Wargames - Del 3

I denne siste delen av Wargames-prosjektet skal du anvende designmønster, innføre terreng som en ekstra variabel i bonusberegningen til enhetene, samt implementere et grafisk brukergrensesnittet basert på skissene fra del 2. I tillegg får du muligheten til å utvide programmet basert på egne ideer. Du skal ta utgangspunkt i din egen kode fra del 2 når du løser oppgavene.

Det kan være lurt å lese gjennom hele dokumentet før du begynner.

Før du begynner

Følgende krav og betingelser gjelder for alle oppgavene:

Enhetstesting

Du må lage enhetstester for den delen av koden som er forretningskritisk, altså for den koden som er viktigst for å oppfylle sentrale krav. Feil her vil få store negative konsekvenser for programmet. GUI-testing er ikke særskilt tema i faget, og det er derfor ikke nødvendig å utvikle enhetstester direkte knyttet til brukergrensesnittet.

Unntakshåndtering

Uønskede hendelser og tilstander som forstyrrer normal flyt skal håndteres på en god måte.

Maven

Prosjektet skal være et Maven-prosjekt med en fornuftig groupId og artefactId, og følge JDK v11 eller høyere. Filer skal lagres iht standard Maven-oppsett. Det skal være mulig å bygge, teste og kjøre med Maven uten feil.

Versjonskontroll

Koden skal være lagt under versjonskontroll og koblet mot et sentralt repo. Sjekk inn og push til sentralt repo med jevne mellomrom. Som vanlig forventer vi at commit-meldingene beskriver endringene som er gjort på en kort og konsis måte.

Oppgave 1: UnitFactory

Anvend Factory design pattern og lag en fabrikk med to metoder:

- Den første metoden skal opprette en Unit basert på type enhet, navn og helseverdi.
- Den andre metoden skal returnere en liste med n antall Units basert på type enhet, navn og helseverdi. Metoden skal kunne brukes når vi trenger mange enheter av samme type (enhetene vil naturlig nok få samme navn og helseverdi).

Refaktorer koden din slik at fabrikken benyttes der det er hensiktsmessig.

Oppgave 2: Terrain

I denne oppgaven skal du innføre terreng (Terrain) som variabel i bonusberegningen til enhetene. Et slag (Battle) skal fra nå av foregå i et terreng. Simuleringen må støtte tre typer terreng: ås (HILL), slette (PLAINS) og skog (FOREST). Enhetstypene må ta hensyn til terrenget når bonus beregnes.

InfantryUnit:

- Enheten har en ekstra fordel når den kjemper i skog (FOREST).
- Enheten har denne fordelen både i angrep og i forsvar.

RangedUnit:

- Enheten har en ekstra fordel når den angriper fra en ås (HILL).
- Når enheten angriper i en skog (FOREST) skal angrepsbonus være mindre enn tidligere. Enheten får fortsatt bonus fordi den er smidig, men mister litt fordi den ikke kan angripe fra avstand i en skog.
- Terreng påvirker ikke forsvarsbonusen til denne enheten.

CavalryUnit:

- Enheten har en ekstra fordel når den angriper på en slette (PLAINS).
- Når enheten blir angrepet i en skog (FOREST) skal total forsvarsbonus nå være 0 (altså ingen fordel).

Oppgave 3: GUI

I del 2 av prosjektet laget du en eller flere skisser for et grafisk brukergrensesnitt. Du skal nå kode brukergrensesnittet i JavaFX. Du kan selv velge om du vil bruke FXML eller kode alt manuelt (ren Javakode). Kravene fra GUI-oppgaven i del 2 gjelder naturligvis fortsatt, men som et *tilleggskrav* må det også være mulig å velge terreng før man kjører simuleringen.

Oppgave 4: Videre arbeid

Frem til nå har du jobbet utifra veldefinerte oppgaver og krav. Underveis har du kanskje fått egne ideer. I denne oppgaven oppfordrer vi deg til å være kreativ og gå utenfor de begrensningene vi har satt hittil. Det er du som bestemmer, men her er noen forslag:

- Flere enhetstyper: hva med en enhet som har magiske krefter? Eller kanskje enda mer spesialiserte typer av de vi allerede har?
- Filhåndtering: kunne man hatt et enklere, mer komprimert format? Burde man hatt støtte for flere formater?
- Kan man innføre andre variabler i simuleringen? Hva om enhetene har en posisjon i et koordinatsystem?
- Bør det være mulig å legge til, endre og slette enheter direkte fra brukergrensesnittet?
- Hva med støtte for flere språk (i18n)?

Viktige sjekkpunkter

Følgende momenter bør dobbeltsjekkes:

- Maven:
    - Er prosjektet et Maven-prosjekt med fornuftige prosjekt-verdier og gyldig katalogstruktur?
    - Kan man kjøre Maven-kommandoer for å bygge, teste, installere og kjøre uten at det feiler?
- Versjonskontroll med git:
    - Er prosjektet underlagt versjonskontroll med lokalt repo?
    - Er det lokale repoet koblet mot et sentralt repo?
    - Er koden sjekket inn med jevne mellomrom og pushet til sentralt repo?
    - Beskriver commit-meldingene endringene på en kort og konsis måte?
- Enhetstester:
    - Har enhetstestene beskrivende navn som dokumenterer hva testene gjør?
    - Følger de mønsteret Arrange-Act-Assert?
    - Tas det hensyn til både positive og negative tilfeller?
- Er fabrikken implementert iht factory design pattern og oppgavebeskrivelsen forøvrig?
- Tar simulering og bonusberegning hensyn til terreng iht kravene i oppgaven?
- GUI:
    - Presenteres arméene på en oversiktlig måte og kan man se detaljert info iht krav og betingelser?
    - Er det mulig å laste inn arméene fra filer og oppgis filnavn/filsti?
    - Kan man velge terreng?
    - Fungerer simuleringen som forventet?
    - Blir man opplyst om det endelige resultatet og tilstanden til de to arméene i etterkant?
    - Er det mulig å resette arméene for ny simulering uten å måtte oppgi fil-lokasjoner på nytt?
    - Er løsningen brukervennlig (oversiktlig og responsiv layout, god kontrast, fornuftig fargevalg og typografi, forståelige feilmeldinger osv)?
- Kodekvalitet:
    - Er koden godt dokumentert iht JavaDoc-standard?
    - Er koden robust (validering, unntakshåndtering mm)?
    - Har variabler, metoder og klasser beskrivende navn?
    - Er klassene gruppert i en logisk pakkestruktur?
    - Har koden god struktur, med løse koblinger og høy kohesjon?