Prosjektrapport i IDATG2001

War Games Application

Laget av:

Martin Hegnum Johannesen

NTNU i Gjøvik

Mai 24, 2022

Innledning

I denne prosjektrapporten skal jeg forklare hvordan jeg har bygget opp en løsning til Wargames-prosjektet, hvilke valg jeg har tatt og hvordan jeg har benyttet ulike designprinsipper. Prosjektet er blitt gitt i oppgave i emnet IDATG2001, og går ut på å utvikle et program som kan simulere et slag mellom to hærer i en krig.

Innhold

Kravspesifikasjon

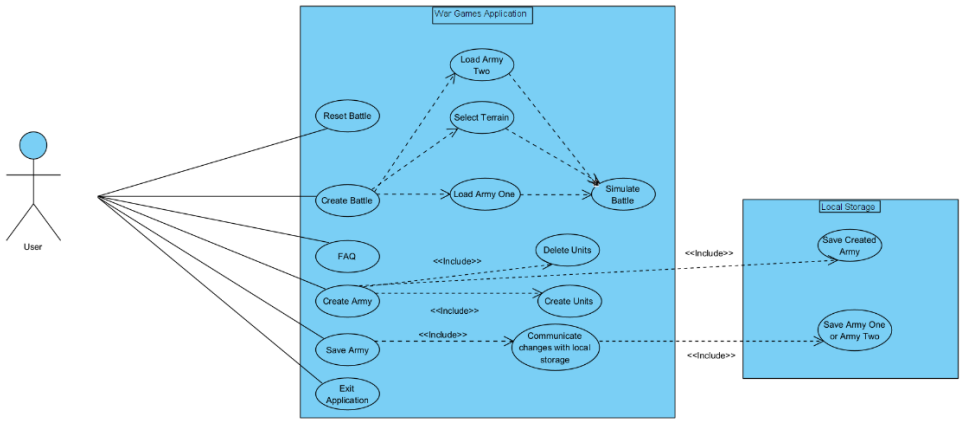
Programmet skal være med full fungerende funksjonalitet uten uforutsette feil. Formålet med systemet er å kunne simulere et slag mellom to armeer.

De funksjonelle kravene

- Det er mulig å laste inn hærene fra filer og filnavn/fillokasjon oppgis.
- Terreng kan velges.
- Man kan simulere et slag, og simuleringen fungerer som forventet.
- Man blir opplyst om resultatet av simuleringen.
- Tilstanden til de to hærene blir vist i etterkant.
- Det er mulig å resette simuleringen, både ved å oppgi fillokasjoner på nytt, og uten.
- Det er mulig å lagre hærene.

De ikke-funksjonelle kravene

- Oversiktlig presentasjon av hærene, og detaljert info (klassetype, navn og liv) blir fremvist.
    - Antall av hver klassetype blir også presentert.
- Oversiktlig layout – fornuftig(e) feilmeldinger, knapper og generell bruk av programmet.



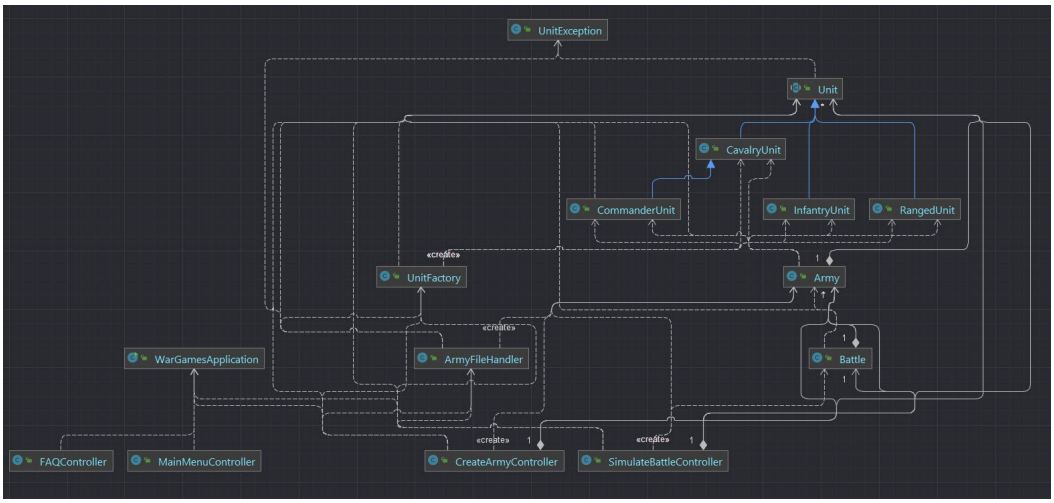
Figur 1: Use-Case Diagram – viser en oversiktlig presentasjon av funksjonaliteten til programmet

