1/26/2023

Obligatorisk Innlevering 1

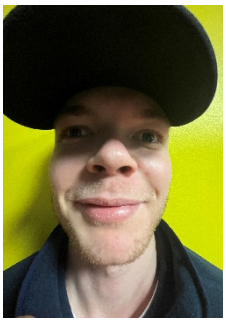
Gruppe 66

Gruppe 66
SIVERT SÆTER
MARIUS HELGÅS
GARD INDREKVAM
SIMON INDREKVAM

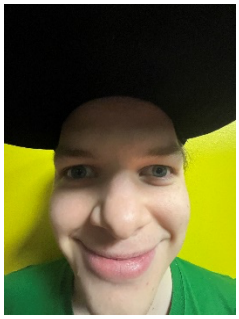
Gruppemedlemmer

Sivert Sæter

Marius Helgås



Gard Indrekvam



Simon Indrekvam

Link til bonobo server repository

https://ider-database.westeurope.cloudapp.azure.com/Bonobo.Git.Server/Repository/7f16f51c-86d2-468c-87b8-4f5ffd60020f/master/Tree/NyasteOblig1

Innholdsfortegnelse

Obligatorisk Innlevering 1 0

..... 0

Gruppemedlemmer 1

Innledning 2

Modell / Diagram 2

Brukstilfellemodell 3

Brukstilfelle stigespill 3

OOA - Domenemodell 4

Applikasjonen vår 4

Sekvensdiagram 5

Klassediagram 7

Innledning

Denne rapporten omhandler øvelse 1 i faget DAT109 – Systemutvikling. Øvelse 1 gikk ut på å modellere og programmere Stigespillet. Formålet med øvelsen er:

- Øve på å modellere et program fra idé til produkt.
- Øve på å forstå en kravspesifikasjon og lage en modell som samsvarer med kravene.
- Øve på å benytte objekt-orienterte prinsipp og utføringsmønstre for å lage en løsning.

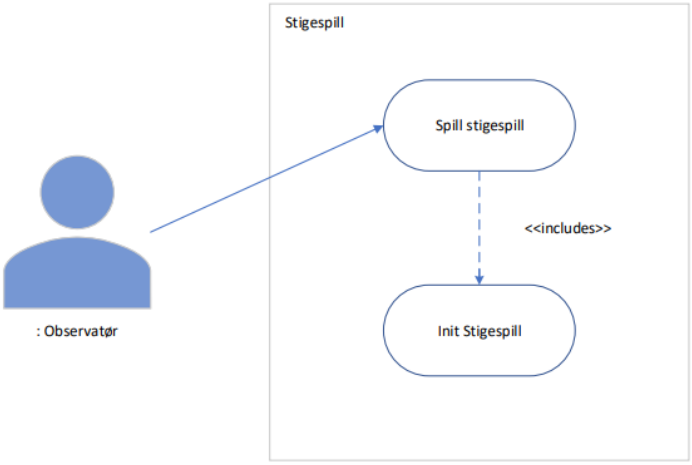
Videre forteller vi om hvordan vi jobbet med prosjektet og hvordan vi løste oppgaven.

Modell / Diagram

Det første vi startet med var å begynne å modellere løsningen og drøfte ulike måter vi kunne gjør dette på. Vi startet først å tegne brukstilfellediagram, hvor vi definerte brukstilfellet. Brukstilfellediagrammet viser hvordan produktet blir brukt hvis det er klart for et spill. Spillet går sin gang med at hver av spillerene kaster en terning og flytter antall øyne terningen viser. Spillet/runden er ferdig ved at den første som kommer til rutenummer 100. Man kan da spille en ny runde på nytt hvis man vil.

