Mappeoppgave: Boardgame

IDATx2003 Programmering 2, våren 2025

v3.0 - 24. mars 2025

Sammendrag

Mappeoppgave for emnet IDATx2003 Programmering 2 for studieprogrammet Bachelor i ingeniørfag, data (dataingeniør) ved NTNU Trondheim, Gjøvik og Ålesund.

Revisjonshistorikk

Versjon	Dato	Forfatter	Beskrivelse
1.0	25. januar 2025	Arne Styve m.fl.[*]	Introduksjon til mappen. Mappe del 1
2.0	25. februar 2025	Arne Styve m.fl. [*]	Mappe del 2 lagt til
2.1	27. februar 2025	Arne Styve m.fl. [*]	Mappe del 2: Presisering om krav til filbehandling (kap 4.3).
3.0	24. mars 2025	Arne Styve m.fl. [*]	Mappe del 3 lagt til

([*] Arne Styve, Ivar Farup, Majid Rouhani, Atle Olsø)

NTNU

Kunnskap for en bedre verden

Innhold

Figurer

Kodelister

1 Om mappevurdering i IDATx2003 Programmering 2

I dette emnet er det ingen skriftlig eller muntlig eksamen. I stedenfor benytter vi mappe som vurderingsform. Mappen skal løses **i grupper på 2 deltagere**. Dette fordi vi ønsker dere skal få erfaring med **par-programmering**.

Mappen leveres ut ca. uke 5 og går parallelt med øvrige øvingsaktiviteter ut semesteret. Detaljert prosjektbeskrivelse blir publisert i 3 deler:

- Del 1 - ca. uke 5: Fokuserer på design av klasser, kodestil og dokumentasjon, samt versjonskontroll og enhetstesting.
- Del 2 - ca. uke 9: Fokuserer på filhåndtering og designmønstre.
- Del 3 - ca. uke 13: Fokuserer på ferdigstillelse av applikasjon med et grafisk brukergrensesnitt, og med komplette enhetstester. I tillegg skal dere utvide løsningen med egne forslag/ideer.

Hver ny del bygger på foregående del. Når ny del publiseres, vil dere få muntlig tilbakemelding på arbeidet som er gjort så langt, med mulighet til å justere/endre/forbedre løsningen. Dere vil få i alt 3 slike tilbakemeldinger, der den siste tilbakemeldingen gis i uke 17.

Endelig fungerende løsning leveres i GitHub/GitLab og som ZIP-fil sammen med en rapport i Inspera innen fristen (se Blackboard for detaljer om innlevering).

Vekting mellom rapport og prosjekt mot endelig karakter:

- Rapport - 30%
- Prosjekt - 70%

Kilder:

- Hva Forskriften sier om mappe som vurderingsform:
  `https://lovdata.no/dokument/SF/forskrift/2015-12-08-1449/KAPITTEL_5#%C2%A75-11`

(a) Monopol **(b)** Stigespill **(c)** Ticket to Ride **(d)** Den forsvunne Diamant

Figur 1: Eksempler på brettspill

2 Problembeskrivelse - Brettspill/Boardgame

Det finnes en mengde ulike brettspill på markedet i dag, både fysiske og digitale. Fellesnevnere for svært mange av disse brettspillene er at

- spillet består av en samling **felt** (engelsk:**«tile»**) som gjerne er **lenket sammen**; med andre ord, når man spiller så beveger man seg fra ett felt til et annet
- man bruker en eller annen tilfeldig tallgenerator for å bestemme hvor mange felt man skal flytte (f.eks. **terninger** – engelsk: **«dice»**)
- spillet kan spilles av en eller flere **spillere**
- ofte når en spiller lander på et felt, kan type felt bestemme videre forløp («stå over et kast», «gå direkte til fengsel», «gå opp stigen» osv.)

I denne oppgaven skal dere lage et brettspill med grafisk brukergrensesnitt. Oppgaven gis i to nivåer; nivå 1 og nivå 2. Dere kan velge om dere bare gjør nivå 1 eller starter med nivå 1 og deretter utvider til nivå 2. Nivå 1 kan sees på som et **minimumskrav** (en jevnt over god prestasjon her vil typisk være en «C»-besvarelse), mens nivå 2 gir dere utfordringer som kan løfte besvarelsen til en høyere karakter.

Videre i dette kapittelet følger kravspesifikasjonen til både nivå 1 og nivå 2. Dette er funksjonalitet som er forventet **etter at dere har fullført Del 3 av denne mappen**, så gjør prosjektet ved å følge de tre delene **Del 1, Del 2 og Del 3** videre i dette dokumentet (tilsvarende mappen i *IDATx1003 Programmering 1*).

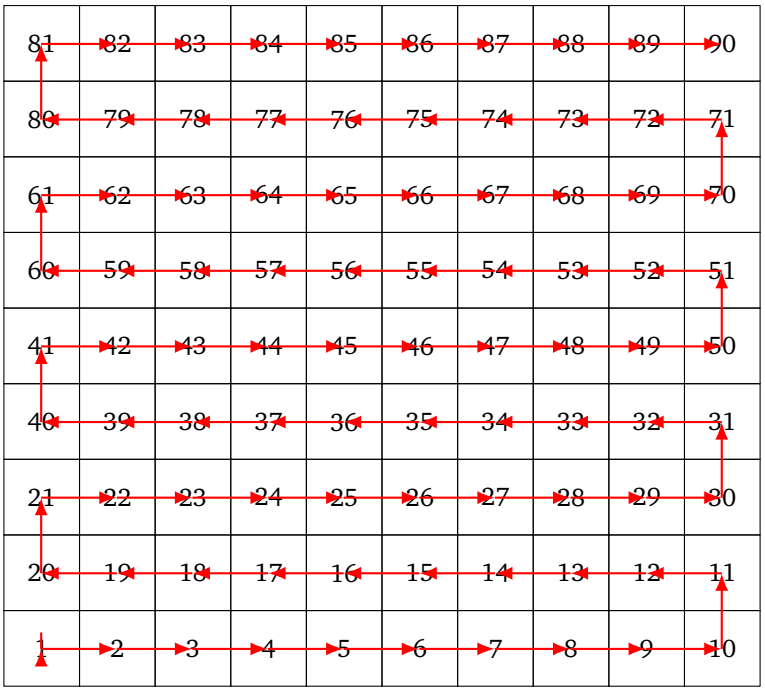
2.1 Nivå 1: Stigespill med noe ekstra (MVP)

Det tradisjonelle stigespillet består av et **spillebrett** med 9 rader med 10 **felt** i hver rad (10 kolonner). Feltene er nummerert fortløpende fra 1 til 90, i «slangemønster» som vist i figur 2.

Spillet starter ved at ingen spillbrikker er på brettet. Yngste spiller begynner ved å kaste to terninger og flytte spillbrikken sin det antall øyne terningene viser fra rute nr. 1 og videre. Den første feltrekken følges helt ut til høyre, så flytter man opp i neste rekke og følger feltene helt ut til venstre. Slik fortsetter man til toppen er nådd.

2.1.1 Stiger

Noen av feltene er koblet sammen med **stiger**; en stige kan enten **flytte en spiller opp** eller **flytte en spiller ned** på brettet. Felt med stige som flytter spiller opp er som regel **farget mørk grønn**. Feltet som stigen leder til er da gjerne **lys grønn** på farge. Felt med stige som tar spilleren nedover på brettet er gjerne **farget rødt** og feltet stigen ender på er gjerne farget **lyserødt/oransje** (se figur 1b).



Figur 2: Spillebrett for Stigespillet

2.1.2 Noe ekstra

For å gjøre spillet litt mer spennende, skal det i tillegg inneholde enkelt felt med effekten

- «rykk tilbake til start»
- «stå over ett kast før spilleren kan gå videre»
- eller lignende..

2.1.3 Kravspesifikasjon – funksjonelle krav

Følgende funksjonelle krav gjelder:

- En bruker skal kunne velge et spill blant et på forhånd definerte varianter av stigespill (f.eks. ved å lese fra fil). Med «varianter» menes f.eks. layout (antall felt) og hvor stigene er plassert).
- Bruker velger så hvor mange spillere som skal delta i spillet (maks fem).
- Hver spiller velger deretter en «brikke» for sin spiller (fra et utvalg av ulike forhåndsdefinerte brikker/figurer/ikoner) og gir sin brikke/spiller et navn.
- Spillet starter ved at det trilles to terninger for hver spiller, og spilleren sin brikke flyttes tilsvarende.
- Avhengig av hvilken type felt spilleren lander på, utføres respektiv handling (stige opp, stige ned, osv.).
- Når første spiller når siste rute (rute nr. 90) er spillet over, og vinneren kåres.