Rammer

Programmet skulle kodes i Java og GUI laget gjennom JavaFX. I tillegg kunne kjøres gjennom Maven. Rammene som har påvirket prosjektets funksjonelle og ikke-funksjonelle krav i størst grad har vært tidsfrister for de ulike innleveringene av prosjektet, generell tid, og kompetanse om GUI.

Design

Måten jeg har bygget opp løsningen på ett overordnet nivå er hovedsakelig gjennom å skille klassene i ulike «packages»: «corefunctionality» («units», Battle og Army), «exception» (ikke implementert), «filehandler» og «ui» (controllers og view). Slik har jeg beholdt klassenes fokus på deres egne oppgaver - dette for å oppnå høy kohesjon.



Figur 2 – Klassediagram, viser hvilke klasser som er avhengige av hverandre.

Et av designmønstrene jeg har benyttet meg av er «creational» mønsteret «Factory». Jeg har laget en egen klasse, «UnitFactory» som sørger for et eget grensesnitt til å opprette objekter av Unit. Videre, lar «Factory» underklassene bestemme hvilken klasse av objektet som skal opprettes – slik at den kan opprette alle typer av Unit.

Unit klassen er abstrakt klasse – så den kan ikke instansieres. Den er også av beskyttet tilgangstype, noe arv støtter. Unit er superklassen til underklassene: InfantryUnit, RangedUnit og CavalryUnit - CommanderUnit arver av CavalryUnit (se Figur 2). Underklassene arver av Unit klassen, og fullfører de abstrakte metodene (getAttackBonus og getResistBonus). De sist nevnte metodene er eksempler på polymorfe metoder, der den faktiske metoden avhenger av den dynamiske objekttypen.

Don Norman: De 6 prinsippene om design

Jeg har brukt Donald Normans 6 prinsipper (Enginess, 2014) for å designe mitt brukergrensesnitt. Ønsket mitt har vært brukergrensesnittet skal være intuitivt, enkelt å bruke, og med fungerende funksjonalitet.

Visability

Handler om at brukergrensesnittet burde være intuitivt: hvilke muligheter man har og hvordan disse kan nås. I brukergrensesnittet er det synlig hva som er knapper og hvor man skal skrive inn

verdier. Hvilke muligheter man har er intuitivt å forstå ut ifra dette, men også beskrevet i FAQ. Sidene inneholder ikke for mye informasjon, og knapper er hensiktsmessig plassert.

Feedback

Omhandler tilbakemelding til bruker etter gjennomført handling. Brukere mottar umiddelbar tilbakemelding etter å ha utført en handling for å informere dem om resultatet av handlingen. Det er lagt inn to forskjellige popUps: confirmationPopUp (dersom noe er vellykket) og errorPopUp (dersom noe går galt).

Affordance

Sammenhengen mellom hvordan ting ser ut i grensesnittet, og hvordan det brukes. Eksempelvis, i brukergrensesnittet er det intuitivt at menu item reset battle from scratch fjerner begge hærene, og at navigate/main menu tar deg til denne siden.

Mapping

Omhandler prinsippet om å ha en klar kobling mellom kontrollene i grensesnittet og hvilken effekt de har. I et godt designet grensesnitt vil kontrollene ligne effekten deres. Eksempelvis, i brukergrensesnittet mitt finner man en valgboks på CreateArmy siden. Denne brukes til å velge underklassen av Unit som skal opprettes. De fleste vet hvordan valgbokser fungerer, ettersom det er en klar kobling til effekten de har. Ved å klikke på dem vil de vise et sett med alternativer du kan velge mellom, deretter kan man klikke på den spesifikke enhetstypen man ønsker å opprette.