Brukstilfellemodell

Brukstilfellemodell

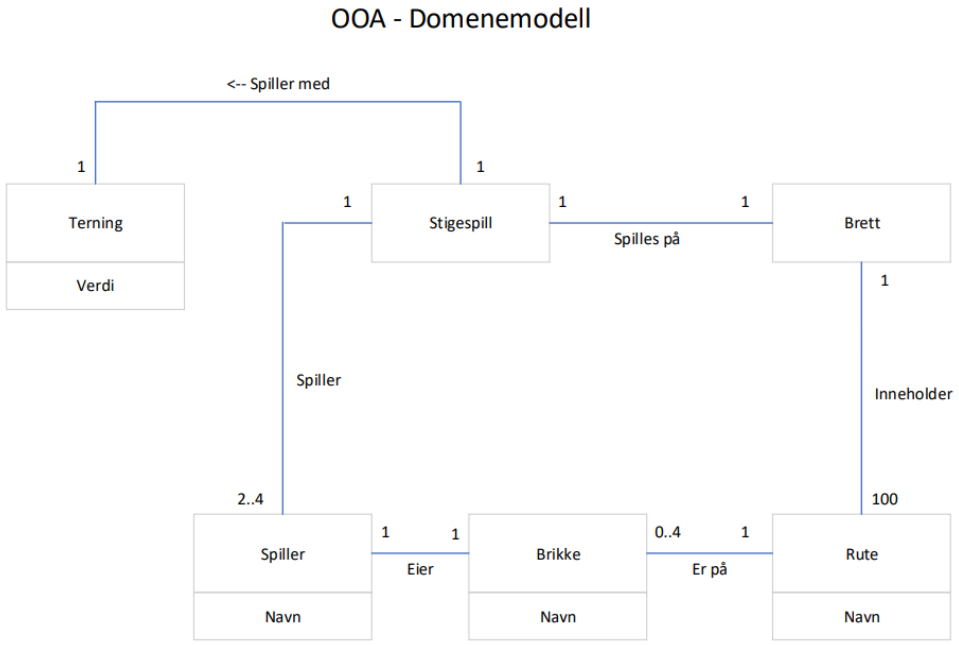


Brukstilfelle stigespill

1. Spillet initialiseres
2. Spillet starter
3. Spillerene gjør hvert sitt trekk.
    a. Trill terning.
    b. Flytt terningen antall øyner den viser.
4. Gjenta fra punkt 3 og helt til den første brikken når rute nummer 100.
5.

En observatøren har lyst å spille stigespillet. Spillet initialiseres først, og så spillet starter. Under ser du brukstilfellediagrammet vårt.

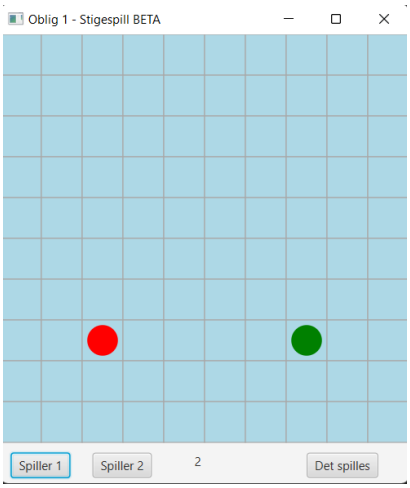
OOA - Domenemodell

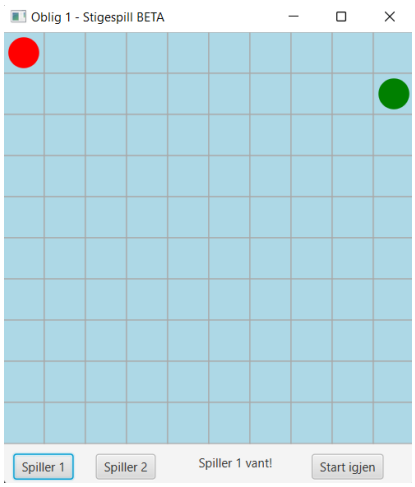


Deretter lagde vi domenemodell. Domenemodellen er ment for å gi et overblikk over applikasjonen. Den viser hvilke objekter som er nødvendig for at det skal være et fungerende spill. Modellen forklarer hva stigespillet inneholder og forholdet mellom disse og hvordan de opererer på hverandre. I denne modellen har spillet ett brett, med hundre ruter. Spillet kan ha to til 4 spillere(brikker). Spillet må også ha én terning for å fungere. Og en rute kan ha null til fire brikker. Under ser du domenediagrammet vårt