2.2 Nivå 2: Flere ulike brettspill

Med utgangspunkt i **felt** som er koblet sammen i en **struktur**, kan man også lage andre brettspill enn stigespillet. Slike brettspill kan deles i to kategorier:

1. Spill der spilleren ikke kan velge hvilken retning hen skal bevege seg (stigespill, monopol osv), og det finnes kun en vei å bevege seg fra ett felt til det neste.

2. Spill der spilleren har flere alternative veier ut fra feltet hen står på (f.eks. «Ludo», «Den Forsvunne diamant»), der spiller også kan gå tilbake samme vei hen kom («Den Forsvunne diamant»)

For dette nivået skal dere bygge videre på nivå 1, og utvide mulighetene til brettspillet. Dere velger selv om dere vil f.eks. gå for en variant av «Monopol», «Den forsvunne diamant», eller om dere vil lage et helt nytt spill.

2.3 Avgrensninger

For begge nivåer gjelder følgende avgrensninger av oppgaven:

- Spillet skal spilles av alle spillere på samme datamaskin (ingen nettverksbasert spill).
- Spillet skal ha et **grafisk brukergrensesnitt** utviklet med biblioteket **JavaFX** uten bruk av FXML.

2.4 Bruk av KI verktøy

Det er tillatt å anvende KI verktøy som støtte for å løse mappen, så lenge

- KI-verktøyet benyttes som sparringpartner/rådgiver og oppslagsverk, og ikke for å generere kode for direkte bruk
- eventuell kode som foreslås av KI-verktøyet gjennomgås kritisk og eventuelt bearbeides før dere tar koden i bruk i prosjektet
- dere kommenterer i koden der dere har fått hjelp av KI-verktøy
- bruken av KI-verktøy dokumenteres i rapporten (hvilke verktøy ble benyttet til hva, og gjerne med eksempler på prompts- og svar)

Eksempler på KI-verktøy:

GitHub Copilot: Bruk chatbot'en, og ikke «code completion». Chatboten er inkludert i plugin for IntelliJ og VSCode. GitHub Copilot er mye bedre enn ChatGPT i forhold til programmeringskode. Dere har gratis tilgang til GitHub CoPilot dersom dere søker om «educational license» hos GitHub (https://education.github.com/pack)

Khan Academy Tutor Me: En variant av ChatGPT som oppfører seg mer som en veileder (tutor). M.a.o. så gir Tutor Me deg hint og tips istedenfor ferdig løsning.
Se: https://chat.openai.com/g/g-hRCqiqVlM-tutor-me

2.5 Øvrige krav til prosjektet

Følgende krav og betingelser gjelder for hele mappen:

2.5.1 Kodekvalitet

Det forventes høy kodekvalitet i prosjektet. Bruk verktøy som **CheckStyle** (med Google sine regler) og **SonarLint**. Begge verktøy finnes som plugins til IntelliJ og VSCode med flere (se vår emnewiki for detaljer).

2.5.2 Enhetstesting

Dere skal lage enhetstester for den delen av koden som er forretnings-kritisk (forretnings-logikken), altså for den koden som er viktigst for å oppfylle sentrale krav. Feil her vil få store negative konsekvenser for programmet. Negativ testing er også viktig. Det er ikke fornuftig å lage enhetstester til klasser som utgjør brukergrensesnittet, med mindre det finnes metoder som kan testes uten at bruker er involvert.

Unngå også å lage enhetstester som leser- eller skriver til fil. Slike tester tar unødvendig lang tid å kjøre siden data skal til- og fra en fysisk disk. I tillegg er det umulig å ha kontroll på testen. Hva om en fil blir ødelagt, eller testen din plutselig ikke har tilgang til filen osv.

2.5.3 Unntakshåndtering

Uønskede hendelser og tilstander som forstyrrer normal flyt skal håndteres på en god måte. Håndteringen skal være rimelig og balansert (ikke for mye, ikke for lite) med det formål å gjøre koden mer robust. Vi gjør oppmerksom på at unntakshåndtering ikke er eksplisitt angitt i oppgavene under. Dere må selv avgjøre når og hvordan unntak skal implementeres. En viktig sammenheng der unntak bør brukes er validering av parametre til metoder.

2.5.4 Versjonskontroll

Prosjektet skal legges under versjonskontroll. Krav til versjonskontroll er ytterligere beskrevet i del 1.

2.5.5 Java-biblioteker og Maven-plugins

Følgende versjoner av Java JDK, biblioteker og Maven plugins skal benyttes:

- Java: Siste tilgjengelige LTS (Long Time Support) versjon **skal** benyttes: versjon 21
- Bibliotek (<dependencies> i pom.xml)
  - JUnit: *org.junit.jupiter:junit-jupiter-engine:5.11.4*
  - JavaFX (GUI): *org.openjfx:javafx-controls:23.0.1*
- MavenPlugins (<build><plugins> i pom.xml)
  - *org.apache.maven.plugins:maven-compiler-plugin:3.13.0*
  - *org.apache.maven.plugins:maven-surefire-plugin:3.2.5*
  - *org.openjfx:javafx-maven-plugin:0.0.8*

For andre biblioteker dere måtte velge/trenge, husk å **alltid** spesifisere eksakt versjon, **aldri** bruke RELEASE eller LATEST!

Kodeliste 1: Eksempel på en pom.xml-fil med riktige versjoner

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- TODO: Replace with your own groupID and artifactID -->
    <groupId>edu.ntnu.iir.bidata</groupId>
    <artifactId>IDATA2003-Mappe3-Boardgame</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <!-- Per Januar 2025 the LTS version is 21 -->
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>5.11.4</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>23.0.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- The Java compiler -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.13.0</version>
            </plugin>

            <!-- The unit-test engine Surefire -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.5.2</version>
            </plugin>

            <!-- Plugin to execute JavaFX applications from Maven -->
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.8</version>
                <configuration>
                    <!-- TODO: Replace with your own main-class -->
                    <mainClass>org.openjfx.App</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2.5.6 Prosjektrapport

Det skal skrives en rapport for prosjektet. I rapporten skal dere forklare hvordan løsningen er bygget opp, hvilke valg som er tatt underveis, hvordan dere har anvendt designprinsipper osv.

Rapporten skal være på **ca. 2500–3000 ord**. I tillegg kommer figurer og eventuelt små kodesnutter for å vise hvordan utvalgte problemer er løst. Språk er enten norsk eller engelsk.

Det er lurt å starte på rapporten allerede fra starten. Dokumentmal finner dere på Blackboard.

Del 1 av 3

Prosjektoppsett og grunnleggende klasser.

3 Mappe del 1 av 3

I denne første delen av prosjektet, skal vi fokusere å følgende:

1. Sette opp prosjektet som et **Maven prosjekt**
2. Legge prosjektet til versjonskontroll lokalt
3. Koble lokalt repository til sentralt repository i GitHub/GitLab
4. Starte på grunnleggende klasser i spillet (Spill, Spiller, Felt, Terninger m.m.), m.a.o. **entitets-** og **forretningslogikk**-klassene.

Punkt 1 - 3 i listen over er det lurt at **én av dere** gjør på sin datamaskin. Når prosjektet er pushet til GitHub/GitLab, kan den andre på gruppa **klone** prosjektet til sin datamaskin før videre arbeid.

3.1 Opprette Java-prosjekt i henhold til Maven

Opprett et nytt Java-prosjekt basert på Maven i en passende mappe på harddisken. Det kan være en fordel å ikke plassere prosjektet i en mappe som er synkronisert med OneDrive, Dropbox, iCloud eller lignende tjenester.

Se Kodeliste 1 i kapittel 2.5.5 for forslag til **pom.xml**-fil.

3.2 Legg prosjektet til kildekodekontroll (Git og GitHub/GitLab)

Når man skal starte med et nytt prosjekt som skal benytte GitHub/GitLab, så er en mulig fremgangsmåte som følger:

1. Opprett prosjektet på GitHub/GitLab. Dersom dere allerede har et prosjekt opprettet på GitHub (f.eks. via GitHub Classroom (Ålesund)), så hopper dere over dette trinnet.
2. En av gruppens medlemmer **kloner** prosjektet til et passende sted på harddisken på sin datamaskin.
3. Vedkommende oppretter et Maven-prosjekt i roten på det klonede prosjektet. Kontroller at prosjektfilen **pom.xml** ligger på rot-nivå i prosjekt-mappen.
4. Prosjektet **committes** og **pushes** før øvrig(e) medlemmer kloner prosjektet til sin datamskin
5. Sjekk at alt er OK ved å skrive ***git status***
6. Når dere utfør en ***git push*** første gang, er det mulig dere blir bedt om å konfigurere navn og e-post. Følg instruksjonene som gis.