Consistency

Prinsippet om å være konsistent med hva kontrollerene i applikasjonen gjør. Eksempelvis, har jeg vært konsekvent med det samme mønsteret for å navigere seg rundt i applikasjonen - med tilbakeknapper og knapper som tar deg til de relevante sidene. Et annet eksempel er plasseringen av alt angående armyOne og armyTwo (på hver sin side).

Constraints

Begrenser brukeren av grensesnittet slik at de ikke blir overveldet med alternativer og informasjon. Dette oppfyller jeg ved å begrense brukeren av alternativer og handlinger de kan utføre i grensesnittet på spesifikke plasser. Brukeren vet dermed når de skal gå videre til neste handling. I tillegg kan ikke brukeren redigere FAQ tekstfeltet og har begrensede valgmuligheter av enhetstyper når brukeren skal opprette en hær.

Implementasjon – av design og funksjonalitet

Kode

Måten jeg har implementert designet og funksjonaliteten i form av kode har først og fremst vært fokusert på for å overholde høy kohesjon og lav kobling. Jeg har forsøkt å holde klassene til sin(e) oppgaver så langt det lar seg gjøre. Det jeg synes er mest interessant med koden min er hvordan jeg resetter simuleringen med de samme hærene, hvordan jeg har implementert terrain og bonus generelt.

Jeg valgte å bruke terrain som en parameter av datatype Character i attack-metoden til Unit og Battle konstruktør. Dette gjør at de abstrakte metodene til unit også tar imot terrain som parameter. Tanken min var at terrain er noe som kan forandre seg underveis i en simulasjon, men dette var noe jeg ikke rakk å gjennomføre. Det eneste problemet jeg måtte på med denne løsningen var at jeg måtte initialisere ett terrain når jeg skal opprette en battle.

Reset funksjonaliteten uten å måtte oppgi fillokasjon på nytt løste jeg gjennom å lage to identiske hærer når jeg leser hærene. Dersom brukeren skulle ønske å resette slaget hentes de dupliserte hærene, og erstatter da de hærene som ble simulert. Etter dette blir det laget en kopi av disse igjen, slik at hvis man skulle ønske å kjøre en ny simulering følger samme prosedyre.

```
Army readFromFileArmy = ArmyFileHandler.readArmyCsv(selectedFile);
armyTwo.setUnits(readFromFileArmy.getAllUnits());
armyTwo.setArmyName(readFromFileArmy.getName());
armyTwoTableView.setItems(armyTwoObservableList);
refreshTableView();
displayUnitCount();
duplicateArmies();
```

Figur 3 – Load Army metode. I denne prosessen setter tabellen hærens verdier inn, refresher den, viser total antall utenfor tabellen og dupliserer hærene.

```
@FXML
private void duplicateArmies() {
  duplicateArmyOne = new Army(armyOne.getName());
  duplicateArmyTwo = new Army(armyTwo.getName());

  for (Unit unit : armyOne.getAllUnits()) {
    duplicateArmyOne.addToArmy(UnitFactory.createUnit(unit.getClassName(), unit.getName(), unit.getHealth()));
  }

  for (Unit unit : armyTwo.getAllUnits()) {
    duplicateArmyTwo.addToArmy(UnitFactory.createUnit(unit.getClassName(), unit.getName(), unit.getHealth()));
  }
}
```

Figur 4 – duplicateArmies metode. Etter hæren har blitt lest, dupliserer jeg den nyopprettede hæren. Her bruker jeg «Creational» designmønster i form av Factory

```
@FXML
private void refreshArmies(){
  armyOne.setUnits(duplicateArmyOne.getAllUnits());
  armyTwo.setUnits(duplicateArmyTwo.getAllUnits());
  duplicateArmies();
}
```

Figur 5 – refreshArmies metode. Metode for å oppdatere hærene

```
@FXML
private void onResetBattleClick(ActionEvent event) {
  winnerText.setText("");
  if(armyOne.hasUnits() || armyTwo.hasUnits()){
    refreshArmies();
    refreshTableView();
  }
}
```

Figur 6 – resetBattle metode. Funksjonen setter bare hærene lik deres dupliserte hærer før simuleringen, dupliserer de på nytt og oppdaterer tabellen.