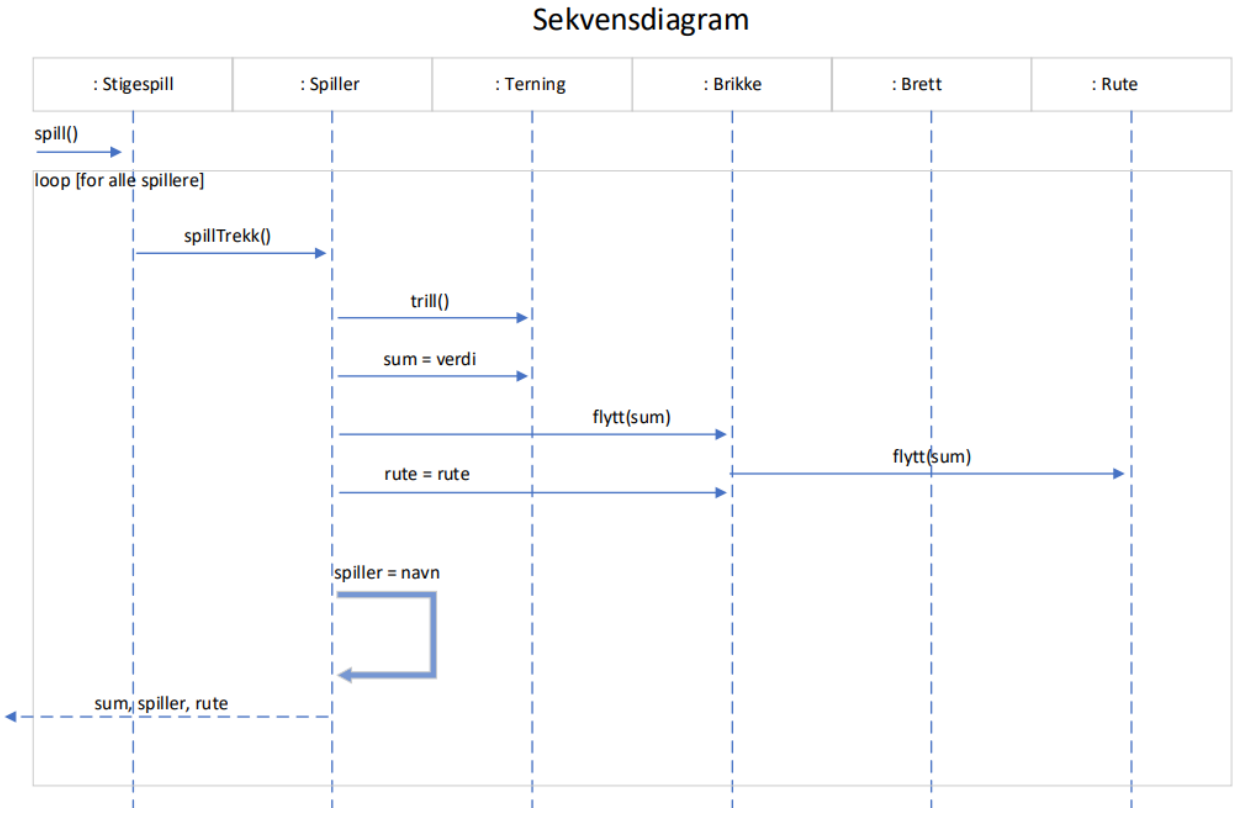
Applikasjonen vår

Når vi planla spillet valgte vi å bruke JavaFX som rammeverk og for å få et fint brukergrensesnitt til spillet vårt. Dette var litt nytt for oss, men vi prøvde det allikevel. Vi kunne også ha programmert spillet til å foregå i konsollvinduet, med Scanner inputs, men valgte å prøve noe nytt. Løsningen vår er kanskje litt kronglete med tanke på at vi bare har laget spillet med to brikker og to knapper. Dette kan vi endre på etterhvert men tenkte at vi skulle få et fungerende spill i første steg. Når applikasjonen blir kjørt, vises brettet og man må trykke på knappen nede til høyre hvor det står «Start spillet». Når knappen er trykket på endres teksten til, «Det spilles». Spillerene må trykke på hver sin knapp for å trille terning mens brikken automatisk flyttes til riktig rute. Når en av brikkene har nådd rute 100, kommer det opp melding om at spiller to vant. Man får da muligheten til å trykke på «Start igjen»-knappen. Gjennom spillet får brukerene hele tiden vite hva terningen viser. Det vises til høyre for de to spiller-knappene. Når en spiller får 6, får han kaste på nytt og flytter antall øyne terningen viser. Bildene under viser hvordan spillet ser ut under og etter en spiller har vunnet.





Sekvensdiagram



Sekvensdiagram viser spillets gang og metodene som blir kalt for å spille stigespillet. Spillet skjer i en loop, helt til en spiller når rute nummer 100. Frem til da blir trill()-metoden brukt på hver spiller, en etter en.

```
static void flyttSpiller1() {
    for(int i = 0; i < Terning.rand; i++) {
        if(posisjonSirkel1 % 2 == 1) {
            spiller1XPosisjon +=40;
        }
        if(posisjonSirkel1 % 2 == 0) {
            spiller1XPosisjon -=40;
        }
        if(spiller1XPosisjon > 380) {
            spiller1YPosisjon -=40;
            spiller1XPosisjon -=40;
            posisjonSirkel1++;
        }
        if(spiller1XPosisjon < 20) {
            spiller1YPosisjon -= 40;
            spiller1XPosisjon += 40;
            posisjonSirkel1++;
        }

        if(spiller1XPosisjon <= 20 && spiller1YPosisjon <= 20) {
            spiller1XPosisjon = 20;
            spiller1YPosisjon = 20;
            Brett.randResult.setText("Spiller 1 vant!");
            Brett.gameButton.setText("Start igjen");
        }

    }
}
```

