COMPUTERSPIL 1

I løbet af de kommende fem uger skal I programmere et computerspil, hvor spillerne rejser rundt mellem byer i forskellige lande og indsamler point. Der er en delaflevering hver uge, hvor I tilføjer nye ting til spillet – ofte baseret på ting, der er gennemgået i de nærmest foregående forelæsninger (og kapitler i BlueJ-bogen).

I denne første delaflevering skal I bl.a. modellere byerne og vejene imellem dem, samt de lande byerne ligger i. Når I er færdige, vil I have den første version af et computerspil, hvor den grafiske brugergrænseflade viser et netværk af udvalgte byer i Norden og vejene imellem dem.

Når I laver de forskellige klasser, skal I samtidig dokumentere dem, således at dokumentationen er helt i top og følger:

- retningslinjerne fra BlueJ bogen (herunder specielt afsnit 6.11, afsnit 9.7 og Appendix I),
- de eksempler fra ***Road*** klassen, der blev præsenteret i en forelæsning.

Hver klasse og hver metode/konstruktør skal have en passende kommentar (der begynder med /** og slutter med */). Tænk på, at første sætning indsættes i dokumentationens *Method/Constructor Summary*, mens hele kommentaren indsættes i dokumentationens *Method/Constructor Detail*. I skal også indsætte *@author* og *@version* tags i jeres klasser, samt *@param* og *@return* tags i jeres metoder/konstruktører. For at undgå for mange sprogskift, anbefales det, at alle jeres kommentarer skrives på Engelsk (eller Amerikansk). Skift jævnligt til *Documentation view* for at kontrollere, at resultatet er fornuftigt og giver relevant og letlæselig information til brugere, der ikke kender implementationen af jeres klasser.

Der er forholdsvis meget at lave i Computerspil 1, men langt de fleste af tingene burde på nuværende tidspunkt være lette at implementere.

Download BlueJ projektet ***Nordic Traveller*** (zip). Projektet indeholder klassen ***TestServer***, der bl.a. har nedenstående to klassemetoder:

- ***test*** metoden uploader jeres projekt til vores testserver. Ved afslutningen af hver opgave, kan I afprøve, om det, som I har lavet i opgaven, fungerer korrekt ved at kalde ***test*** med parameteren "CG1-X", hvor X er nummeret på den pågældende opgave. Testserveren afprøver konstruktøren og metoderne i opgave X ved hjælp af nogle regression tests, som vi har skrevet. Men det er ikke altid muligt at gentage gamle tests. Når man f.eks. i opgave 5 ændrer City klassens konstruktør, så den tager en ekstra parameter, kan man ikke længere køre den gamle test fra opgave 1 uden at denne fejler.
- ***download*** metoden skal I først bruge i opgave 7, hvor den vil downloade nogle nye klasser til jeres projekt. Hvis I kalder den tidligere, vil I ikke kunne oversætte jeres projekt, idet de nye klasser bruger nogle af de ting, som I endnu ikke har skrevet.

Der vil altid være små fejl i så kompliceret en opgave som computerspillet – det gælder både opgaveformulering, udleverede klasser og testserveren. Brug webboardet til at rapportere eventuelle fejl/problemer, som I måtte støde på. Så retter vi dem hurtigst muligt og uploader nye versioner, når det er nødvendigt. Af samme grund er det vigtigt, at I alle følger med på webboardet, således at I bliver opmærksomme på nye versioner eller hints til at omgå problemer.

Når I får en fejlrapport, bør I rette **alle** de fejl, der rapporteres og teste, at rettelserne er korrekte, før I atter forsøger at køre testserveren Hvis I blot retter en enkelt fejl ad gangen (uden selv at teste om rettelsen fungerer) kommer I let til at bruge alt for megen tid på at vente på, at testserveren genererer rapporter til jer (specielt hvis der er kø).

Opgave 1

Implementér ***City*** klassen, der repræsenterer en by, som man kan besøge. En by har et navn, der er repræsenteret ved en tekststreng, samt en værdi, der er repræsenteret ved et heltal.

