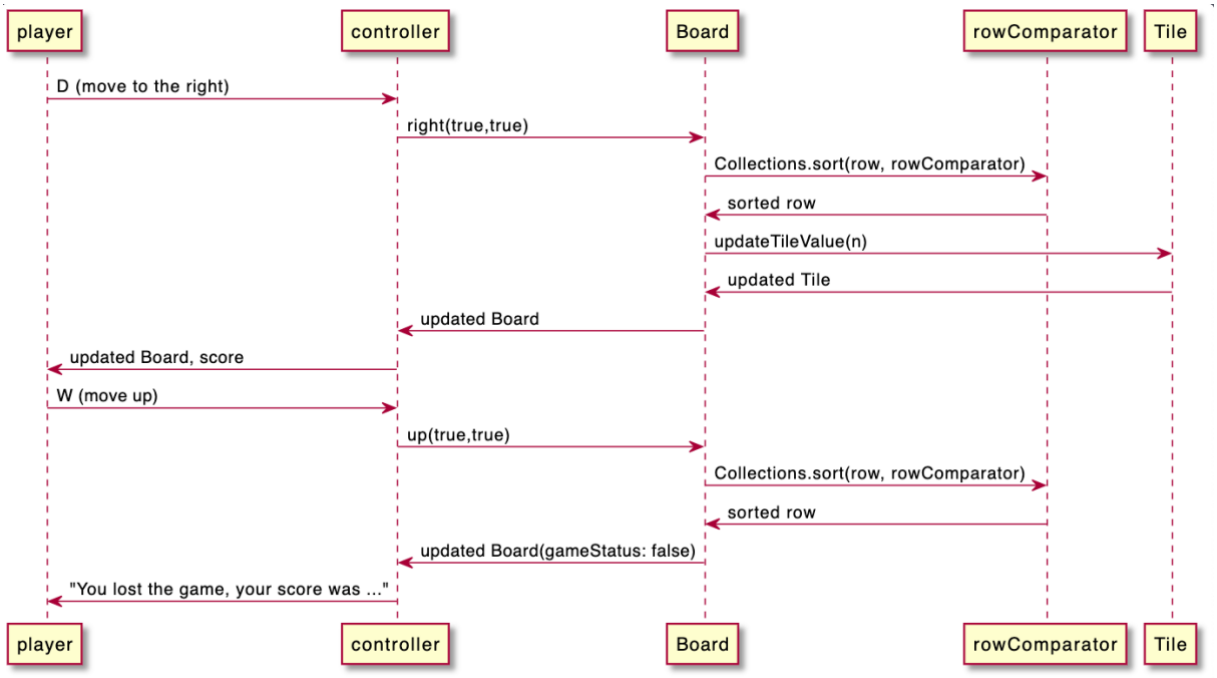
Dokumentasjon prosjekt java vår 2022

Del 1 – Beskrivelse av appen

Appen jeg har laget er en kopi av spillet 2048. Spillet går ut på at man har et 4x4 brett med brikker som har en tallverdi og farge. Poenget med spillet er å få så høy poengsum som mulig. Dette gjøres ved å slå sammen brikker med samme verdi. Spillet starter med en brikke tilfeldig plassert på brettet. Hvis spilleren trykker på W,A,S eller D vil alle brikkene flytte seg opp, venstre, ned eller til høyre. Når brikkene slåes slås sammen vil de få dobbelt så høy verdi. Spilleren vil da få poeng tilsvarende sammenslåingen. For eksempel vil to brikker med verdi 2 gi 4 poeng. Hvis spilleren gjør et trekk og det er bevegelse på brett vil det komme en ny brikke på brettet på et tilfeldig sted. Hvis ingen brikker endrer tilstand vil brettet se likt ut som forrige trekk. Hvis brettet er fullt og det ikke er noen trekk som forandrer tilstanden til spillet, er spillet over.

Del 2 – Sekvensdiagram

Jeg har valgt å lage et sekvensdiagram siden det kan illustrere hvordan brettet oppdaterer seg etter hvert som spilleren gjør trekk. Under er et sekvensdiagram hvor en spiller først flytter alt til høyre. Her går alt som det skal og han får et oppdatert brett tilbake. Neste trekk er brettet fullt. Det er visse ting som ikke blir gjennomført og han får beskjed om at spillet er over.



Del 3 – Spørsmål

Pensum som dekkes i appen:

- I appen finner vi flere klasser som jobber sammen ved at det er flere Tile objekter på et Board objekt. Board objektet flytter rundt på Tile objektet og delegerer oppgaver til dem.
- Det er en comparator klasse som sammenligner Tile objekter slik at brikkene sorteres riktig når et trekk blir utført. Dette gjøres for å lytte brikkene til høyre.
- Gjennom hele appen er det brukt innkapsling for å sikre at kun de feltene som skal være offentlige, er offentlige. Eksempler på dette er at tilstanden er privat, mens noen av metodene er public. Dette gjøres slik at man ikke kan endre tilstanden til et objektet på en ulovlig måte som vil ødelegge spillet. Måten man endrer tilstanden vil være gjennom settere.
- Appen har fillagringsfunksjonaliteter ved at den har mulighet for å lagre tilstanden til spillet opp til et brukernavn. Det er også mulighet for å hente spillet fra fil via brukernavnet. I tillegg blir det automatisk lagret ny highscore til en egen fil dersom den nye scoren er rekorden.
- Oppførselen til objektene er sikret ved at det er brukt validering for input. Det er også lagt inn unntakstilstander for å sikre til at hvis det kommer en ugyldig input, vil ikke appen stoppe å fungere. Dette skjer i forbindelse med lesing og skriving til fil.
- Det er brukt grensesnitt for å gjøre det lettere å lagre tilstanden til appen med andre filformater hvis det skulle trengs i fremtiden. Dette er ikke noe som blir brukt aktivt i denne applikasjonen, men muligheten er der.
- Java-streams blir brukt for å gå gjennom Tile- objekter og sette verdien til 0 når man restarter spillet. Dette var en litt raskere måte enn å lage en dobbel foreach løkke.
- Delegering blir for eksempel brukt når Board klassen får beskjed om å gjøre et trekk og ber Tile klassen om å endre verdien sin hvis det er relevant. Dette skjer når man setter sammen to brikker, eller man restarter spillet.
- Det er brukt FXML elementer i kontroller-klassen for å koble sammen brukergrensesnittet(Game.fxml) med det som skjer i appen.
- Observatør-observert teknikken blir brukt ved at Board- klassen har oversikt over de forskjellige Tile objektene som er på brettet. Når en Tile endrer verdi og farge vil Board objektet observere dette.
- Til slutt er det skrevet tester for å sjekke at appen og fillagring fungerer som forventet.

Deler av pensum som ikke er inkludert i appen er:

Arv er noe jeg kunne brukt for å lage flere typer brikker hvis jeg skulle utvide spillet. Her ville det vært et hierarki av brikker som arvet av hverandre. Den mest generelle brikker ville vært superklassen, og mer spesifikke brikker som arvet av den i tillegg til å ha egne funksjonaliteter som gjør brikken unik.

Model-View-Controller prinsippet

Koden forholder seg til model-view-controller-prinsippet ved at det er lite logikk i kontroller-klassen. Når spilleren trykker på en knapp vil kontrolleren få beskjed om at en knapp har blitt trykket og at model skal utføre en funksjon. Funksjonen blir utført og spillet er oppdatert. Deretter henter kontroller-klassen opp verdiene til appen og sender de videre til view.

Det er ingen kode som er direkte koblet til spillet som skjer i kontroller-klassen, men et eksempel på hvor det ble brukt logikk i kontroller-klassen er når spilleren skal lagre til fil eller hente et spill fra en fil. Hvis det allerede er lagret et spill på navnet, vil brukeren få valget om å overskrive spillet eller ikke.

Et annet eksempel på bruk av logikk i kontrolleren, er at det er validering. Hvis en spiller prøver å lagre eller hente opp et spill uten å skrive noe i input feltet vil man få en pop-up som forteller at man er nødt til å skrive inn noe. Det er validering for dette i FilSkriving klassen også, men man kommer ikke dit om man ikke skriver inn noe i brukernavnfeltet. Dette er gjort for at brukeren skal få umiddelbar tilbakemelding om at spillet ikke ble lagret.

Valg av tester

Når jeg skulle skrive tester for appen valgte jeg å skrive tester som sjekket alle hovedfunksjoner til spillet, i tillegg til fillagring. Det vil si at jeg ikke tester at en enkelt metode oppfører seg som forventet, men jeg tester heller at hvis det skal skje en endring i spillet fungerer dette som forventet.

Under testen av Tile og Board her jeg testet selve funksjonen til spillet. Dette vil si at spillet skal oppføre seg som tenkt. Et eksempel på dette kan for eksempel være at hvis en spiller trykker «D» skal alle brikker flyttes til høyre. Her tester jeg at brikker befinner seg på stedet de skal etter funksjons kallet, at brikker som skal settes sammen gjør det og at poeng blir oppdatert. I tillegg vil jeg også sjekke her at hvis brettet er fullt, er spillet over.

Under testene av fillagring, gikk testene ut på å sjekke at feilhåndtering fungerte som det skulle. Eksempler på dette kan være at en spiller prøver å lagre et spill uten å skrive noe eller hvis en fil ikke finnes. Jeg har også testet selve fillagringen, og hvordan den ble utført. Dette skjedde ved å sjekke at det som skal bli skrevet til filen, blir skrevet på riktig format. Dette

fikk jeg bekreftet da jeg hentet fra fil, hvor jeg fikk se om den hentet opp riktig spill. Til slutt testet jeg om overskriving av en lagring fungerte som forventet.

Refleksjon av hele prosjektet – Utfordringer

Gjennom prosjektet var det flere ting som var vanskelig. For det første var logikken i spillet vanskelig å implementere. Spesielt var det vanskelig å få brikkene til å flytte riktig. Når brikkene skal flytte seg til venstre, opp eller ned roterer jeg brettet og flytter brikkene til høyre. Deretter roterer jeg brettet tilbake. Dette var særlig vanskelig siden jeg måtte transponere matrisen(brettet).

En annen ting som var vanskelig, var skriving og lagring fra fil. Det var vanskelig å finne et format til fillagringen som gjorde at det var mulig å hente ut all informasjonen som trengtes når jeg skulle lese fra fil. I tillegg var det vanskelig å finne en måte å overskrive en fillagring. Jeg endte opp med å lese hele filen og sette det i et hashmap, og deretter bytte ut det gamle spillet med den nye. Etter det skriver jeg til filen på nytt.

Til neste gang hadde jeg satt av mer tid til å lese meg opp på ting jeg viste jeg kom til å få bruk for. Under prosjektet har jeg gjort mange ting som virket intuitivt i starten, men som har vist seg å ikke være det. Deretter har jeg måtte gjøre om på alt jeg har gjort.