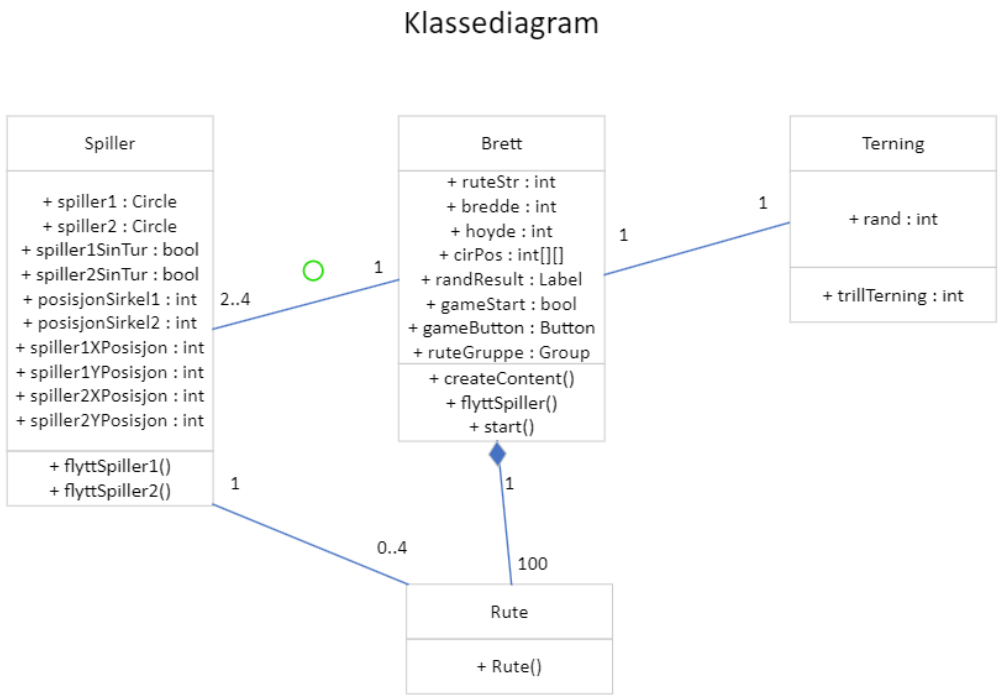
Her bruker vi en for-loop og et par if-setninger for å flytte spilleren en plass om gangen. For-loopen kjører like mange ganger som det øynene på terningen viser.

```
14 usages
public static int trillTerning(){
    rand = (int) (Math.random() * 6 +1) ;
    return rand;
}
```

Her bruker vi vi bare vanlig kode for å generere et tall mellom 1 og 6.

Klassediagram

Her er klassediagrammet vårt.



Enhetstester

```
@Test
public void tesTrillTerning1() {
    if (Terning.trillTerning() == 1) {
        assertEquals(Terning.trillTerning(), actual: 1);
    }
}
no usages
@Test
public void tesTrillTerning2() {
    if (Terning.trillTerning() == 2) {
        assertEquals(Terning.trillTerning(), actual: 2);
    }
}
no usages
@Test
public void tesTrillTerning3() {
    if (Terning.trillTerning() == 3) {
        assertEquals(Terning.trillTerning(), actual: 3);
    }
}
no usages
@Test
public void tesTrillTerning4() {
    if (Terning.trillTerning() == 4) {
        assertEquals(Terning.trillTerning(), actual: 4);
    }
}
no usages
@Test
public void tesTrillTerning5() {
    if (Terning.trillTerning() == 5) {
        assertEquals(Terning.trillTerning(), actual: 5);
    }
}
no usages
@Test
public void tesTrillTerning6() {
    if (Terning.trillTerning() == 6) {
        assertEquals(Terning.trillTerning(), actual: 6);
    }
```

Bilde over viser testen vi har laget til å sjekke om terningen fungerer.

Konklusjon/sluttresultat

Konklusjonen av dette kodeprosjektet om stigespill er at det har vært en vellykket implementering av et fungerende stigespill med to spillere. Det har vert noen utfordringer underveis, som å implementere at spillerene skal flytte seg i form av koordinater. Vi skjønner at koden kunne blitt enda mer optimalisert, men det var denne løsningen vi valgte å gå for denne gangen.