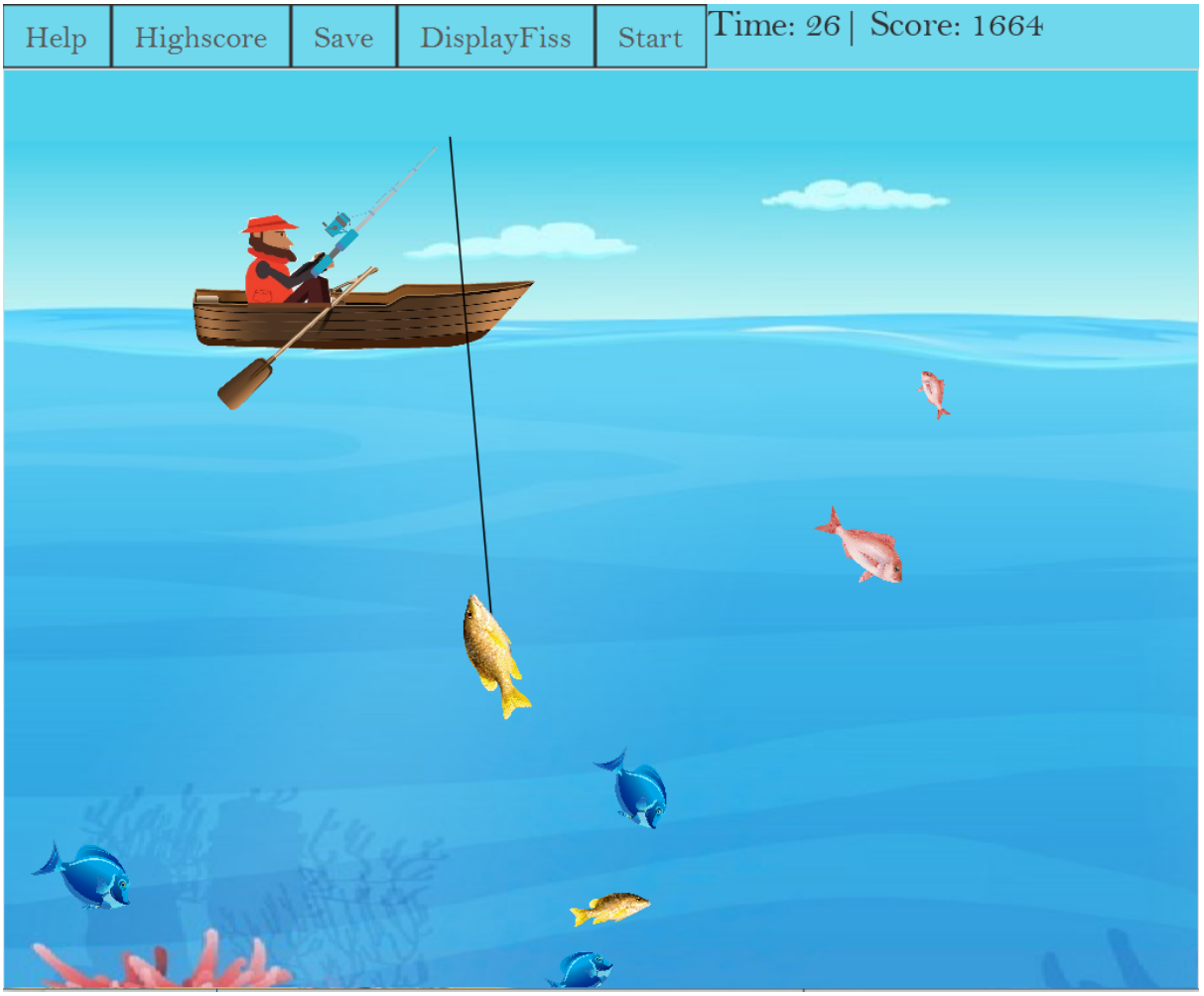
