

Prosjektrapport

Wargames



Av Aliaan Nadeem Azam

NTNU Gjøvik

Mai 23, 2002

Innhold

1. Innledning	3
2. Kravspesifikasjon	3
2.1 Use-Case diagram	4
3. Design	5
3.1 Balsamiq Wireframes	5
3.1.1 Main menu	5
3.1.2 Army Registration	6
3.1.3 Unit registration	6
3.1.4 Army details	7
3.1.5 Load army from file	7
3.1.6 Battle simulation	8
3.2 Designprinsipper og designmønstre	8
3.3 Klassediagram	9
3.3.1 Klassediagram 1	9
3.3.2 Klassediagram 2	10
4. Implementasjon	11
4.1 Model-View-Controller designmønster	12
5. Prosess	12
5.1 Commits	13
6. Refleksjon	14
7. Konklusjon	14

1. Innledning

Jeg har fått i oppgave i emnet IDATG2001 om å utvikle et program kalt Wargames. Dette programmet skal simulere et slag mellom to armeer som kriger mot hverandre. Prosjektet var delt opp i tre deler, hvor den første delen gikk ut på å kode enhetene som finnes i armeene. I den andre delen av prosjektet skulle vi utvide programmet, implementere filhåndtering og bygge et grafisk brukergrensesnitt. I den tredje og siste delen av prosjektet skulle man legge til mer funksjonalitet, anvende designmønstre og fullføre programmet fullstendig.

2. Kravspesifikasjon

Løsningen min på prosjektet gir brukeren muligheten til å opprette en arme fra bunn, akkurat som brukeren selv ønsker. Programmet er designet slik at brukeren har frihet til å opprette armeen som de vil. Brukerens oppgave er å opprette en arme som skal brukes for å kjempe mot en fiendtlig arme. Den fiendtlig armeen kan lastes fra fire forskjellige filer. Disse to armeene skal kjempe mot hverandre i et slag. Man starter ved å først opprette en arme ved å velge navn for den. Deretter går man videre for å redigere armeen som man ønsker. Brukeren legger inn enheter i armeen sin manuelt ved å velge en enhetstype og skrive inn navn for enheten. Man kan legge velge mellom fire forskjellige enhetstyper, som alle har forskjellige attributter. Etter at man har valgt enhetstype og navn for enheten sin og lagt den til i armeen ved å trykke på «Add unit to army»-knappen, vil den vises i en liste i brukergrensesnittet. I denne listen kan du se all informasjonen om enhetene i armeen din, som enhetstype, navn, helse, attack og armor. Brukeren har også mulighet til å laste inn en arme fra filer, dersom de måtte ønske det. Da vil en ferdig lagd arme dukke opp i listen din. Dette går mye raskere enn å opprette enheter manuelt. Brukeren av programmet har også muligheten til å slette en enhet fra armeen sin ved å velge den fra listen og deretter trykke på knappen for å slette en enhet fra arme.

Etter hvert som du oppretter armeen din har du muligheten til å se flere detaljer av armeen som for eksempel hvor mange enheter det totalt er i armeen eller hvor mange enheter det er av en spesifikk enhetstype. Dette gjøres ved å trykke på «View army details»-knappen. Da vil et pop-up vindu vises som vil vise flere detaljer om armeen din.

Etter at du har opprettet armeen din går du videre til scenen hvor simuleringen av slaget skjer. Først må man velge en fiendtlig arme å kjempe mot. Denne armeen velger man ved å trykke på et av de tre «load» knappene helt til høyre i scenen. Hver av disse vil laste inn en unik

arme fra en fil, som brukeren er ment å kjempe imot. For å starte simulering så må man velge et terreng, hvor slaget skal foregå. De ulike enhetene har forskjellige fordeler og ulemper i de forskjellige terrengene, så hvilket terreng man velger kan være avgjørende for hvilken arme som vinner slaget.

Etter at slaget er over kan man trykke på knappen «View battle results» i scenen som vil vise deg resultatet av slaget. Du får se hvem som vant slaget og andre detaljer om de to armeene, som for eksempel hvor mange enheter de var igjen i hvert av armeene etter krigen og hvor mye helse de hadde igjen. Så lenge det er to armeer i tabellene, så kan du kjøre simuleringen om igjen og igjen og få forskjellig resultat for hver kamp. Dersom du ønsker å kjempe mot en ny fiendtlig arme, er det bare å trykke på en av de andre «load»-knappene, og kjøre simulering på nytt. Merk at hvis du bytter terreng ofte og kjører simulering på nytt så vil applikasjonen av og til fryse opp.

