Dokumentasjon for prosjekt i OOP

Del 1:

Beskrivelse av appen Her skal dere gi en kort beskrivelse av appen dere har laget og hva den gjør. Krav: 150-200 ord

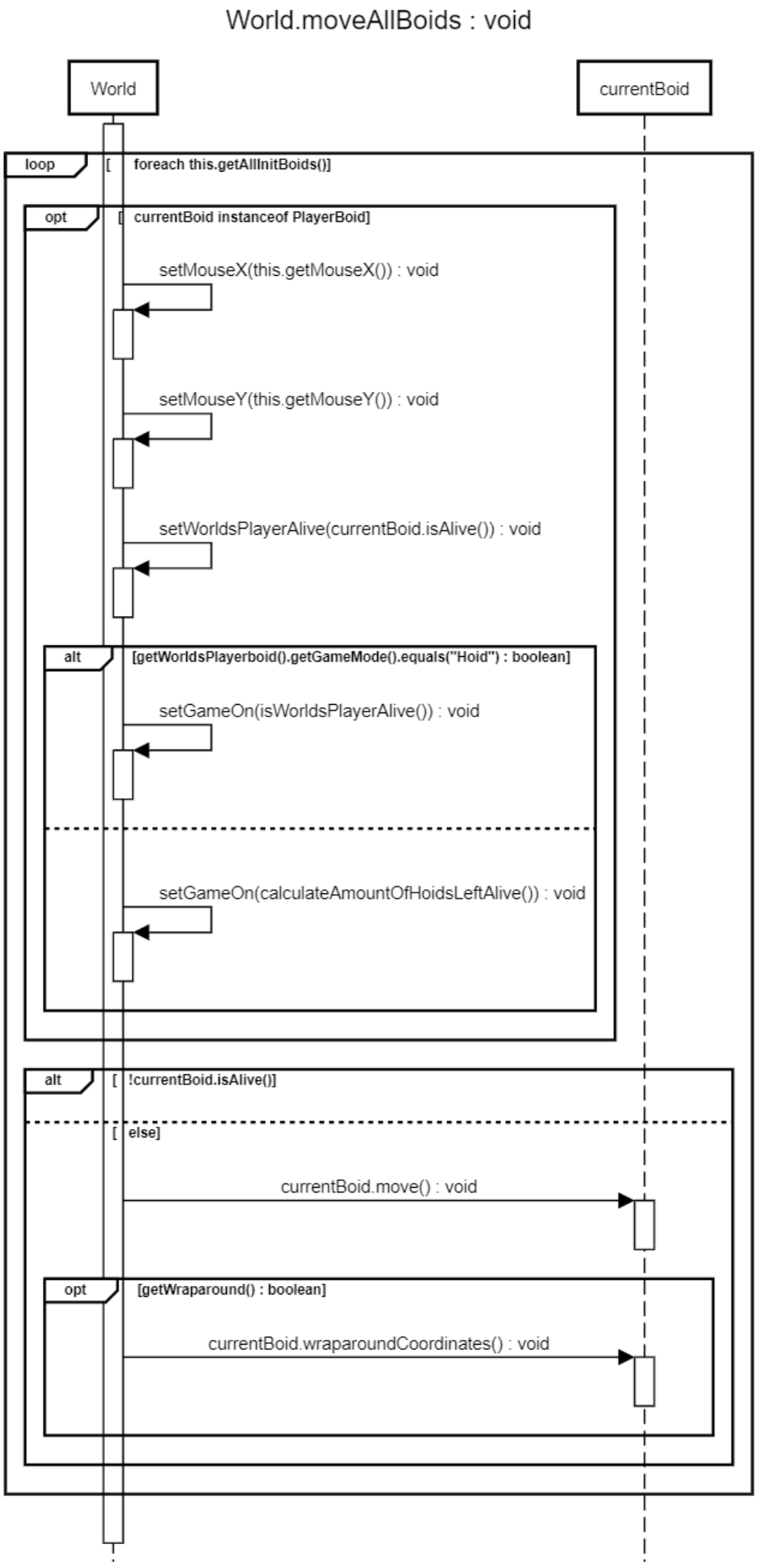
Applikasjonen jeg har laget heter «Boids the Game». Det er et spill som baserer seg på Craig Reynolds algoritmen Boids («Boids», 2022). Algoritmen prøver å etterligne flokkaktiv adferd hos dyr som fugler eller fisker. Målet for spillet kommer an på hvilken spillmodus som er valgt. Om man velger flokkfugl «Hoid» så er målet å overleve så lenge som mulig, mens rovfugl skal prøve å spise så mange hoids som mulig. Spillet er satt sammen av 10 klasser som utgjør Modellen. Den har en kontroller per scene og en standardkontroller som alle kontrollerne arver av.