3.3 Brukergrensesnitt

Start på planleggingen av det grafiske brukergrensesnittet. Bruk penn og papir eller verktøy som **Figma** (https://www.figma.com/) eller **Balsamiq Desktop** (https://balsamiq.com/wireframes/desktop/). Velger du **Balsamiq** må du laste ned desktop-applikasjonen (finnes for Mac og Windows, men ikke for Linux dessverre).

Bruk følgende lisens (gyldig til 31 desember 2025):

```
IDATAvarious2024|QhlFeJxzCncxiQ+p8XRxDHFUKEssyswvLVYwMjAyqTE0NzMxM7Q0MgABAOt9CsU=
```

3.4 Rapporten

Start på rapporten tidlig. Med utgangspunkt i informasjonen du allerede nå har om prosjektet, kan du starte på prosjektrapporten ved å fylle inn følgende avsnitt:

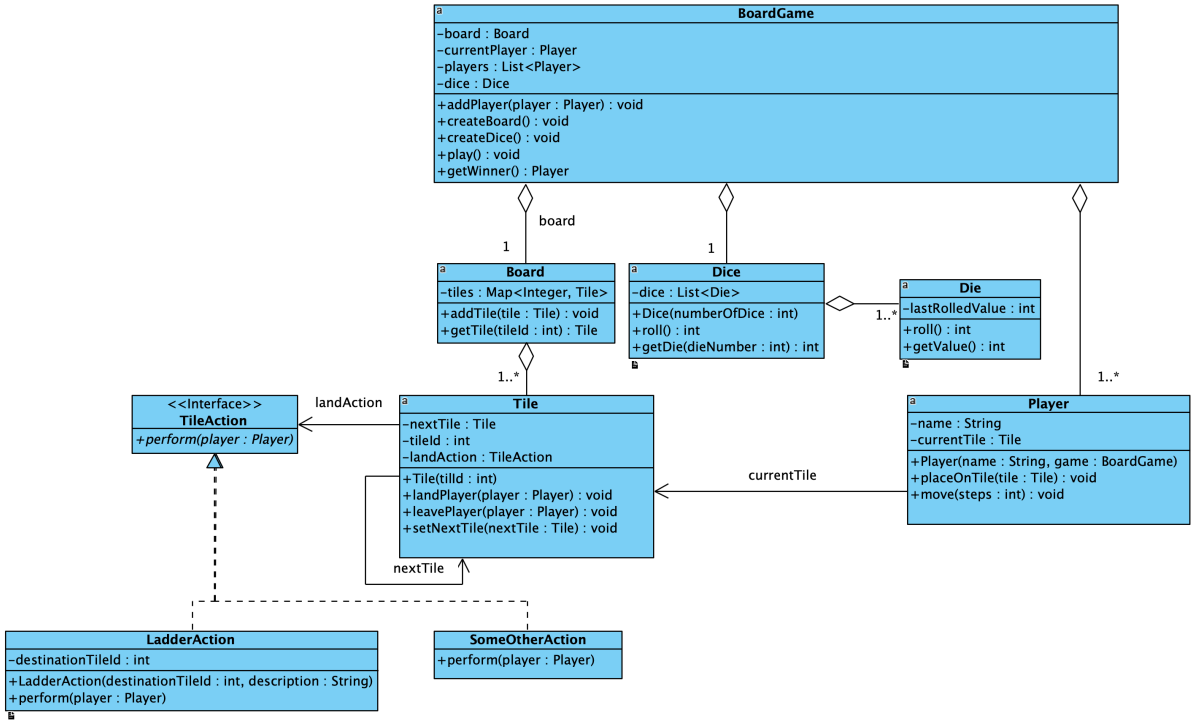
- innledning/problembeskrivelse (med use case-diagram)
- avgrensninger
- begreper
- teori

- metode (prosess og verktøy)

I tillegg kan du så smått starte på **resultat** med de første skissene av klassediagram.

Dere kan selv velge om dere ønsker å skrive på norsk eller engelsk.

3.5 Grunnleggende entitet-klasser og spillogikk



Figur 3: Klassediagram som viser forslag til klasser

Ved å analysere kravspesifikasjonen i kapittel 2 i henhold til **substantiv-metoden**, kan vi identifisere følgende **kandidater til klasser**:

- Spill (engelsk: Game)
- Terning (eng: Die) og terninger (eng: Dice)
- Spillbrett (eng: Board)
- Spiller (eng: Player)
- Felt (eng: Tile)
- Aksjon(som f.eks. gå opp stige/gå ned stige) (eng: Action)

Disse begrepene utgjør også som regel **ordboken/domenebegrepene** (eng: dictionary) til løsningen. I tillegg til å identifisere de ulike kandidatene til klasser, kan vi også si noe om **relasjonene** mellom klassene; et spillbrett (Board) **består av** en eller flere felt (Tile). et spill består av en eller flere spillere (Player) osv.

Figur 3 viser **forslag** til klasser for spill-logikk og entitetsklasser og **relasjonene** mellom klassene. Du står fritt til å velge andre klasser, så lenge du opprettholder designprinsippene, og da spesielt kohesjon (eng: cohesion). NB! Klassediagrammet er på ingen måte komplett; her mangler det mange metoder og potensielt også felt/attributter.

Under følger en beskrivelse av rolle/ansvar til hver av klassene.

3.5.1 BoardGame

Ansvarlig for selve spillet, herunder:

- sette opp brettet (Board)
- sette opp terninger
- registrere spillerne som skal delta
- spille selve spillet ved å iterere over spillerne og la hver enkelt trille terninger og flytte på brettet respektivt; når første spiller har nådd siste felt (mål), avsluttes spillet og vinner kåres.

3.5.2 Board

Representerer selve spillbrettet bestående av en samling **felt (engelsk: Tile)**. Spillbrettet holder også en totaloversikt over alle felt. Hvert felt har en unik ID som spillbrettet kan bruke for å holde oversikt over samtlige felt i spillet. Dette blir viktig i forbindelse med stige-funksjonaliteten: spiller skal hoppe fra et felt til et annet felt.

3.5.3 Tile

Representerer **ett felt** på spillbrettet. I denne mappen har vi sagt at felt skal henge etter hverandre. Derfor har et felt alltid en referanse til «neste felt». Dersom det ikke finnes et «neste felt» er feltet det siste feltet i spillet (altså mål).

Et felt kan ha tilknyttet en «aksjon/handling» (engelsk: Action) som blir utført på den spilleren som lander på feltet.

3.5.4 TileAction

Dette er et ***interface*** som spesifiserer et standard grensesnitt for ulike typer aksjoner (engelsk: Action) som skal kunne utføres på en spiller som lander på et felt. Konkrete hendelser må defineres, og må **implementere** dette grensesnittet (som f.eks. «LadderAction» og «SomeOtherAction». («SomeOtherAction» er tatt med her for å illustrere at man kan ha flere ulike typer actions.)

For eksempel kan et «LadderAction» objekt være tilknyttet spillfelt 23, med destinasjonsfelt 10. Når en spiller lander på feltet 23, vil aksjonen som er tilknyttet feltet, utføre aksjonen på spilleren, som i dette tilfelle er å flytte spilleren til nytt felt 10. (m.a.o. spilleren ramler ned stigen fra felt 23 til felt 10).

3.5.5 Player

Representerer en spiller i spillet. En spiller har et navn, og «bor» til enhver tid på et felt. En spiller kan bli **plassert på et felt** og kan også **bevege seg et antall steg** på spillbrettet. Når spilleren når siste felt eller passerer siste felt, har spilleren **nådd slutten av spillet** (mål).

3.5.6 Dice og Die

I dette spillet skal det benyttes **terninger** (engelsk: Dice). Samlingen av alle terninger er representert av klassen **Dice**. En enkelt terning er representert av klassen **Die**. Terningene rulles samtidig, og øynene summeres.

3.6 Enhetstester

Det skal lages **enhetstester** til samtlige klasser som utgjør «entiteter»-, «modell»- og «logikk»-klasser. Husk både **positive** og **negative** tester.

Et lite råd: Ikke stol blindt på alt AI (GitHub CoPilot eller lignende) foreslår som enhetstester. Det er ikke alltid at disse testene har god kvalitet

3.7 Noen gode råd

Start med det enkle først, som f.eks. klassene **Dice** og **Die**. Implementer disse, og lag enhetstester for disse for å verifisere at de fungerer riktig. Ta deretter f.eks. **Player** etterfulgt av **Tile**, **Board** og **BoardGame**. Vent med **Actions** til senere.