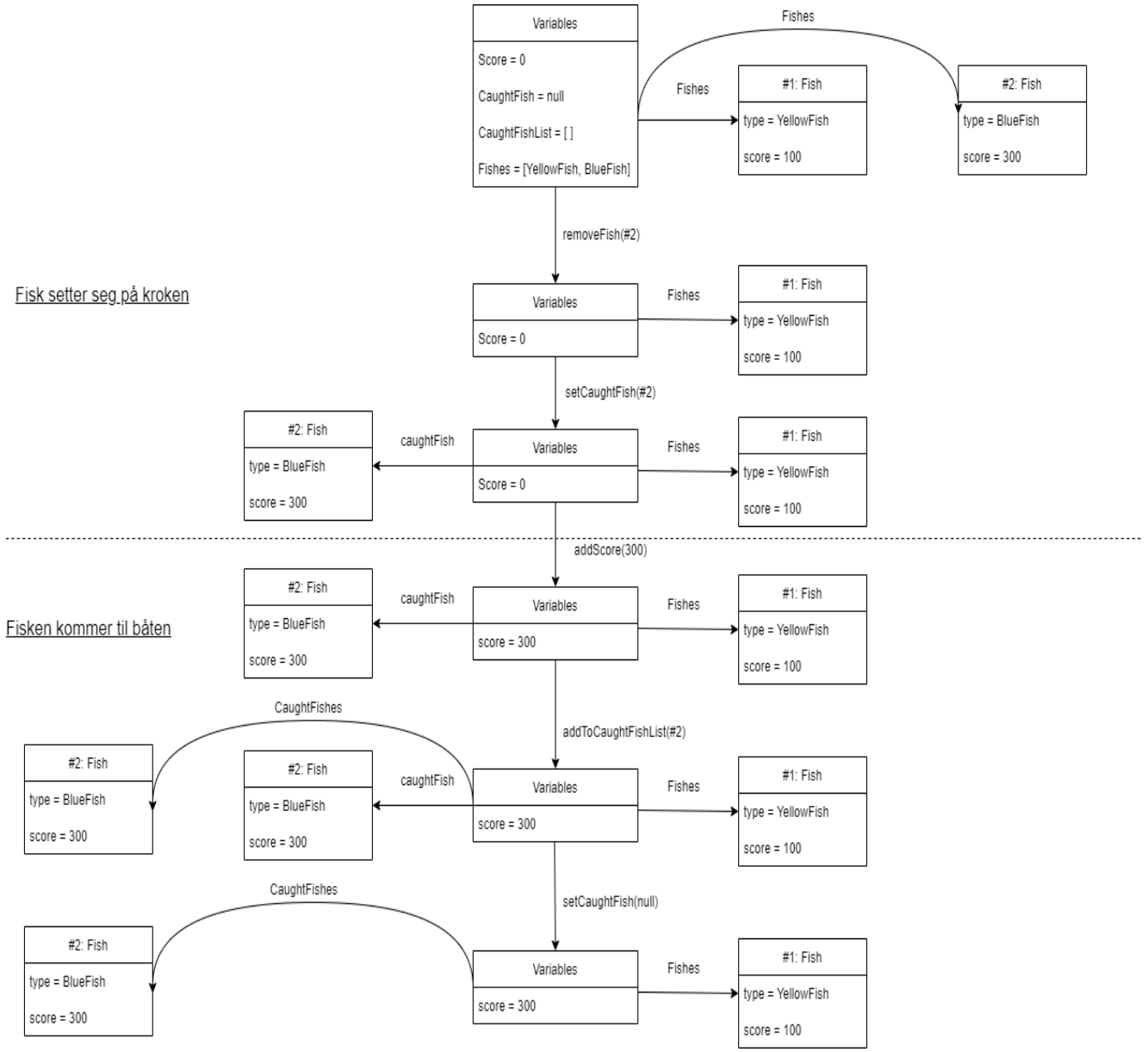
Dokumentasjon