Det finnes en abstrakt superklasse som kalles Boid (Bird-oid object). Boid har all felles funksjonalitet og felter samt to abstrakte metoder move() og isFriendlyBoid(). Klassene Hoid (herd-like boid), Poid (Predetor-like boid) og PlayerBoid utvides av Boid. Hoids følger boids algoritmen. Dermed vil de prøve å komme seg til lokalt flokksentrum, prøve å bevege seg i samme retning og hastighet som de andre og holde en minimumsavstand fra andre boids. I tillegg prøver den å bevege seg vekk fra andre rovfugler. Poids beveger seg etter nærmeste Hoid eller PlayerBoid med spillmodus Hoid. PlayerBoiden er klassen som spilleren har mest kontroll over. Den vil følge musen. World klassen representere den digitale 2d verdenen Boidsene lever i.

Del 2:

Diagram Her skal dere tegne ett diagram, som skal være med i dokumentasjonen som et bilde. Dere kan selv velge blant de fire diagramtypene i pensum; klassediagram, objektdiagram, objekttilstandsdiagram, sekvensdiagram. Diagrammet skal vise en interessant/viktig del av appen. Dere skal også kort begrunne valg av diagramtype. Dersom dere gjør noen antagelser som er viktige for å forstå diagrammet deres, skal disse også noteres ned her. Krav: Kun ett diagram.

I «Boids the Game» er det essensielt å forstå seg på hvordan alle boidsene blir beveget. Bevegelsen til alle boidsene blir påvirket av alle andre boids som den selv kan se. Klassen World har ansvaret for å bevege alle Boidsene. Funksjonen moveAllBoids() har ansvaret for å bevege hver enkelt boid som skal kjøres. Denne metoden blir påkalt av kontrolleren etter hvert frame.


World.moveAllBoids : void
World
currentBoid
loop
[foreach this.getAllInitBoids()]
opt
[currentBoid instanceof PlayerBoid]
setMouseX(this.getMouseX()) : void
setMouseY(this.getMouseY()) : void
setWorldsPlayerAlive(currentBoid.isAlive()) : void
alt
[getWorldsPlayerboid().getGameMode().equals("Hoid") : boolean]
setGameOn(isWorldsPlayerAlive()) : void
setGameOn(calculateAmountOfHoidsLeftAlive()) : void
alt
[!currentBoid.isAlive()]
[else]
currentBoid.move() : void
opt
[getWraparound() : boolean]
currentBoid.wraparoundCoordinates() : void

Del 3:

1. **Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For eksempel bruk av arv, interface, delegering osv.)**
   «Boids the Game» dekker store deler av pensum som innkapsling, for å forhindre andre klasser i å ha direkte kontroll til tilstander som de ikke skal ha, validering og unntakshåndtering, for å forsikre riktig kjøring av programmet og gyldigheten til tilstander.

   Bruk av arv gjennomsyrer prosjektet. Det er viktig å ikke repetere kode og dermed kan det være mer leselig og skaper mindre mulighet for feil ved å arve metoder som mange forskjellige klasser skal bruke. I «Boids the Game» er det to klasser som skal arves fra: Boid og Controller. Boid er en abstrakt klasse som gir både metoder, felter og krav om metoder. Dette er også viktig for å kunne behandle alle subklassene likt. Her kommer også polymorfisme med metoder som IsFriendlyBoid() som vil være forskjellige etter hvilken type Boid metoden blir kalt på. Her kunne man heller valgt å ha Boid som et interface. Men dette ville skapt mer repetisjon av kode. Ved hjelp av casting og av instanceof kan programmet aksessere særegen metoder som PlayerBoids setMouseX() og setMouseX()

   World klassen har eksempel på delegering i moveAllBoids(). Den henter det interne getAllInitBoids feltet som er Collection<Boid> og den delegerer arbeidet til hver enkelt boid med å bevege seg.
   Gjennom FileHandlerInteface er det bruk av interface, men det er kun til filehander klassen
   Programmet bruker flere steder Varargs f.eks Boid.validArgs(final int… args). Dette er for funksjoner hvor det er ukjent mengde argumenter, dette gjør slik at en ikke trenger å bruke masse method overloading og generelt minker mengden med overflødige linjer og tester.
   Programmet bruker også lambda utrykk for å sammenligne verdier i listen og deretter sortere dem, samt for å filtrere lister etter sannhetsverdier gitt av metoder.