Lag til slutt en enkel tekstbasert brukergrensesnitt-klasse (f.eks. «BoardGameApp») der du setter opp ett spill med to terninger og noen navngitte spillere, og kjører spillet med utskrift til konsoll som for eksempel kan se slik ut:

```
The following players are playing the game:
Name: Arne
Name: Ivar
Name: Majid
Name: Atle

Round number 1
Player Arne on tile 8
Player Ivar on tile 6
Player Majid on tile 10
Player Atle on tile 3

Round number 2
Player Arne on tile 16
Player Ivar on tile 15
Player Majid on tile 22
Player Atle on tile 9

Round number 3
Player Arne on tile 23
Player Ivar on tile 19
Player Majid on tile 29
Player Atle on tile 15

.....

Round number 10
Player Arne on tile 72
Player Ivar on tile 80
Player Majid on tile 82
Player Atle on tile 74

Round number 11
Player Arne on tile 81
Player Ivar on tile 90
Player Majid on tile 88
Player Atle on tile 78

Round number 12

And the winner is: Ivar
```

Del 2 av 3

Filbehandling, designmønstre og forberedelse for GUI

4 Mappe Del 2 av 3

I denne delen av mappen skal dere

- implementere **filbehandling** for *spillere* og *spillbrett (Board)*
- innføre designmønstre
- lage egne unntaksklasse(r) (exception)
- forberede for grafisk brukergrensesnitt (GUI)

Men først noen avklaringer fra Del 1.

4.1 Avklaringer og tips fra del 1

Noen tips og avklaringer fra første del av mappeoppgaven:

4.1.1 Enkel tekstbasert UI

I slutten av del 1 ba vi dere lage en **enkel** tekstbasert brukergrensesnitt-klasse (f.eks. «BoardGameApp») der du setter opp ett spill med to terninger og noen navngitte spillere, og kjører spillet med utskrift til konsoll. Dette kun for å teste at **forretningslogikken** i applikasjonen din fungerer som den skal. Her er eksempelkode på hvordan dette *kan* se ut:

Kodeliste 2: Eksempelkode for enkelt tekstbasert brukergrensesnitt

```
public class BoardGameApp {

    BoardGame boardGame = new BoardGame();

    ....

    public void start() {
      this.boardGame.createBoard(); // Creates a standard board of 90 linked tiles
      this.boardGame.createDice(2); // Creates 2 Dice

      this.boardGame.addPlayer(new Player("Arne", this.boardGame));
      this.boardGame.addPlayer(new Player("Ivar", this.boardGame));
      this.boardGame.addPlayer(new Player("Majid", this.boardGame));
      this.boardGame.addPlayer(new Player("Atle", this.boardGame));

      // Play the game
      listPlayers();
      int roundNumber = 1;
      while (!this.boardGame.isFinished()) {
        System.out.println("Round number " + roundNumber++);
        this.boardGame.playOneRound();
        if (!this.boardGame.isFinished()) {
          showPlayerStatus(); // Displays the names of the players and which tile they are on
        }
        System.out.println();
      }
      System.out.println("And the winner is: " + this.boardGame.getWinner().getName());
     }

     ....
}
```

4.1.2 Nivå 2 – flere ulike spill

Vi anbefaler at dere lager et fullverdig klassisk **stigespill** før dere vurderer nivå 2. Gjennom implementasjonen av stigespillet (nivå 1) vil dere innhente viktig erfaring og lære å kjenne valgt design noe som gjør det enklere å ta valg i forhold til nivå 2. Når det gjelder vurdering (karakter) er det

tilstrekkelig at øvrige spill fortsatt er av typen der spillerne kun beveger seg i en retning (forover), og feltene på brettet kun er lenket med ett annet felt. Tenk «forenklet Monopol» (uten hus, hotell, sjanse osv) fremfor «Den forsvunne diamant». Hele formålet med nivå 2 er at dere får vist at med valgt design er løsningen såpass generell at den kan benyttes for et utvalg varianter av spill. Men har dere lyst å utfordre dere selv, så er det selvsagt fritt frem.

4.2 Rapporten

I rapporten bør dere i samtalen for del 2 ha fått på plass det meste av

- innledningen og problembeskrivelsen med Use Case diagram og kort oppsummering av kravspek (funksjonelle og ikke-funksjonelle krav)
- begreper - domenespesifikke begreper med beskrivelse av tolkningen/forståelsen av begrepet.
- teori - kort presentasjon av teorier dere har benyttet dere av i implementasjonen. Husk å henvise til kilde. Dersom dere har kommet over teorier rundt spilldesign bør dette også nevnes (og refereres til).
- metode - Hvordan dere planlegger å jobbe (metodikk/prosess) og hvilke verktøy (med versjoner) dere har/vil benytte
- resultat - Beskriv løsningen så langt som dere har kommet

Husk å få tilbakemelding på rapporten ved del 2 samtalen!

4.3 Filbehandling

Siden denne applikasjonen skal kunne tilby flere varianter av brettspill, vil det være fornuftig å kunne lagre de ulike spillene i form av filer som applikasjonen kan lese ifra. Det kan også være greit å kunne lagre listen over spillere, slik at man slipper å legge inn spillere manuelt hver gang man spiller spillet.

Funksjonelle krav knyttet til filbehandling:

- Det skal være mulig å lagre listen over spillere til en CSV-formatert tekstfil.
- Det skal være mulig å lese spillere fra en CSV-formatert tekstfil.
- Det skal være mulig å lagre et «hardkodet» spillbrett (et brett konfigurert i Java-koden) til en JSON-formatert tekstfil.
- Det skal være mulig å lese et spillbrett fra en JSON-formatert tekstfil.

4.3.1 Plassering av filer

Dere må selv ta stilling til **hvor** i mappestrukturen til Maven-prosjektet slike brukerdefinerte filer skal ligge. Beskriv dette i rapporten og begrunn valgene dere gjør.

4.3.2 Fil med spillere

I applikasjonen må det være mulig for en bruker å velge å lagre spillere, samt å lese inn spillere fra fil. Filen skal være en enkel tekstfil. Hver linje skal ha spillerens navn og brikke, separert med et komma. Eksempel:

```
Arne,TopHat
Ivar,RaceCar
Majid,Cat
Atle,Thimble
```

I fila er brikken identifisert med et unikt navn. Men den kan gjerne representeres med et bilde eller ikon i det grafiske brukergrensesnittet. Hva slags type brikker dere ønsker å støtte er opp til dere.

Fila skal ha filendelsen ".csv", siden den inneholder komma-separerte verdier.

4.4 Filer for ulike spill

En bruker må kunne lese et spill fra en fil. Med andre ord skal det være mulig å lese et **Board**-objekt med tilhørende **Tile**-objekter og eventuelle **TileAction**-objekter fra en fil. Her skal dere benytte **JSON** som filformat. La gjerne filene ha ending av formen **.json**.

4.4.1 Om JSON-formatet

JavaScript Object Notation (JSON) er et **tekstbasert** format for å representere strukturert data basert på JavaScript-objektsyntaks. JSON benyttes mye til utveksling av data mellom ulike deler av et program (mellom frontend og backend f.eks.) og til lagring og serialisering/deserialisering av data. JSON er uavhengig av programmeringsspråk og plattformer (Windows, Linux, MacOS etc). Du kan lese mer om JSON her: https://en.wikipedia.org/wiki/JSON. Eksempel på en JSON-fil ser du i Kodeliste 3.

Kodeliste 3: Eksempel på en JSON fil

```
{
  "addressBook": [
    {
      "name": "Alice Johnson",
      "age": 30
    },
    {
      "name": "Bob Smith",
      "age": 25
    },
    {
      "name": "Charlie Davis",
      "age": 40
    }
  ]
}
```

Kodeliste 4: Java kode av klassen Person

```
class Person {
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    .....
}

// AddressBook class
class AddressBook {
    List<Person> persons = new ArrayList<>();

....
}
```

Som vi ser av eksempelet, lagres data i **key-value par** i JSON-fila, der **key** (nøkkelen) er navnet på feltet i klassen (her «name» for name-feltet i Person-klassen), og **value** (verdien) er veridien til feltet (her "Alice Johnson"). En samling av slike key-value par tilhørende **et objekt** er omsluttet av «{...}». Samlinger (som f.eks. en **Array**) er omsluttet av «[..]».