Opret ***City*** klassen med tre feltvariabler ***name***, ***value*** og ***initialValue***.

Klassen skal have accessor metoder for alle tre feltvariabler og en konstruktør på formen:

- ***City(String name, int value)***

hvor den første parameter bruges til at initialisere feltvariablen ***name***, mens den anden parameter bruges til at initialisere de to sidste feltvariabler.

Herudover skal klassen have nedenstående metoder:

- ***void changeValue(int amount)*** // Adderer ***amount*** til ***value***
- ***void reset()*** // Resetter ***value*** til ***initialValue***
- ***String toString()***

Den sidste metode returnerer en tekststreng på formen "London (60)", hvor London er byens navn og 60 dens værdi.

Klassen skal implementere interfacet ***Comparable<City>***, dvs. at I skal lave en ***compareTo*** metode, der definerer byernes sorteringsrækkefølge (alfabetisk efter navn).

Til sidst skal I implementere ***equals*** og ***hashCode*** metoder for ***City*** klassen, idet I anvender de principper, der blev gennemgået i en forelæsning. To ***City*** objekter er ens, hvis og kun hvis de har samme navn.

Husk at bruge *@Override*, når I overskriver eksisterende metoder, såsom ***toString***, ***equals*** og ***hashCode***.

Opgave 2

Implementér ***Road*** klassen, der repræsenterer en vej mellem to byer. En vej har referencer til de to byer (***City*** objekter) den forbinder. Den har også en længde, der angiver, hvor langt der er mellem de to byer.

Opret klassen ***Road*** med tre feltvariabler ***from***, ***to*** og ***length***.

Klassen skal have accessor metoder for alle tre feltvariabler og en konstruktør på formen:

- ***Road(City from, City to, int length)***

Herudover skal klassen have nedenstående metode:

- ***String toString()***

der returnerer en tekststreng på formen "London (60) -> Liverpool (40) : 8", hvor 8 er afstanden mellem de to byer.

Klassen skal implementere interfacet ***Comparable<Road>***, dvs. at I skal lave en ***compareTo*** metode, der definerer vejenes sorteringsrækkefølge. Først sorteres efter de byer, hvor vejene starter. Hvis to veje starter i samme by sorteres efter de byer, hvor vejene slutter og dernæst efter længden (lavest til højest).

Til sidst skal I implementere ***equals*** og ***hashCode*** metoder for ***Road*** klassen. To ***Road*** objekter er ens, hvis og kun hvis de tre feltvariabler har ens værdier.

Opgave 3

Implementér ***Position*** klassen, der repræsenterer en spillers position. Klassen har fire feltvariabler:

- ***from*** // Den by spilleren kommer fra
- ***to*** // Den by spilleren er på vej til
- ***distance*** // Den afstand der er tilbage
- ***total*** // Den totale afstand mellem de to byer

Klassen skal have accessor metoder for alle fire feltvariabler og en konstruktør på formen:

- ***public Position(City from, City to, int distance)***

hvor parameteren ***distance*** også bruges til at initialisere feltvariablen ***total***.

Herudover skal klassen have nedenstående metoder:

- ***boolean move()*** // Tæller ***distance*** 1 ned hvis ***distance > 0***
  // Returværdien angiver om ***distance*** ændres
- ***void turnAround()*** // Vender spillerens bevægelsesretning uden at spilleren flyttes
- ***boolean hasArrived()*** // Returnerer ***(distance == 0)***
- ***String toString()***

Den sidste metode returnerer en tekststreng på formen "London (60) -> Liverpool (40) : 3/8", hvor 3 er den afstand, der mangler, mens 8 er den totale afstanden mellem de to byer.

Til sidst skal I implementere ***equals*** og ***hashCode*** metoder for ***Position*** klassen. To ***Position*** objekter er ens, hvis og kun hvis de fire feltvariabler har ens værdier.

Opgave 4

Implementér ***Country*** klassen, der repræsenterer et land med et netværk af byer og veje. Klassen har to feltvariabler:

- ***name*** // Landets navn
- ***network*** // Landets veje repræsenteret som en ***Map<City, Set<Road>>***

Netværket (network) er en afbildning (map), hvor hver by tilknyttes en mængde indeholdende alle de veje, der starter i den pågældende by.