Beskrivelse av appen

Appen er et spill som er inspirert av spillet *FishingLife* som er lansert både på iOS og Android. Når man starter spillet ser man en båt med en mann som sitter inni. Denne båten kan spilleren styre. Under båten er det fisker som svømmer rundt. Fiskene har forskjellige størrelser og dermed forskjellig antall poeng. Gameplayet er som sportsfisking. Spilleren skal bevege båten, og fiske så mange fisker som mulig innen 60 sekunder. Det antallet fisker som spilleren klarer å fiske innen tiden renner ut, samt typen fisk som blir fisket, blir lik poengsummen spilleren har når de 60 sekundene er over. Poengsummen blir lagret, og med high score systemet kan spilleren prøve å slå high scoren med å prøve å fiske flere fisker enn gangene før.



DIAGRAM



Dette ObjektTilstands Diagrammet illustrerer hvordan hendelsesforløpet og sekvens av kall endrer objekt strukturen fra en fisk setter seg på kroken til den er fisket opp i båten. Variables klassen inneholder ulike verdier knyttet til den sentrale delen av spillet. Når en fisk blir fanget blir tilstanden til denne klassen endret. CaughtFish er den fisken som er festet på kroken. Når den er dratt opp i båten blir denne fisken “tatt av kroken” og lagt til i en liste over alle fisker som er fanget iløpet av runden.

SPØRSMÅL

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

Det er mange ting i prosjektet som dekker pensumet. Klasser, objekter og metoder blir selvfølgelig brukt. Klassene blir brukt for å lage en «blueprint» for alt i spillet slik at man kan lage flere ting (instanser/objekter) av klassen (for eksempel Fish). I hver klasse er det flere metoder. Metodene er der for å få objektet til å fungere slik vi ønsker. De gir objektene funksjonalitet. Fish har for eksempel metoden setPos som tar inn en variabel av type Point2D, og metoden flytter rett og slett objektet generert av Fish, til den gitte posisjonen, ergo har vi bevegelse. Det blir også brukt interfaces for filbehandlingen. Interfacen gjør det lettere å endre på filbehandling, og endre funksjonaliteten til filbehandlingsklassen. Akkurat nå blir det brukt string-oriented filbehandling, men om man ønsker, kan det blir endret til for eksempel byte-oriented, ved at vi har en interface. I kontroller klassen brukte vi også grensesnittet Initializable som lot oss initialisere ulike ting helt på starten når spillet kjøres. Synlighetsmodifikatorer blir brukt for funksjonene og variablene vi har. Vi har for eksempel private variabler som vi ikke ønsker brukeren skal kunne aksessere utenom den gitte klassen, og vi har public metoder som kan bli brukt utenfor den gitt klassen (altså de kan bli tilkalt fra en annen klasse for å gjøre noe med det genererte objektet). Collection-rammeverket blir brukt for å bruke List<T> grensesnittet og ArrayList<T> klassen. De blir for eksempel brukt til å lagre alle fiskene i en liste slik at det ikke er en øvre begrensning i antall fisker, og fordi en liste er muterbar. Det blir brukt Exceptions slik at vi for eksempel vet om en fil eksisterer eller ikke når en fil skal bli brukt for lagring. De tre fiske klassene YellowFish, BlueFish og PinkFish arver metoder fra superklassen Fish blant annet fordi det gjør det enklere å skille de ulike objektene fra hverandre. Det gjør det mulig at hver type fisk har noen grunn likheter som f.eks metoder for bevegelse, men at de har hver sin individuelle metode for å hente poengsum og for å instansiere objektet med riktig bilde.

2)
Vi kunne ha lagt inn flere funksjoner for feilhåndtering via Exceptions, f.eks hvis en fisk prøver å bevege seg utenfor banen eller at man prøver å kaste ut snøret når snøret allerede er ute. Vi brukte heller ikke Observatør-Observert teknikken. Dette kunne vi implementert i håndteringen av Highscore lista. Dersom poengsummen man oppnådde etter endt runde er god nok for Highscore lista kunne et objekt som lytter på denne lista bli informert om endringen og oppdatert den lista som vises i grafikken i spillet.

3)

Model-View-Controller:

Koden er delt inn i ulike hoveddeler hvor kontroller-klassen håndterer hvordan input fra brukeren blir registrert og formidlet videre til andre deler av koden. FishMain er klassen som håndterer hvordan objekter, og spesielt noder (objektene som vises i spillet) interagerer med hverandre, og klasser som Fish og Boat som representerer de ulike nodene som objekter. Med tanke på Model-View-Controller prinsippet vil .FXML filen representere View biten, kontroller klassen representerer Controller biten, og FishMain og de resterende klassene representerer Model biten. Det er klassene som representerer Model biten som står bak all logikken i spillet. Hvert synlige objekt i spillet har en klasse med metoder som står bak dens mosjon, kalkulasjoner, og egenskaper generelt. Kontroller klassen fungerer som bindeleddet mellom Model og View delene, og gir Model klassene de nodene fra View delen som skal utføres beregninger på. Den kontroller alt som skjer i selve spillet. Altså at nodene vises skikkelig, at knappene fungerer osv.