4.4.2 Lese og skrive JSON filer i Java

Siden JSON er en «standard» tekstfil, er det svært lett å lese (og skrive) JSON filer i Java. Men det kan være litt utfordrende å tolke informasjonen i tekstfilen (eller «parse» som vi også kaller det). Derfor kan det være greit å benytte eksisterende **biblioteker** som utfører selve parsingen for oss. De mest populære bibliotekene er (se også: https://www.baeldung.com/java-json):

- Jackson
- Gson
- json-io
- Genson
- JSON-P

Vi foreslår at dere bruker GSON. Dette biblioteket ble opprinnelig utviklet for interne formål hos Google, med versjon 1.0 utgitt 22. mai 2008 under vilkårene i Apache License 2.0. Den nyeste versjonen, 2.11, ble utgitt 20. mai 2024. Nyttige kilder om GSON:

- https://github.com/google/gson
- https://www.tutorialspoint.com/gson/gson_quick_guide

GSON biblioteket har metoder for «automatisk» å serialisere og deserialisere objekter til/fra JSON-format gjennom metodene ***fromJson()*** og ***toJson()*** i klassen **Gson**. Vi anbefaler å ***ikke*** bruke

disse metodene. Benytt isteden klassene **JsonObject** og **JsonArray** som vist i eksempelet i Kodeliste 5.

Kodeliste 5: Eksempelkode for serialisering og deserialisering av addressebok til JSON

```
/**
* Returns a JsonObject made up of the address book and all it's content,
* provided by the parameter.
*
* @param addressBook the address book to convert to JSON
* @return a JsonObject representing the addressbook.
*/
public JsonObject serializeAddressBook(AddressBook addressBook) {
    JsonObject addressBookJson = new JsonObject();
    JsonArray personsJsonArray = new JsonArray();

    for (Person person : addressBook.getPersons()) {
        JsonObject personJson = new JsonObject();
        personJson.addProperty("name", person.getName());
        personJson.addProperty("age", person.getAge());
        personsJsonArray.add(personJson);
    }

    addressBookJson.add("persons", personsJsonArray);
    return addressBookJson;
}

/**
* Returns an object of AddressBook based on parsing the JSON-formatted text string provided
* by the parameter.
*
* @param jsonString the address book as a JSON formatted string
* @return an object of AddressBook as parsed from the JSON string.
*/
public AddressBook deserializeAddressBook(String jsonString) {
    JsonObject addressBookJson = JsonParser.parseString(jsonString).getAsJsonObject();
    JsonArray personsJsonArray = addressBookJson.getAsJsonArray("persons");

    List<Person> persons = new ArrayList<>();
    for (JsonElement personElement : personsJsonArray) {
        JsonObject personJson = personElement.getAsJsonObject();
        String name = personJson.get("name").getAsString();
        int age = personJson.get("age").getAsInt();
        persons.add(new Person(name, age));
    }

    return new AddressBook(persons);
}
```

4.4.3 JSON-format for Board Game

Kodeliste 6 viser et eksempel på et JSON-format som kan benyttes i brettspill applikasjonen.

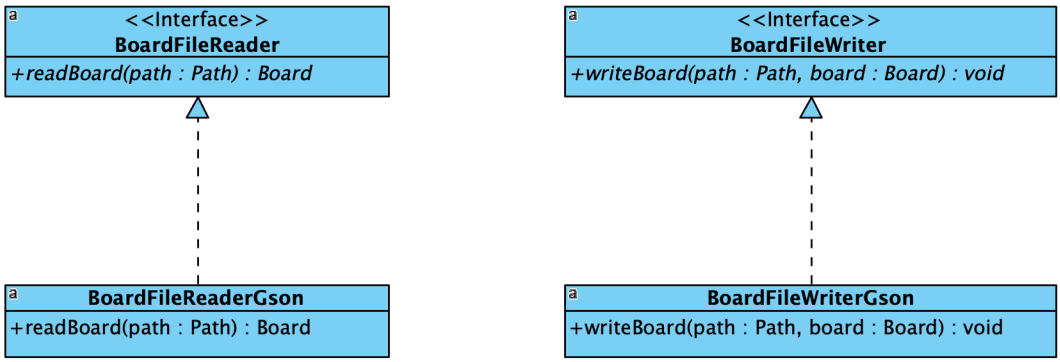
Kodeliste 6: Eksempel på en JSON fil

```
{
  "name": "Stigespill 90",
  "description": "Et klassisk stigespill med 90 felt og 2 stiger.",
  "tiles": [
    {
      "id": 1,
      "nextTile": 2
    },
    {
      "id": 2,
      "nextTile": 3
    },
    {
      "id": 3,
      "nextTile": 4
    },
    {
      "id": 4,
      "nextTile": 5,
      "action": {
        "type": "LadderAction",
        "destinationTileId": 14,
        "description": "Ladder from 4 to 14"
      }
    },
    {
      "id": 5,
      "nextTile": 6
    },


    .....

    {
      "id": 87,
      "nextTile": 88
    },
    {
      "id": 88,
      "nextTile": 89
    },
    {
      "id": 89,
      "nextTile": 90
    },
    {
      "id": 90
    }
  ]
}
```

4.4.4 Tips til klasser for filhåndtering

Å lagre objekter av klasser til fil bør håndteres utenfor klassen som skal lagres. Det kan være fristende å tenke at når men designer i henhold til **Responsibilitydriven Design**, altså at den som eier dataene bør behandle dataene. Men her er det altså en fordel å holde lagringen utenfor klassene som skal lagres. Dette også for å ha fleksibilitet; det kan være vi ønsker å ha mulighet til å lagre objekttilstandene på ulike format (CSV, XML, JSON osv) i ett og samme program. Av samme grunn er det også lurt å da definere et **interface** med nødvendige metoder for å henholdsvis lagre til fil

og lese fra fil. Da kan vi opprette ulike klasser for hver filtype og la disse klassene implementerer interfacene, som vist i Figur 4.



Figur 4: Interface for lesing og skriving fra/til fil med respektive GSON implementasjoner.

4.5 Designmønstre

Det er gode muligheter for å benytte **designmønstre** i dette prosjektet. For eksempel kan det være greit å tilby et par forhåndsdefinerte varianter av stigespill «ut av boksen», altså stigespill som ikke nødvendigvis ligger lagret som JSON-filer som brukeren har tilgang til.

4.5.1 Factory

Lag en klasse `BoardGameFactory` som lager noen varianter av brettspill (ulikt antall stiger og plassering av disse). Her kan dere for eksempel velge å definere 2-3 stigespillbrett som er «hardkodet» i brettfabrikken, i tillegg til at brettfabrikken for eksempel kan lese hvilke brett-filer som er lagret på harddisken.

4.5.2 Observer

Når dere etterhvert skal implementere grafisk brukergrensesnitt, kan det være fornuftig å implementere en måte der GUI får beskjed fra BoardGame-klassen når spillet har endret «tilstand». Eksempel på endret tilstand kan være når en spiller har flyttet/utført sitt trekk. En annen tilstandsendring kan være at en vinner har blitt kåret.

Lag et grensesnitt (*interface*) `BoardGameObserver` som øvrige objekter som ønsker varsel om endringer hos `BoardGame` må implementere. Utvid klassen `BoardGame` med mekanismer for å registrere obersvatører og varsle disse.

4.6 Domene-spesifikke unntak (*Exceptions)*

Identifiser og implementer *domenespesifikke unntaksklasser*. Ett eksempler på unntak som kan oppstå i spillet er ved parsing av JSON-fil feiler. GSON-biblioteke definerer sine egne unntaksklasser som `JsonSyntaxException`, men dersom vi på et senere tidspunkt velger å bytte JSON-bibliotek, vil det være en fordel om vi kan unngå å måtte endre i for mye av koden vår. Lag et eget unntak for å signalisere feil ved parsing av JSON-fil. Husk at standard unntak som f.eks. `IllegalArgumentException` **kun** skal benyttes dersom argumenter i en metode har ugyldig verdi.