2.1 Use-Case diagram

Jeg har inkludert med et «use case» – diagram for å gi en grafisk fremstilling av hvilke muligheter brukeren har av interaksjoner med applikasjonen.



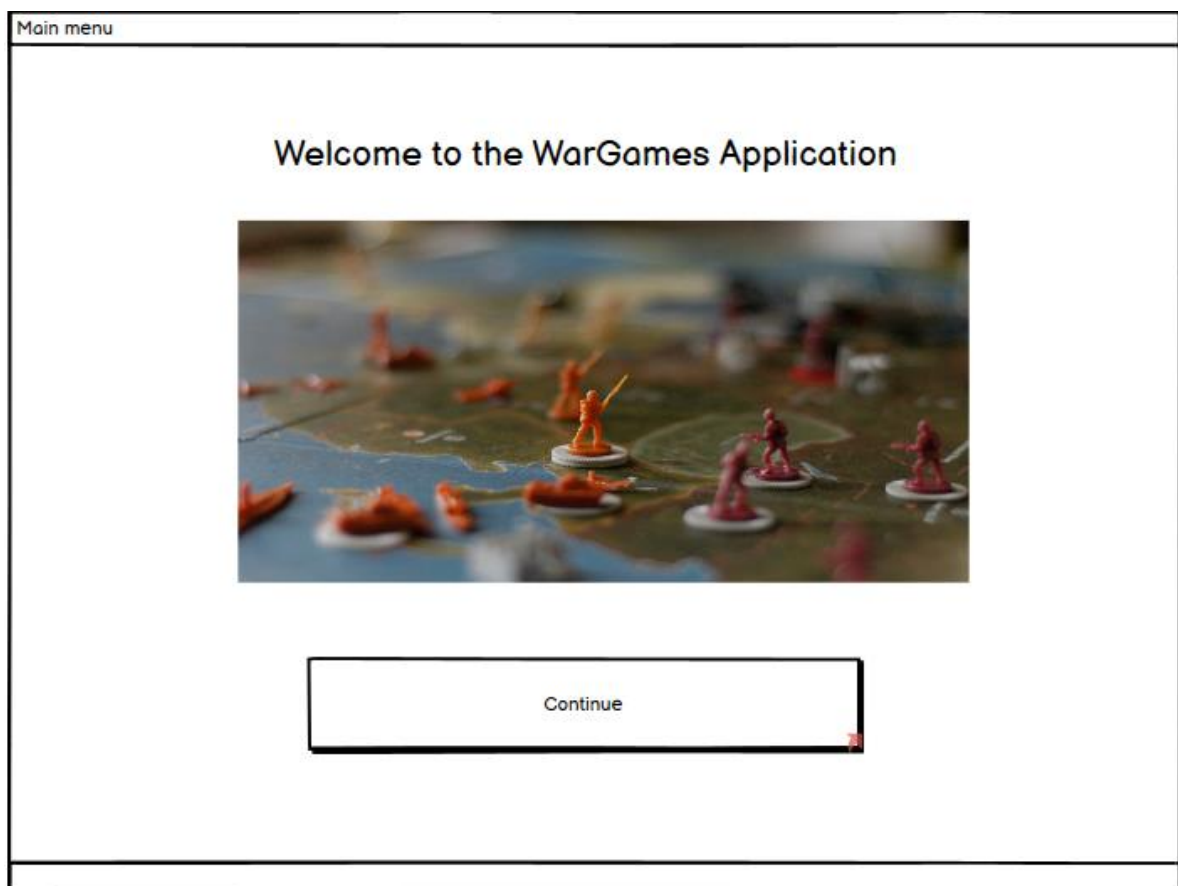
Figur: Use-case diagram av applikasjonen

3. Design

I del 2 av prosjektet fikk vi i oppgave å lage en eller flere skisser for hvordan det grafiske brukergrensesnittet skulle se ut. Skissene skulle illustrere layout og utseende til det grafiske brukergrensesnittet i wargames-applikasjonen. For å få til dette brukte jeg verktøyet *Balsamiq Wireframes*, hvor man enkelt kunne lage slike skisser. Her er skjermbilder av skissene som ble lagd i Balsamiq Wireframes:

3.1 Balsamiq Wireframes

3.1.1 Main menu



Figur: Skisse av hovedmenyen av applikasjonen


3.1.2 Army Registration

Army registration

Register Armies

Army 1


Write name of army:

Add units to army: 

Vs.