Filer

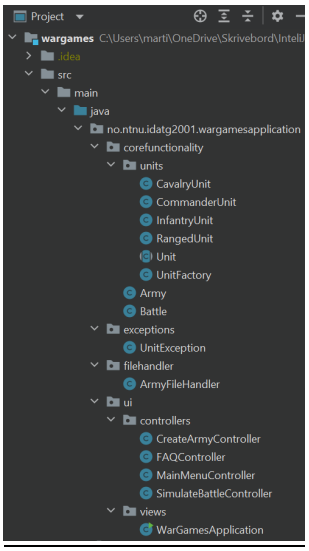
Ønsket om å bevare informasjon eller gjenstander over tid kalles persistens. Dette er noe jeg har forsøkt å oppnå ved å bruke en csv-tekstfil (andre muligheter: database eller en binær-fil). Klassen ArmyFileHandler tar hånd om dette i programmet, og består av to metoder: writeArmyCsv og readArmyCsv. Når filen blir lest brukes informasjonen, byte-ene og dataene til å gjenskape objektet i minnet. Programmet leser filen med en «try, catch», hvis den ikke klarer å lese filen kaster den et IO-unntak. Når den blir skrevet skriver den en Army til en csv-fil. Utenom dette har jeg benyttet meg av PNG filer (bilder brukt i applikasjonen) og FXML (fil for oppsett av brukergrensesnittet).

```
public static Army readArmyCsv(File file) throws IOException {
    Army army = new Army( name: "");

    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String lineOfText;
```

```
public static void writeArmyCsv(Army army, File file) throws IOException {

    try(BufferedWriter writer = new BufferedWriter(new FileWriter(file))){
        writer.write( str: army.getName()+"\n");
```

Figur 7 og 8 – ArmyFileHandler (try methods). Grunnen til at jeg har en condition i try er grunnet hvis den ikke går gjennom vil den lukka fila.

Pakker

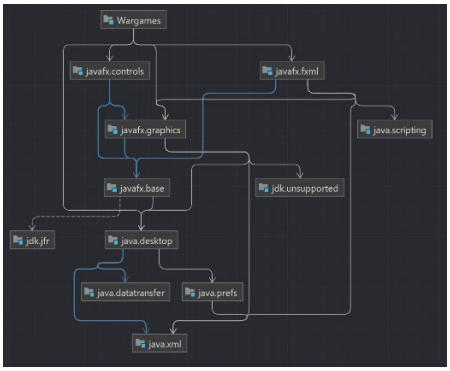


Figur 9 - Pakke struktur

En samling av kode som henger sammen logisk kalles en pakke. I programmet har jeg flyttet de ulike klassene i pakker ut ifra deres oppgaver. Jeg har hovedsakelig delt de i hovedgruppene er: «corefunctionality», «exceptions», «filehandler» og «ui».

I et større program ville man dele opp disse pakkene av større grad, da hver pakke burde inneholde få instanser.

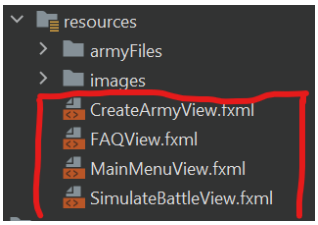
Moduler



Figur 10 - Modul diagram

I programmet benyttet jeg meg av disse modulene. Eksempelvis er «javafx-fxml», «javafx-controls» og «javafx-jupiter» ikke standard for å bygge prosjektet.

FXML



Jeg har benyttet meg av FXML for å spare tid på å lage et intuitivt brukergrensesnitt.

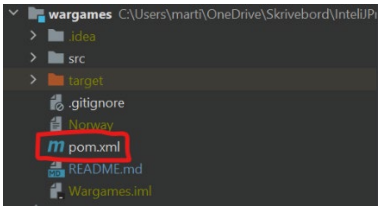
Figur 11 - FXML filene. Lokasjon: under resources

Biblioteker

Jeg har kun benyttet meg av standard Java biblioteker.