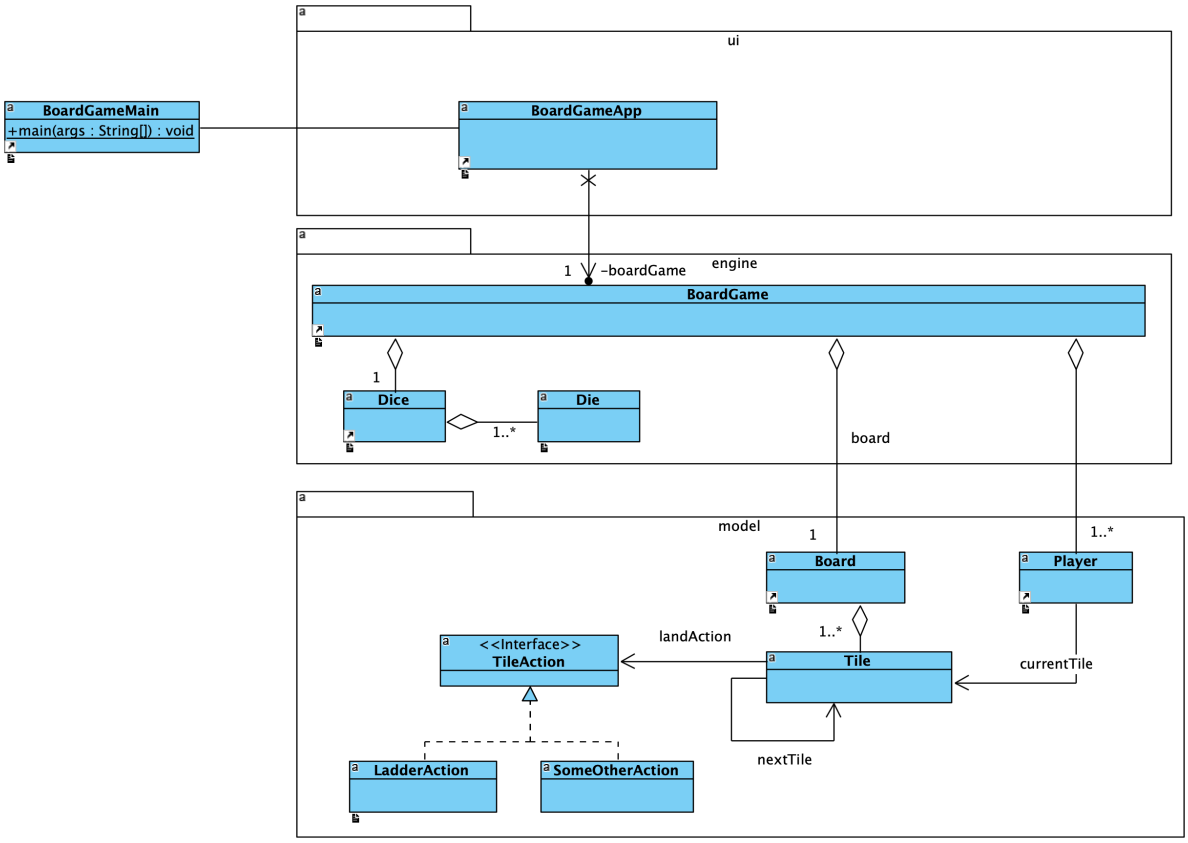
4.7 Forberede for grafisk brukergrensesnitt (GUI)

I del 1 startet dere med å lage skisser til brukergrensesnitt. Nå når vi så smått har startet på introduksjon til JavaFX, vil dere oppdage at det er både komplisert og fort mye arbeid å lage avanserte

brukergrensesnitt. Spesielt utfordrende blir det dersom dere har valgt å lage et grensesnitt med mange ulike skjermer dere må veksle mellom. Tenk derfor KISS («Keep it Simple, Stupid»). Hold dere til ett vindu og bytt heller deler (pane) innenfor vinduet.

Vi anbefaler også at dere gjør kortspill-øvinga **før** dere begynner å kode GUI i Mappen. Arbeidskravet gir dere en god intro i å bygge et GUI med JavaFX.

4.7.1 BoardGame som *fasade*



Figur 5: Klassen `BoardGame` som *fasade* for brukergrensesnittet

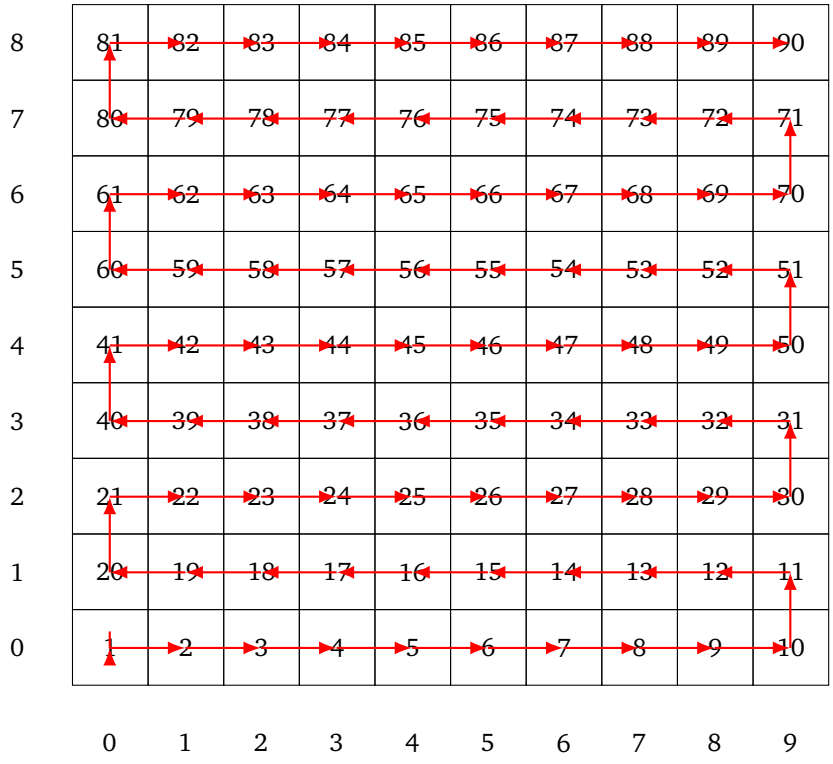
I litt mer komplekse systemer, er det vanlig å operere med det vi kaller et ***fasade-objekt*** som utgjør grensesnittet mellom brukergrensesnittet og den underliggende logikken. Dette er så vanlig, at det er definert et eget *designmønster* kalt «Fasade-mønsteret». Et fasade-objekt skal kunne tilby brukergrensesnitt-laget all funksjonalitet for å sette opp et brettspill, opprette spillere og terninger, samt for å spille spillet. Som vist i Figur 5, kan typisk klassen `BoardGame` utøve rollen som *fasade-objekt*. Et godt laget fasade-objekt gjør at det er mulig å relativt enkelt bytte type brukergrensesnitt, fra tekstbasert, til grafisk og til og med til web.

Gjør nødvendig *refaktorering* av klassen `BoardGame` slik at den fremstår som en god fasade for brukergrensesnittet.

4.7.2 Grafisk spillbrett - noen tips

Et viktig element i brukergrensesnittet vil naturligvis være selve **spillbrettet**. En utfordring som da raskt dukker opp er hvordan håndtere å vise at en spiller f.eks. står på en gitt brikke (tile). Så lenge brettet er et rutenett, er ikke dette så utfordrede, men hva om du ønsker å lage et brettspill der feltene som spillerne skal bevege seg på ligger «ustrukturert» på spillbrettet (som f.eks. i spillet «Den forsvunne diamant»)? En måte å håndtere dette på er å innføre **koordinater** for feltene i spillbrettet

(i tillegg til ID) som illustrert i Figur 6. Her vil for eksempel feltet med ID 57 være plassert i posisjon (5,3).



Figur 6: Spillebrett for Stigespillet med koordinater

Som en forberedelse til å lage det grafiske brukergrensesnittet, bør dere derfor vurdere å

- utvide `Tile`-klassen med koordinater (husk å da også oppdatere JSON-filen med koordinatene)
- utvide `Board`-klassen med felt for *antall rader* og *antall kolonner*.

Et spillbrett i brukergrensesnittet kan gjerne være et grafisk bilde, der spillerene har sin grafiske visuelle representasjon som legges over spillbrettet. I JavaFX vil det være naturlig å benytte klassen `Canvas` for å lage det visuelle spillbrettet. En grei tutorial som viser hvordan man kan legge bilder over et canvas, og bevege disse bildene, finnes i dette dokumentet: The JavaFX Canvas - Part I: Using Images. Et alternativ til å benytte `Canvas` er å bruke klassen `javafx.scene.layout.Pane`-klassen. Se dokumentasjonen til klassen `Pane` her.. Denne klassen tillater plassering av komponenter med x-, y-koordinater.

Koordinatsystemet til JavaFX klassene `Pane` og `Canvas` vil ikke være det samme som koordinatsystemet på spillbrettet. Vi må derfor regne om mellom spillbrettes koordinatsystem og lerretet i brukergrensesnittet sitt koordinatsystem. Matematikken for å gjøre dette finner du i Vedlegg A.2.