Army 2

Write name of army:

Add units to army: 

Figur: Skisse av arme registrering vindu

3.1.3 Unit registration

Unit registration

Register Units to "Army name"

Choose type of unit:

Unit Types

Infantry Unit

Ranged Unit

Cavalry Unit

Commander Unit

Enter name of unit:

Total number of units in army:

Figur: Skisse av enhet registrering vindu

3.1.4 Army details

Army Details

Army Details

Unit Type	Name	Health	Attack	Armor
Infantry Unit	ho	100	40	60
Cavalry Unit	ga	100	35	50
RangedUnit	frf	80	30	30
Cavalry Unit	ong	100	35	60
Commander Unit	lmo	120	50	70

Unit type	Number of units
Infantry Unit	1
Cavalry Unit	2
RangedUnit	1
Commander Unit	1

Total number of units in army:

< Back to Army Registration

Figur: Skisse av arme detaljer vindu

3.1.5 Load army from file

Load Army from file

Load Army from file

Write in name of the file

Unit Type	Name	Health	Attack	Armor
Infantry Unit	ho	100	40	60
Cavalry Unit	ga	100	35	50
RangedUnit	frf	80	30	30
Cavalry Unit	ong	100	35	60
Commander Unit	lmao	120	50	70

Total number of units in army:

< Back to Army Registration

Figur: Skisse av filhåndterings vindu

3.1.6 Battle simulation

Battle simulation

Start battle

Army: name

Vs.

Army: name

Unit Type	Name	Health	Attack	Armor
Infantry Unit	ho	100	40	60
Cavalry Unit	gae	100	35	50
RangedUnit	frfr	80	30	30
Cavalry Unit	ong	100	35	60

Unit Type	Name	Health	Attack	Armor
Infantry Unit	ho	100	40	60
Cavalry Unit	gae	100	35	50
RangedUnit	frfr	80	30	30
Cavalry Unit	ong	100	35	60

Choose terrain of the battlefield:

Terrain

start simulation

reset

Back to main menu

Figur: Skisse av kamp simulasjons vindu

3.2 Designprinsipper og designmønstre

Brukergrensesnittet er konstruert og bygget i tråd med Don Normans prinsipper om design. Don Normans er en kjent forsker innenfor emnet menneske-maskin-interaksjon, og han har delt flere prinsipper på hva man bør fokusere på for å oppnå god menneske-maskin-interaksjon når man designer et brukergrensesnitt.

Jeg har fulgt prinsippet om synlighet (visibility), som går ut på at brukere av en applikasjon bør kunne intuitivt vite hvordan man kan få tilgang til den informasjonen man leter etter og hvilke muligheter man har i programmet. Det bør ikke være mer informasjon enn nødvendig på et vindu i et brukergrensesnitt, ettersom dette kan forvirre brukeren ved å gi dem alt for mye informasjon på en gang. Det går imot prinsippet om synlighet.

Jeg har også fulgt prinsippet om tilbakemeldinger (feedback), som går ut på at brukere av en applikasjon bør få tilbakemelding etter at de har utført en handling i programmet. Det er viktig for brukeren å få vite om handling de utførte ble registrert eller ikke. Dette blir gjort i programmet gjennom dialog bokser.