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

   Programmet bruker ikke Set, Enums eller HashMaps. Kunne brukt det for å lagre brukerdata
   Hvis programmet skulle ekspanderes slik highscores ikke lagres lokalt, men på en database kunne det vært nyttig med en bruker klasse. Den kunne ha et internt felt som er et HashMap, hvor nøkkelen er tidspunktet en runde ble ferdig. Dette HashMap-et kunne sammenlignes med de beste rundene i forskjellige kategorier og om en av brukerens runder er bedre enn de andre kunne den sendes til databasen. Denne nye bruker klassen kunne ha brukernavn og passord hvor man kunne validere ved hjelp av RegEx utrykk. Den nåværende applikasjonen bruker ikke RegEx.

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 for å få full uttelling på dette spørsmålet. Det er mulig (og bra) å reflektere rundt svakheter i egen kode)**

   Model-View-Controller er et veldig viktig designmønster som «Boids the Game» prøve å følge. Controllerene i applikasjonen tar inn informasjon fra brukeren og manipulerer tilstander og påkaller handlinger i Model klassene ofte kommer bruker inputet fra

eventListeners som påkaller FXML funksjoner hvor controlleren tar og sender forespørselen til å endre tilstander til Modellen. Det er viktig at det ikke skal være noe særlig med tilstander og metoder, det er heller modellen som skal ha alle funksjonene. Controlleren kan deretter hente informasjon fra modellen og fortelle view-et hvordan den skal se ut og view-et vises for Brukeren. Det meste i programmet følger dette prinsippet, men det er en del av GameController.java som ikke står helt i tråd. Det lages en AnimationTimer som forteller når hver oppdatering skal skje. De forskjellige elementene burde være separert ettersom at det blir vanskelig å arbeide med. Ofte kan Model, Controller og View være kodet i forskjellige språk og det blir vanskelig for andre programmerere å lese koden eller vedlikeholde den hvis kode som ikke samhandler er gruppert sammen.

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)**
   Når det gjelder testing av programmet var det viktig å passe på at nesten alle linjene i koden gås gjennom i testene (code coverage). Det er essensielt å teste situasjoner som er realistiske, men det er også viktig å sjekke for scenarioer som ikke skal/kan oppstå. Dette er viktig for å kunne bruke koden senere til andre prosjekter og for at programmet skal kunne lettere vedlikeholdes og skaleres. Det er også viktig å teste det meste av kodebasen for å forsikre at brukerfeil ikke skal ødelegge programmet samt for å forsikre det ikke er logiske feil i koden. Enhetstestene i programmet burde sjekke om alle funksjonen fungerer som forventet. I testsettet laget til alle Klasser i modell delen av prosjektet er det vært fokus på å teste at konstruktøren lager objektet på riktig måte og hvordan den håndterer parametere som ikke er lovlige. Det er også viktig å teste interne felter for å sjekke om de blir mutert av funksjoner som kalles hvor de ikke skal muteres, samt er det viktig å teste oppførselen til klasser. Hvis en instans skal kaste unntak burde det testes for at det oppstår i forventede situasjoner.

5. **Har dere møtt på noen utfordringer i løpet av prosjektet? Hva ville dere gjort annerledes en annen gang?**
   Jeg ville startet med å lage flere controllere. Tidligere i utviklingsfasen hadde jeg kun en kontroller for alle de forskjellige scenene. Etter en stund ble denne controlleren vanskelig å håndtere. Det ble også feil med objekter som ble definert før de skulle eksistere som førte til flere NullpointsExceptions siden de ble definert som null. Hadde programmet vært mer i tråd med MVC på et tidligere stadium ville det vært tidsbesparende.

   Jeg burde også laget diagrammer for programmet i tidligere faser av utviklingen. Dette ville hjulpet med forståelse av hvordan kjøringen og strukturen av programmet skulle fungere.

Kilder:

Boids (2020, 21. februar) *Wikipedia* hentet https://en.wikipedia.org/wiki/Boids