Del 3 av 3

Ferdigstillelse av GUI og egne utvidelser

5 Mappe Del 3 av 3

I denne delen av mappen skal dere

- implementere komplett grafisk brukergrensesnitt for spillet
- ferdigstille funskjonelle krav
- innføre egne utvidelser

5.1 Implementere komplett GUI

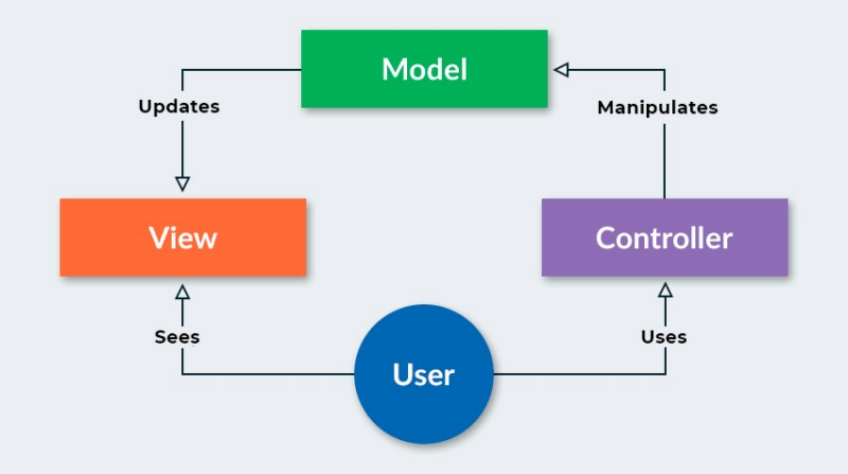
Nå skal du implementere og ferdigstille det grafiske brukergrensesnittet. Har du/dere gjort et godt forarbeid i del 2 av mappen, og laget skisser i f.eks. Figma, er mye av arbeidet gjort. Du kan for eksempel gjenbruke styling-informasjon fra Figma (i form av CSS) for å style ditt JavaFX brukergrensesnitt. Her er noen lenker til ressurser om bruk av CSS og JavaFX:

- https://jenkov.com/tutorials/javafx/css-styling.html
- https://openjfx.io/javadoc/24/javafx.graphics/javafx/scene/doc-files/cssref.html

Selv om vi har krevd at dere ikke skal implementere GUI ved bruk av FXML, er det selvsagt ingenting i veien for å bruke verktøyet **Scenebuilder** som et alternativ til Figma for å lage skissen til GUI. Da vil du også få litt hjelp til å implementere GUI'et i Java-kode siden Scenebuilder viser deg *scenegraphen* (nederst venstre vindu).

5.1.1 Model-View-Controller

En kort recap av Model-View-Controller (MVC): Grafiske brukergrensesnitt følger svært ofte (om ikke alltid) MVC-arkitekturmønsteret som vist i Figur 7. En svært god beskrivelse av MVC kan du finne her [1]: https://codegym.cc/groups/posts/303-part-7-introducing-the-mvc-model-view-controller-pattern.

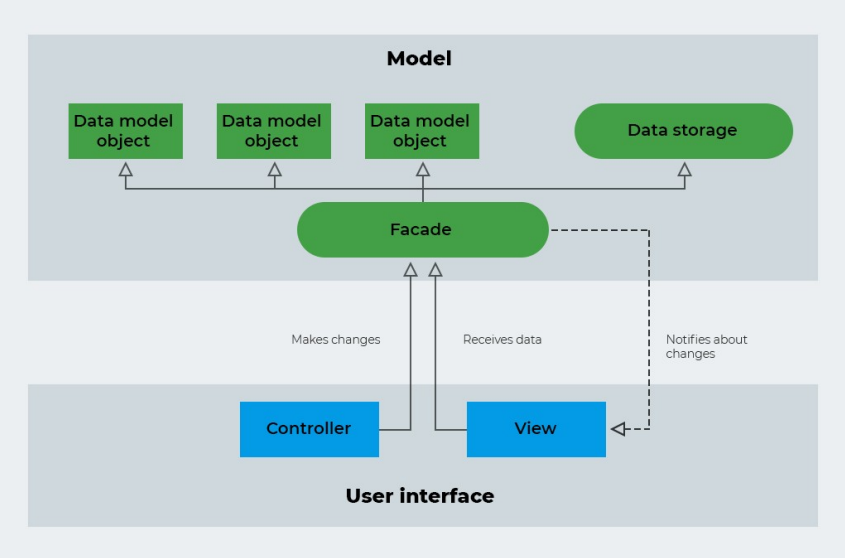


Figur 7: Model-View-Controller (MVC) [1]

Merk at begge rollene «View» og «Controller» tilhører **brukergrensesnittet**. Disse rollene er også svært tett koblet, så det vil være naturlig at de tilhører samme *pakke*. Dette er godt illustrert i Figur 8.

5.1.2 Observer-pattern og MVC

Som illustrert i Figur 8, bør vi designe arkitekturen vår slik at klassene som utgjør «Modell»-rollen, ikke har kjennskap til hverken **Controller** eller **View** rollen. Likevel er det viktig at brukergrensesnittet blir oppdatert når det har foregått endringer i modellen sin tilstand (som at en spiller har flyttet). Dette er et klassisk eksempel på et sted der designmønsteret **Observer** virkelig passer. Når modellen



Figur 8: The full picture of MVC [1]

endrer tilstand, kan modellen enten gi beskjed til sine abonnenter om at endring har skjedd (ved f.eks. å kalle en metode av typen ***update()*** på alle abonnenter, og overlate til abonnenten å hente inn eventuelle endringer ved at abonnenten (her «View»-rollen) kaller nødvendige metoder hos modellen, eller modellen kan sende med informasjon om endringen i sitt kall til ***update()*** metoden til abonnentene. Siste variant er metoden som f.eks. benyttes i forbindelse med eventhåndtering i GUI.

Finn en god måte å implementere Observer-mønsteret på mellom View-rollen og Model.

5.2 Ferdigstille funksjonelle krav

De funksjonelle kravene er beskrevet i kapittel 2.1.3. Disse skal være tilgjengelig for brukeren gjennom det grafiske brukergrensesnittet.

5.3 Innføre egne utvidelser

Herfra er det fritt frem for alle slags utvidelser av applikasjonen. Det forventes at dere gjør noen utvidelser utover det som er spesifisert i oppgaveteksten, men poenget er ikke å legge til mest mulig funksjonalitet. ***Kvalitet er viktigere enn kvantitet.*** Velg utvidelser som gir deg mulighet til å vise frem at du behersker ulike læringsmål i emnet.

5.4 Ferdigstill rapporten

Ikke glem rapporten. Husk at rapporten skal holdes innenfor 2500-3000 ord. Igjen er **kvalitet viktigere enn kvantitet**. Husk også at det er ***resultat*** og ***diskusjon***-kapitlene som er de viktigste og bør være de med flest ord. Husk også i diskusjonen å diskutere **din løsning** opp mot **metoden(e)** du/dere har valgt og **teoriene** dere har lagt til grunn for designet/løsningen.

6 Viktige sjekkpunkter

Når dere løser oppgaven, bør dere passe på følgende:

Maven

- Er prosjektet et Maven-prosjekt med fornuftige prosjekt-verdier og gyldig katalogstruktur (korrekte versjoner av Java, JavaFX osv iht til kravene)?
- Kan man kjøre Maven-kommandoer for å bygge, teste, pakke, samt starte GUI uten at det feiler?
- Er det mulig å bygge, teste, pakke, samt starte GUI fra terminalen med ***mvn*** (`mvn clean compile javafx:run`)?

Versjonskontroll med git

- Er prosjektet underlagt versjonskontroll med sentralt repo?
- Sjekkes det inn (commit) jevnlig? Av **begge** gruppemedlemmene?
- Beskriver commit-meldingene endringene på en kort og konsis måte?
- Benyttes grener («branches») på en hensiktsmessig måte?
- Benyttes **tags** på riktig måte i forhold til versjonen i pom.xml og på applikasjonen?

Enhetstester

- Har enhetstestene beskrivende navn som dokumenterer hva testene gjør?
- Følger de mønstret Arrange-Act-Assert?
- Tas det hensyn til både positive og negative tilfeller?
- Er testdekningen fornuftig?
- Er testene godt nok dokumentert til at det klart går frem hva som testes, og hva forventet resultat av testene er?

Filbehandling (fra Del 2 og ut)

- Er det mulig å lese spillere fra fil og lagre spillere til fil på oppgitt format?
- Er det mulig å lese og lagre en beskrivelse av et spill i JSON-format?
- Hensyntas feil i data og filformat ved innlesning?
- Lukkes filressurser på en trygg måte etter bruk?
- Har man et bevisst forhold til hvor ulike typer filer lagres?

GUI

- Oppfyller GUI kravene og betingelsene som beskrevet i mappeoppgavens del 3?
- Er det mulig å forstå hvordan applikasjonen skal brukes uten noen bruksanvisning?
- Håndteres feilsituasjoner på en fornuftig måte, f.eks. med beskrivende feilmeldinger, gjerne med bruk av dialogbokser?

Kodekvalitet

- Er koden godt dokumentert iht. JavaDoc-standard? (NB! samtlige metoder som er public **skal** dokumenteres, også enkle metoder som get()- og set()-metoder).
- Er koden robust (unntakshåndtering, validering med mer)?
- Har variabler, metoder og klasser beskrivende navn?
- Er klassene gruppert i en logisk pakkestruktur?
- Har koden god struktur, med løse koblinger og høy kohesjon?