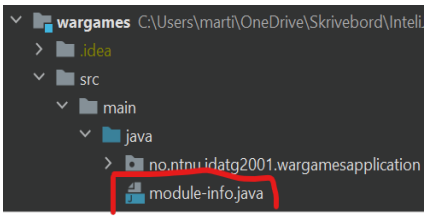
Maven

Programmet benytter seg av konvensjon byggesystemet Maven. Mappestrukturen er standard: «main» inneholder all kildekode, «test» inneholder all testkode og «target» inneholder resultatet av byggingen. I roten av prosjektet ligger «pom.xml» som er prosjektfilen til Maven, denne inneholder informasjonen som ikke er standard for å bygge prosjektet (se figur 10). Maven sikrer at eksterne bibliotek er av riktig versjon, og at prosjektet ikke er knyttet til en spesifikk instans. Programmet kan kjøres gjennom Maven, men også gjennom InteliJ grunnet «module.info».



Figur 12 - pom.xml

Figur 13 - pom.xml dependencies



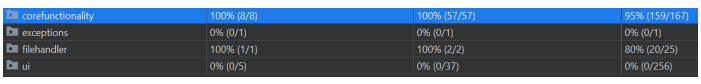
Figur 14 - module-info

Testing

Gjennom all implementasjon har jeg testet først og kodet etterpå – dette for å redusere feil. Jeg har blant annet brukt negative- og positive tester.

Alle testene har jeg satt opp etter formatet Arrange-Act-Assert. Eksempelvis, Arrange: initialiserer objektet/ene. Act: Legger til objektet i en liste. Assert: Bruker factory metoden for å lage en liste av de to objektene, og sjekker om String-en er lik unitList (se figur 15).

Figur 15 - Test eksempel (CreateUnitList)



Figur 16 - Coverage report

Prosess

Wargames-prosjektet har jeg arbeidet med i 3 ulike deler (se figur 1). Jeg har hovedsakelig forholdt meg til denne fordelingen da jeg har arbeidet med prosjektet.

Figur 17: Wargames prosjektets deler

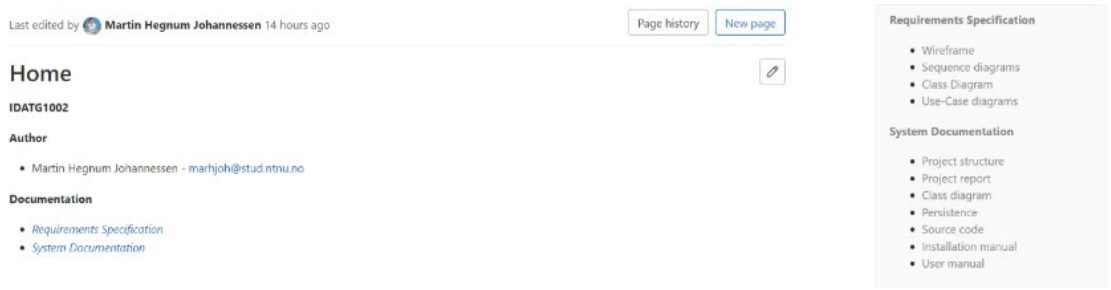
Del 1 - enheter og simuleringslogikk

Del 2 - utvidelse av programmet (filhåndtering og et grafisk brukergrensesnitt)

Del 3 - anvendelse av designmønstre og implementasjon av mer funksjonalitet.

Jeg har jeg benyttet meg av GitLab for lagring av informasjon, og har laget en Wiki som holder på alt jeg har laget i forbindelse med prosjektet. Under prosjektet har jeg blant annet benyttet meg av metodikker og verktøy som versjonskontroll med git (både gjennom Git Bash og InteliJ). Dette hjalp betydelig da jeg drev på med «debugging» av prosjektet, benytte meg av tidligere brukte metoder og generell lagring av programmet.