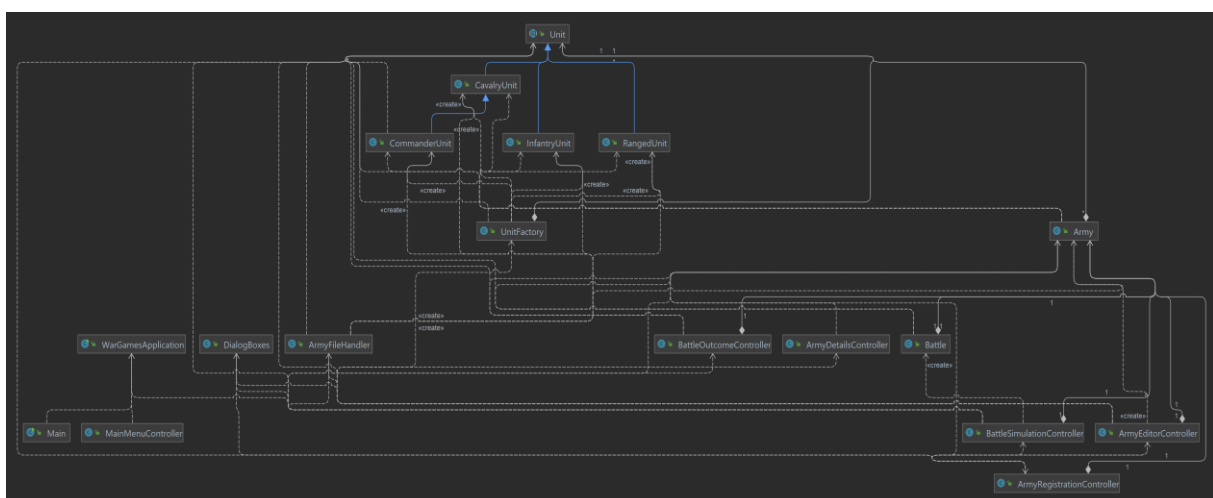
Samtidig som jeg har gitt brukeren stor frihet, har jeg også satt noen begrensninger på hva brukeren får lov til. Dette omhandler også et av Don Normans prinsipper om design. Det er viktig å ha en begrensning på hvor mye informasjon du gir brukeren på et enkelt vindu, og hvor mange muligheter de har i det vinduet. Å begrense brukerens muligheter er nødvendig for at brukeren selv skal kunne forstå når de burde gå videre til neste vindu. Under utvikling av applikasjonen hadde jeg derfor fokus på å ikke ha altfor mye informasjon eller knapper på ett vindu. Dersom brukeren ville ønske mer informasjon, som for eksempel i vinduet hvor man kan redigere en arme, så implementerte jeg heller ett pop-up vindu som kunne vise den informasjonen som ikke var på det andre