4)

Bruk av Tester:

Vi lagde tester for noen av klassene våre, som for eksempel, Boat, og Fish. De ble laget for å sjekke at logikken og metodene i klassene fungerer som de skal. Begge klassene genererer objekter som beveger på seg, eller roterer gitt noen matematiske kalkulasjoner. Ettersom disse objektene, og dens metoder er sentrale deler av prosjektet, er det kritisk at de fungerer på riktig måte. Dermed lagde vi JUnit tester for å teste konstruktørene i disse klassene, samt logikken, som for eksempel bevegelse, som nevnt tidligere.

Vi har også fillagring i applikasjonen vår, og det er viktig at det også fungerer som det skal for å forhindre at feil data blir skrevet og lest. Ergo laget vi JUnit tester for dette også. Det sjekket om filen blir lagret riktig, og om verdiene som er i dokumentet er de riktige verdiene, altså verdiene fra “Highscore”-listen i spillet.

En del av funksjonaliteten i spillet ligger i ulike AnimationTimere slik at spillet oppdateres konstant og at alt beveger seg problemfritt. Disse oppdateres som sagt kontinuerlig, og vi fant det ikke hensiktsmessig å teste ulik funksjonalitet gitt av AnimationTimere ettersom vi da måtte ha konstruert store overflødige testscenarioer.

Klassene som Fish og Fishingrod har også metoder som bruker tilfeldige tall generert av Random() klassen, i logikken. Ettersom vi ikke kan forutse hva verdiene kommer til å være når vi kjører metodene, fant vi det vanskelig å teste dem gjennom JUnit tester. Heldigvis er appen en veldig visuell app slik at mye av testing vår underveis i utviklingen baserte seg på visuell testing.

Vi har i stedet aktivt brukt innkapsling og testing av gyldig tilstand integrert i applikasjonens kode. I tillegg ble det brukt mye testkode som gikk ut på print-statements ved ulike posisjoner i koden mens en ny funksjonalitet var under utvikling. Det er kanskje en mindre sivilisert måte å teste kode på, men i vårt tilfelle der ting skjer på løpende bånd med fisker som beveger seg osv. fant vi det nyttigere å ha enkle tester på mange posisjoner i koden enn større enhetstester færre steder.

5. Har dere møtt på noen utfordringer i løpet av prosjektet? Hva ville dere gjort annerledes en annen gang?

Vi møtte på noen utfordringer. Den originale planen vår var å lage en fysikk-simulator med en ragdoll man kunne leke med (spinne rundt, kaste den osv), men vi byttet idé etter at vi innså at det kanskje hadde blitt for vanskelig å gjøre det på så kort tid. Det vi fant mest vanskelig gjennom prosjektet var JUnit tester ettersom vi ikke klarte å finne situasjoner der de passet naturlig. Da hadde vi måttet skrevet om kode og kanskje lagt til kode kun for å få brukt JUnit tester, når koden allerede fungerte feilfritt. En bedre løsning for neste gang kunne kanskje vært å lage JUnit testene før vi skrev all kode, istedet for omvendt. Som nevnt i spørsmål 4 benyttet vi oss heller av mer primitive testmetoder slik som print-statements. Iblant “glitcher” en fisk seg i en av sidekantene. Det ser ut som den spreller mye frem og tilbake. Dette er forsøkt fikset, og det fungerer som oftest. Men ikke alltid.

Kilder:

Bilder brukt:

- https://clipartpng.com/?3023,blue-fish-png-clipart
- https://clipart-best.com/animals/fish/42
- http://www.stickpng.com/cat/animals/fish?page=1
- https://www.vectorstock.com/royalty-free-vector/cartoon-sea-bottom-background-for-game-design-vector-9961574
- https://clipartpng.com/?1392,wooden-boat-png-clip-art
- https://favpng.com/png_view/fishing-fisherman-fishing-boat-png/Vhvc1Zm0
- https://www.seekpng.com/ipng/u2y3q8e6o0y3w7e6_fishing-rod-cartoon-cartoon-fishing-rod-png/
-