Under hele prosjektet har jeg programmert under en branch, dersom jeg hadde arbeidet med flere på dette prosjektet ville jeg ha opprettet minst 3 brancher til (developer, feature og bugfix). Jeg benyttet meg ikke av verktøy som kanbantavler i GitLab eller gantt charts ettersom prosjektet var individuelt. Kanbantavler vil jeg benytte meg neste gang jeg skal ha et lignende prosjekt, ettersom det gjør det enklere å overholde en helhetlig oversikt over et prosjekt.



Figur 18: GitLab Wiki - ble brukt til å oppbevare alt rundt prosjektet.

Refleksjon

Jeg vet at løsningen min tilfredsstiller de funksjonelle og ikke-funksjonelle kravene til prosjektet. Utover dette kurset har jeg lært mye om designmønstre, men jeg har ikke hatt tid til å implementere dem. Dersom jeg hadde begynt prosjektet på nytt hadde jeg gjort livet mitt lettere ved å benytte meg i mye større grad av disse.

Hvis jeg skulle endret noe ville jeg blant annet laget en «model» (denne ville f.eks. kunne bli brukt til å kalle på objekter). Jeg ville også laget en Observer klasse - denne ville brukt til å observere battle (oppdatere tableView og hærene under/og etter simulering). Dette ville vært stor hjelp for å unngå mye kobling i kontroller klassene, spesielt i update metodene. Jeg ville også ha vurdert å lage en egen bonus interface.

En situasjon som dukket opp under prosjektet var at jeg prøvde å få en tableView til å vise antall units av hver type – noe som gikk (men var en svært dårlig løsning ettersom jeg blant annet opprettet en Observable list i Army). Grunnen til at jeg lagde klassen i første omgang var at tableView-en ikke kunne ta imot en liste av units uten at jeg måtte skille dem i første omgang. Jeg endte opp med å refaktorere store deler av koden

```java
package no.ntnu.idatg1002.wargamesapplication.corefunctionality;

public class ArmyCount {
    private final String unitName;
    private final int amount;

    public ArmyCount(String unitName, int amount) {
        this.unitName = unitName;
        this.amount = amount;
    }

    public String getUnitName() { return unitName; }

    public int getAmount() { return amount; }
}
```

Figur 19 - ArmyCount klassen for tableView

```java
/**
 * This method sets up a summary list for the army
 *
 * @return a list of the different unit classes as an ObservableList of ArmyCount
 */
public ObservableList<ArmyCount> getArmyCount() {
    ObservableList<ArmyCount> result = FXCollections.observableArrayList();
    result.add(new ArmyCount( unitName: "CavalryUnit", getCavalryUnits().size()));
    result.add(new ArmyCount( unitName: "CommanderUnit", getCommanderUnits().size()));
    result.add(new ArmyCount( unitName: "InfantryUnit", getInfantryUnits().size()));
    result.add(new ArmyCount( unitName: "RangedUnit", getRangedUnits().size()));
    return result;
}

/**
 * This method sets up a list of all the units in the army
 *
 * @return a list of all the different units as an observableList of Unit.
 */
public ObservableList<Unit> getFullArmy() {
    ObservableList<Unit> result = FXCollections.observableArrayList();
    for(Unit unit : militia){
        result.add(UnitFactory.createUnit(unit.getClassName(), unit.getName(), unit.getHealth()));
    }
    return result;
}
```

Figur 20 - getArmyCount metoden ble brukt til å sette verdiene i tableView-en

Konklusjon

For å oppsummere kort har jeg har laget en løsning til Wargames-prosjektet, der jeg benytter meg av blant annet «Factory» og Don Norman 6 prinsipper om design. Programmet er av full funksjonalitet som oppgaven ber om, og mer – f.eks. muligheten til å opprette en ny hær i brukergrensesnittet

Kildeliste

Litteratur

- Barnes, D. Kolling, M. (2017). Objects First with Java (utgave: 6). University of Kent: Pearson.
- Enginess. (2014, 03. november). The 6 Principles Of Design, a la Donald Norman. Hentet fra https://www.enginess.io/insights/6-principles-design-la-donald-norman

Figurer