vinduet istedet for å ha alt informasjonen om en arme på et og samme vindu. Dette virket mer effektivt og ble en mye ryddigere måte å vise informasjon på.

Et designmønstre som jeg har brukt i applikasjonen er fabrikk. UnitFactory klassen bruker design mønsteret fabrikk, og mønsteret går ut på at du definerer en abstrakt for å lage objekter, men lar subclassene av denne abstrakte klassen bestemme hvilken klasse som skal instansers. Subklassene er altså ansvarlig for å opprette instanser av klasser. Et annet designmønster jeg har brukt er model-view-controller (MVC), dette skal vi se nærmere på i neste kapittel.

3.3 Klassediagram

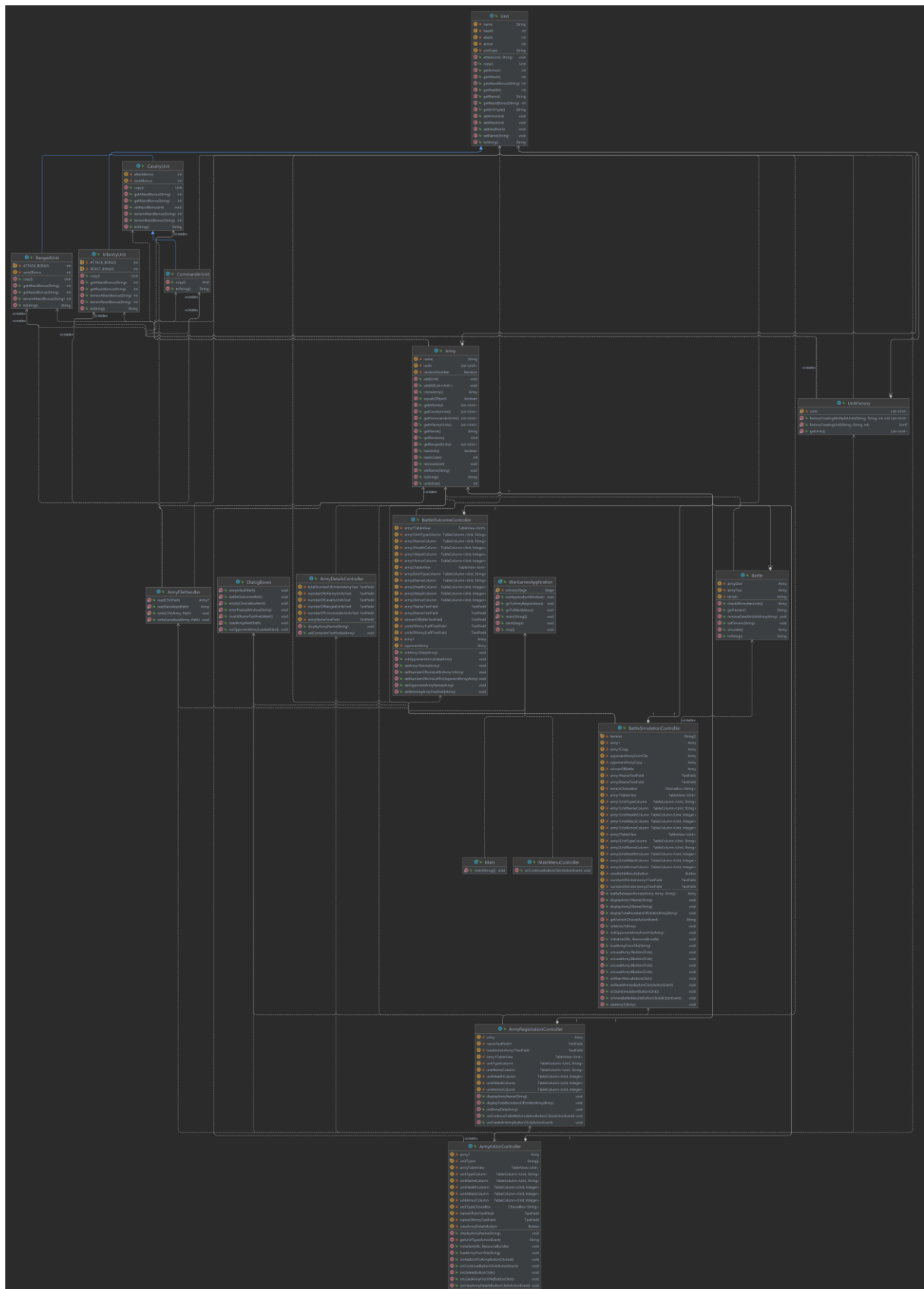
Etter å ha implementert ulike designprinsippene og designmønstre i applikasjonen, endte arkitekturløsningen med å se slik ut:

3.3.1 Klassediagram 1



Figur: Klassediagram som kun viser avhengigheter (dependencies)

3.3.2 Klassediagramm 2



Figur: Fullstendig klassesdiagram med avhengigheter, felt og metoder

4. Implementasjon

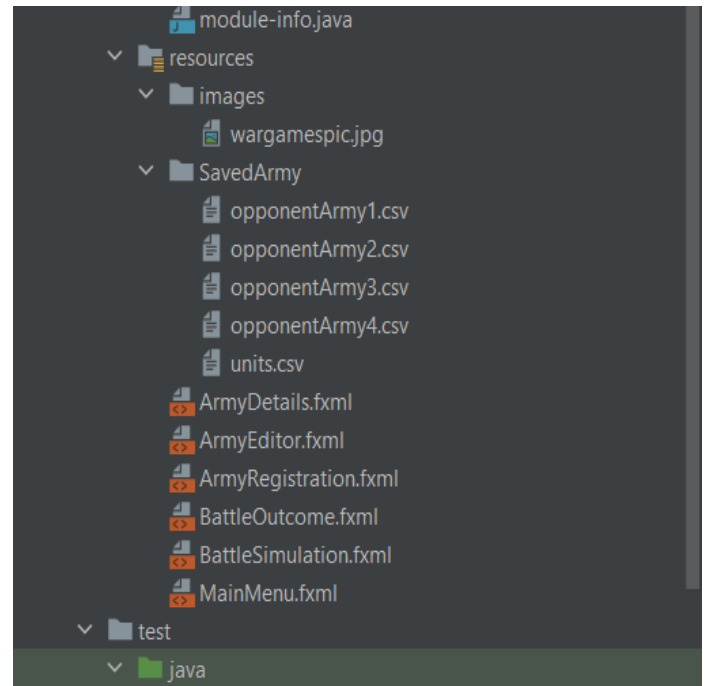
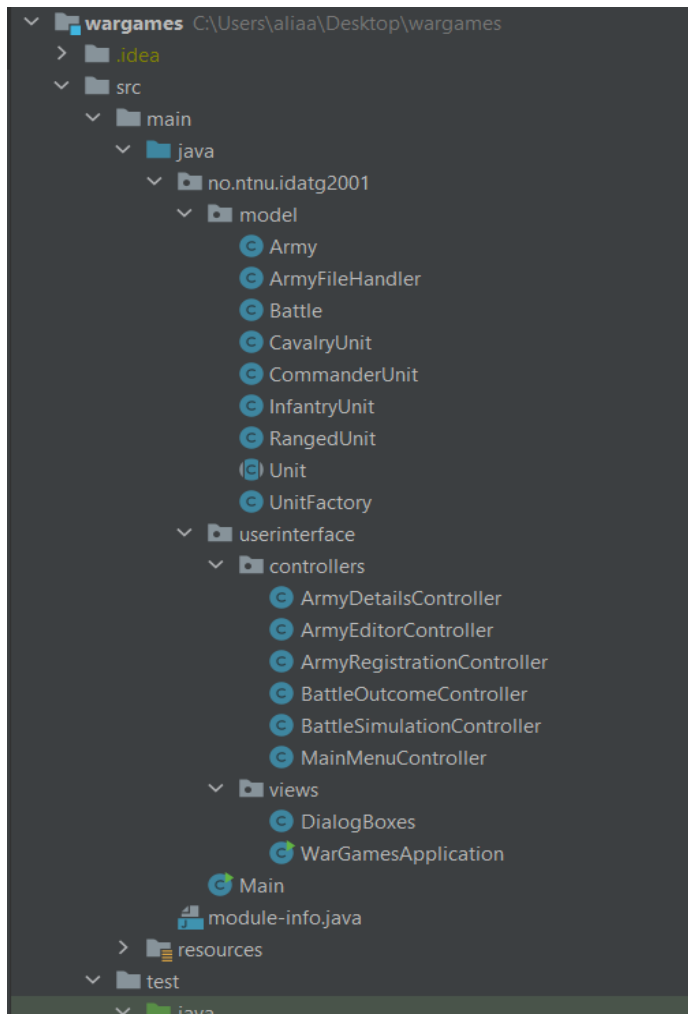
Programutviklere bruker ulike designmønstre for å strukturere prosjektene deres slik at den blir ryddigere og lettere å forstå. I utvikling av denne applikasjonen har jeg brukt designmønsteret ***Model-View-Controller*** (MVC). MVC er et designmønster som deler applikasjonen inn i tre komponenter; data (*model*), brukergrensesnittet (*view*), og et mellomliggende komponent som brukes for å kommunisere mellom data og brukergrensesnitt, kalt *controller*.

Implementasjonen av designmønsteret fører til koden blir ryddig og systematisk strukturert. For å oppnå dette designmønsteret lagde jeg forskjellige pakker som jeg har puttet klassene inn i. Modellkomponenten er hvor dataen befinner seg og den holder på all data i applikasjonen. Den tilsvarende kjernekode og all datarelatert logikk som brukeren arbeider med. Denne dataen finner man i pakken kalt *model*. Alle klasser som omhandler enheter, armeer, lese armeer fra filer og slaget mellom armeer, ligger i denne pakken.

Brukergransesnitt- og kontrollerkomponenten er samlet i pakken *userinterface*. Her ligger alle klasser som omhandler brukergrensesnittet av applikasjonen. Pakken *userinterface* består av pakkene *controllers* og *views*. I *controllers* pakken så ligger alle kontroller-klassene til java-FXML filene. FXML-filene finner man under *resources*, og hver eneste FXML-fil har sin egen kontroller klasse. Kontroller klassene sørger for å flytte data mellom modellkomponenten og brukergrensesnitt.

Under *resources* finner man også en katalog kalt *SavedArmy* som inneholder filer med armeer som man kan skrive ut og bruke i simulasjonen.