- Benyttes arv og grensesnitt slik at løsningen er fleksibel og lett å utvide/endre?
- Benyttes funksjonell programmering (lambda og streams) på en hensiktsmessig måte der det er naturlig?
- Benyttes designmønstrene Factory og Observer i henhold til oppgaveteksten?
- Benyttes andre designmønstre der det er hensiktsmessig?
- Benyttes linter (f.eks. SonarLint) og tilsvarende verktøy (f.eks. CheckStyle) for kvalitetssjekk?

A Vedlegg

Her finner dere lenke til nyttige ressurser, samt tips og triks.

A.1 Nyttige verktøy

For en oversikt over nyttige verktøy til bruk ved utvikling av programvare, se vår Wiki: https://www.ntnu.no/wiki/display/idatx1001. (Se under menyen «Verktøy og IDE-er»).

A.1.1 UML-modellering

PlantUML Finnes som plugin til IntellJ. Gratis/Open Source. Støtter samtlige UML-diagramtyper med mer. Se: https://www.ntnu.no/wiki/display/idatx1001/Plugins+for+IntelliJ+IDE

VisualParadigm Et fullskala modelleringsverktøy for UML, ER-diagrammer m.m. NTNU har site-lisens. Gå til: https://ap.visual-paradigm.com/norwegian-university-of-science-and-technology. Der finner du også lisenskoden. Husk å registrer deg med din **@stud.ntnu.no**-adresse for å få tilgang til lisensen.

A.1.2 Wireframes og Mockups av GUI

Balsamiq NTNU har lisens på **desktop-versjonen** av Balsamiq (IKKE web-versjonen). Last ned fra: https://balsamiq.com/wireframes/desktop/. License Key:

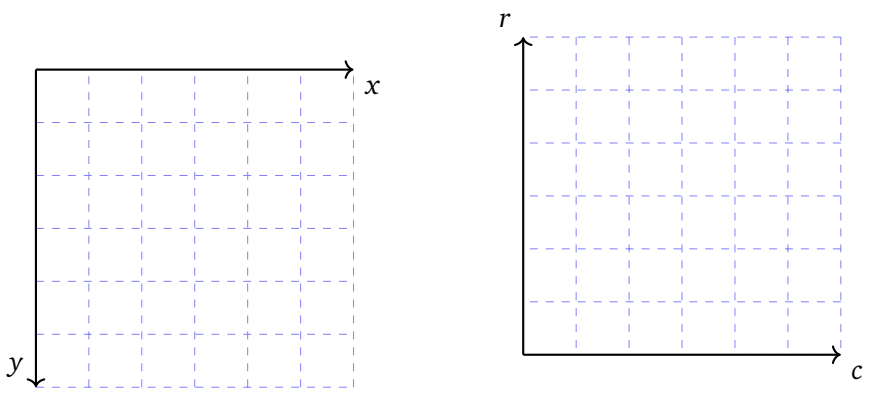
`IDATAvarious2024|QhlFeJxzCncxiQ+p8XRxDHFUKEssyswvLVYwMjAyqTE0NzMxM7Q0MgABAOt9CsU=`
License End Date: Dec 31, 2025

Figma Figma tilbyr gratis student-lisenser. Registrer deg med din student-epost. Figma finner du her: https://www.figma.com/

A.2 Konvertering mellom koordinatsystem

Vi har to koordinatsystem som vist i Figur 9.

- Kanvaset i brukergrensesnittet benytter x og y koordinater, med origo øverst til venstre og $x \in \{0, 1, \ldots, x_{\max}\}$ og $y \in \{0, 1, \ldots, y_{\max}\}$ (se Figur 9a)
- Spillbrettet opererer med r og c for *row* og *coloumn* der origo er nederst til venstre og $r \in \{0, 1, \ldots, r_{\max}\}$ og $c \in \{0, 1, \ldots, c_{\max}\}$ (se Figur 9b)



(a) Koordinatsystem for canvas **(b)** Koordinatsystem for spillbrett

Figur 9: Ulike koordinatsystemer

Så både rekkefølge og retning er forskjellig. I tillegg er skaleringen forskjellig. Vi trenger med andre ord en formel for å regne om mellom (r, c)-koordinater på spillbrettet, til (x, y)-koordinater på lerretet der spillbrettet skal tegnes opp på.

Formelen for å regne ut x og y når du kjenner r og c blir da:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{x_{\max}}{c_{\max}} c \\ y_{\max} - \frac{y_{\max}}{r_{\max}} r \end{pmatrix}$$

Tilsvarende for å regne ut r og c når du kjenner x og y:

$$\begin{pmatrix} r \\ c \end{pmatrix} = \begin{pmatrix} \frac{c_{\max}}{x_{\max}} x \\ r_{\max} - \frac{r_{\max}}{y_{\max}} y \end{pmatrix}$$

Dette forutsetter at begge koordinatsystemer har sitt origo i $(0, 0)$.

B Hvordan bygge en eksekverbar JAR-fil

Det er mulig å bygge en JAR-fil som man kan dobbelt-klikke på for å starte og kjøre applikasjonen. En måte å gjøre dette på, er å pakke **samtlige biblioteker** som applikasjonen er avhengig av inn i en stor JAR-fil sammen med klassene fra din applikasjon, en såkalt "uber-JAR". Dette kan gjøres ved å bruke en Maven plugin kalt **SHADE**. https://maven.apache.org/plugins/maven-shade-plugin/index.html

Kodeliste 7: Eksempel på build-delen i en pom.xml fil med Shade-plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-maven-plugin</artifactId>
      <version>${javafx.maven.plugin.version}</version>
      <configuration>
        <mainClass>the.full.name.of.the.mainclass</mainClass>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.6.0</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
            <transformers>
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                <mainClass>the.full.name.of.the.mainclass</mainClass>
              </transformer>
            </transformers>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Bygg applikasjonen med ***mvn package***. Det vil da bli oprpettet en JAR-fil under mappen «target». Denne JAR-filen kan du nå flytte til hvor som helst på din egen datamaskin elelr til en annen datamskin (med samme operativsystem som maskinen der applikasjonen ble bygget) og kjøre applikasjonen ved å dobbeltklikke på filen.

C KI-deklarasjon

KI-deklarasjonen som *skal* leveres inn utfylt, er vedlagt på neste side. Last ned Word-malen og fyll den ut, og last opp generert PDF.

Deklarasjon om KI-hjelpemidler

Har det i utarbeidingen av denne rapporten blitt anvendt KI-baserte hjelpemidler?

☐ Nei

☐ Ja

Hvis *ja*: spesifiser type av verktøy og bruksområde under.

Tekst

☐ **Stavekontroll**. Er deler av teksten kontrollert av:
Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinka.ai eller lignende verktøy?

☐ **Tekstgenerering**. Er deler av teksten generert av:
ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr eller lignende verktøy?

☐ **Skriveassistanse**. Er en eller flere av ideene eller fremgangsmåtene i oppgaven foreslått av:
ChatGPT, Google Bard, Bing chat, YouChat eller lignende verktøy?

Hvis ja til anvendelse av et tekstverktøy - spesifiser bruken her:

Kode og algoritmer

☐ **Programmeringsassistanse**. Er deler av koden/algoritmene som i) fremtrer direkte i rapporten eller ii) har blitt anvendt for produksjon av resultater slik som figurer, tabeller eller tallverdier blitt generert av: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* eller lignende verktøy?

Hvis *ja t*il anvendelse av et programmeringsverktøy - spesifiser bruken her:

Bilder og figurer

☐ **Bildegenerering**. Er ett eller flere av bildene/figurene i rapporten blitt generert av: *Midjourney, Jasper, WriteSonic, Stability AI, Dall-E* eller lignende verktøy?

Hvis ja til anvendelse av et bildeverktøy - spesifiser bruken her:

Andre KI-verktøy

☐ **Andre KI-verktøy.** har andre typer av verktøy blitt anvendt? Hvis ja spesifiser bruken her:

☐ Jeg er kjent med NTNUs regelverk: *Det er ikke tillatt å generere besvarelse ved hjelp av kunstig intelli- gens og levere den helt eller delvis som egen besvarelse.* Jeg har derfor redegjort for all anvendelse av kunstig intelligens enten i) direkte i rapporten eller ii) i dette skjemaet

---

Underskrift/Dato/Sted

Referanser

[1] CodeGym. «Introducing the MVC (Model-View-Controller) pattern.» (2024), adresse: https://codegym.cc/groups/posts/303-part-7-introducing-the-mvc-model-view-controller-pattern (sjekket 21.03.2025).