4.1 Model-View-Controller designmønster

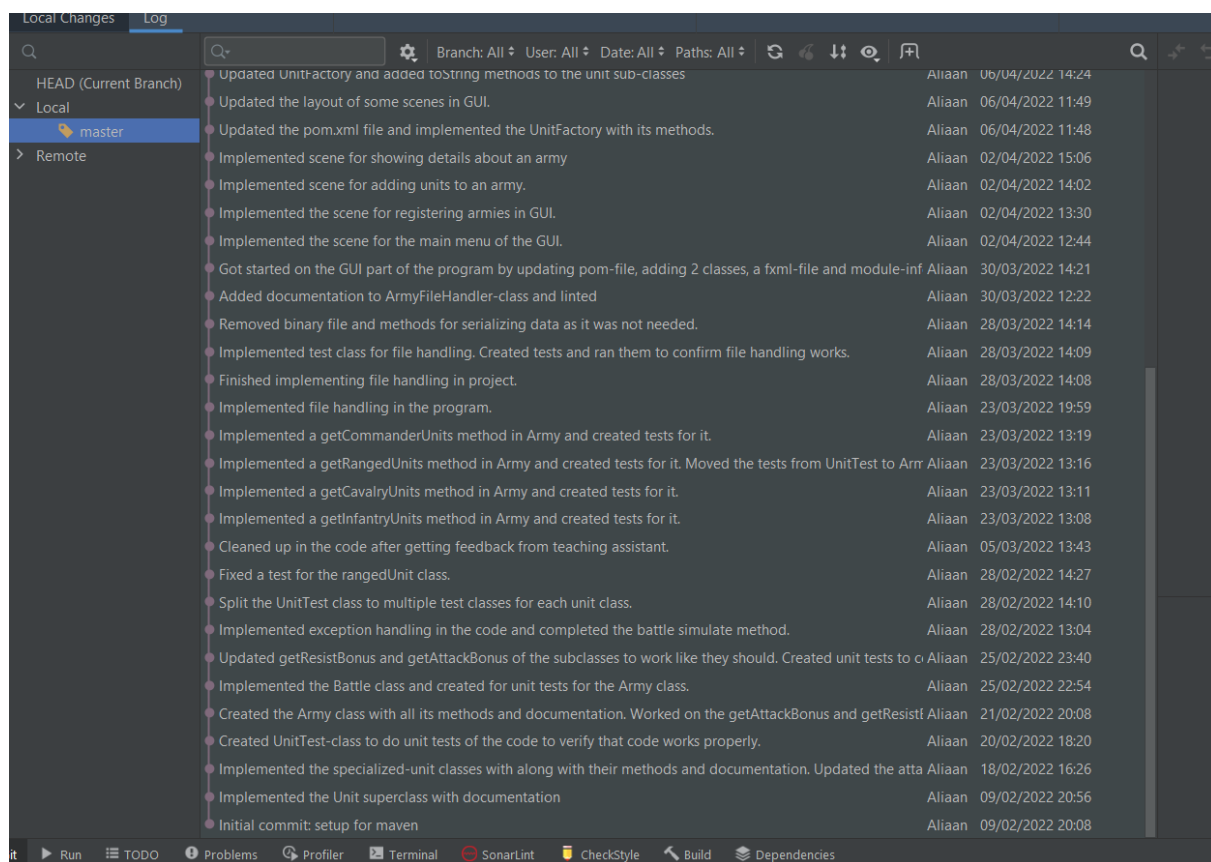
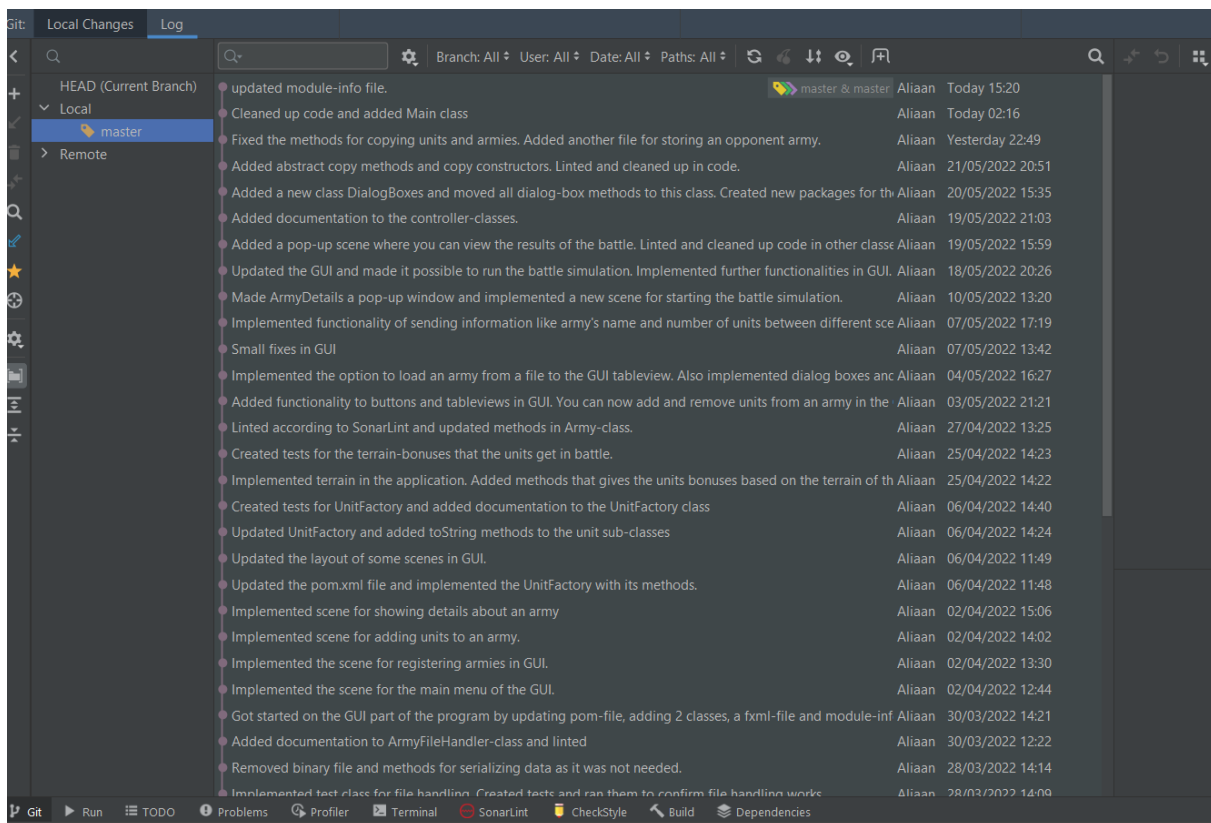


Figur: Struktur til prosjektet

5. Prosess

Gjennom semesteret har jeg arbeidet jevnt og systematisk med prosjektet. Etter hver forelesning i emnet hvor vi lærte noe nytt, så sørget jeg for å jobbe videre på prosjektet og implementere det nye jeg hadde lært inn i prosjektet. Dersom det var noe jeg slet med å implementere eller noe som jeg ikke forsto, så tok jeg kontakt med studentassistentene eller lærer og spurte dem om hjelp. Jeg har brukt verktøy som versjonskontroll i git og husket å «commite» endringer etter å ha arbeidet videre på prosjektet. Dette var et nyttig verktøy som hjalp med å holde styr på ulike versjoner av prosjektet mitt, og spore progresjon og fremgang. Jeg har også brukt verktøy som *visual paradigm* for å illustrere «use case»-diagrammer og *Balsamiq Wireframes* for å skissere det grafiske brukergrensesnittet til applikasjonen.

5.1 Commits



Figur: Skjermbilde av «commit»-meldinger

6. Refleksjon

Jeg er fornøyd med arbeidet mitt i dette prosjektet. Jeg har lært utrolig mye mer om programmering dette semesteret og wargames-prosjektet har vært veldig gøy å jobbe med. Jeg har lært om maven, arv, polymorfi, filhåndtering, streams, lambda, unntakshåndtering, designmønstre, hvordan bygge og designe et grafisk brukergrensesnitt og mer. Alt jeg har lært gjennom semesteret har kommet til god nytte i arbeidet med prosjektet.

Noe jeg ville ha gjort annerledes er å titte nærmere på kravene på prosjektet. Jeg fikk inntrykket av at brukeren skulle kunne lage egne armeer fra bunnen av og bruke disse i kamp. Jeg brukte mye tid på å prøve å få til å opprette to armeer fra bunn, men forsto etter hvert at dette ikke et krav. Jeg hadde allerede klart å implementere oppretting av én arme fra bunn så jeg bare fortsatte videre med det siden det falt likevel under noe man kunne implementere i oppgave 4 i del 3 som handlet om videre arbeid av prosjektet.

Et problem oppsto hos meg når jeg prøvde å kjøre applikasjonen fra Maven via plugin javafx -> javafx:run, hvor jeg alltid ville få en feilmelding. Prosjektet ble sendt til professoren og andre medstudenter for å teste om problemet ville oppstå hos dem, men det gjorde det ikke. Applikasjonen ble testet fra flere andre datamaskiner enn min og kjørte uten problemer for dem. Jeg antar derfor at det må være en «bug» i mitt system, som ikke lar meg kjøre javafx:run, men det skal funke for andre.

7. Konklusjon

For å konkludere så fikk jeg i oppgave å utvikle et program kalt wargames i emnet IDATG2001. Wargames prosjektet går ut på at man skal utvikle et program som skal simulere et slag mellom to armeer. For å få til dette så har jeg jobbet jevnt og systematisk gjennom semesteret, og implementert all kunnskapen min om programmering inn i prosjektet.