Klassen skal have en accessor metode for den første feltvariabel og en konstruktør på formen:

- ***Country(String name)***

hvor ***network*** initialiseres til at være en tom afbildning (map) ved hjælp af følgende udtryk ***new TreeMap<>()***. Bemærk, at typen for feltvariablen ***network*** og signaturerne for nedenstående metoder bruger typerne ***Map*** og ***Set***, der er interfaces. Vi bruger kun konkrete klasser, når vi instantierer et objekt via ***new*** operatoren.

Landet skal have nedenstående metoder:

- ***Set<City> getCities()*** // Returnerer landets byer
- ***City getCity(String name)*** // Returnerer byen med det angivne navn
- ***Set<Road> getRoads(City c)*** // Returnerer alle veje, der starter i byen c
- ***void reset()*** // Resetter alle landets byer
- ***String toString()*** // Returnerer landets navn

I den første metode kan man bruge en af ***Map*** interfacets metoder til at finde alle de byer, der forekommer som nøgler i netværket.

Den anden metode returnere ***null***, hvis der ikke findes en by i netværket med det angivne navn. Den kan implementeres ved hjælp af en af algoritmeskabelonerne, hvor man dog nu gennemløber en mængde i stedet for en liste.

Den tredje metode returnerer en tom mængde ***new TreeSet<>()***, hvis den angivne by ikke findes som nøgle i ***network***. Dette kan afgøres ved hjælp af en af ***Map*** interfaces metoder. Ellers returneres den mængde af veje, som er tilknyttet nøglen (byen).

Til sidst skal I implementere ***equals*** og ***hashCode*** metoder for ***Country*** klassen. To ***Country*** objekter er ens, hvis og kun hvis de har samme navn.

Opgave 5

Når en spiller ankommer til en by, modtager hun en bonus, der er en del af byens værdi (som samtidigt formindskes tilsvarende). Bonussens størrelse udregnes i ***Country*** klassen, idet vi senere vil introducere lande, hvor udregningen foretages på en anden måde.

Hver by skal derfor have en feltvariabel ***country***, der fortæller, hvilket land byen ligger i. Introducér denne feltvariabel (sammen med en accessor metode ***getCountry***) og tilføj en parameter til konstruktøren for ***City*** klassen, således at man kan angive, hvilket land en by ligger i:

- ***City(String name, int value, Country country)***

Modificér ***equals*** og ***hashCode*** metoderne i ***City*** klassen, således at de anvender den nye feltvariabel. To ***City*** objekter er ens, hvis og kun hvis de har samme navn og ligger i samme land.

I ***Country*** klassen implementeres nedenstående metode:

- ***int bonus(int value)*** // Beregner størrelsen af den bonus, som spilleren modtager

Hvis parameterværdien er positiv, returneres et tilfældigt heltal i intervallet ***[0, value]***. Ellers returneres ***0***.

Testserveren forudsætter, at I ikke laver unødvendige kald af metoderne i ***Random*** objektet. Hvis f.eks. ***bonus*** metoden laver to kald af ***nextInt*** for a beregne bonussen, vil de pseudo-tilfældige tal komme ud af sync, og testen vil fejle.

I ***City*** klassen implementeres en metode:

- ***int arrive()*** // Fratrækker den udbetalte bonus fra byens værdi

der kaldes af ***Game*** klassen (der introduceres senere), hver gang en spiller ankommer til en by. Metoden kalder ***bonus*** metoden i det land, som byen tilhører (med en parameterværdi, der er lig med byens værdi). Den værdi, som ***bonus*** metoden returnerer, fratrækkes fra byens værdi og returneres som ***arrive*** metodens returværdi.

Opgave 6

Implementér nedenstående metoder i ***Country*** klassen:

- ***void addCity(City c)*** // Tilføjer en by til netværket
- ***void addRoads(City a, City b, int length)*** // Tilføjer veje mellem a og b

Den første metode indsætter den angivne by som nøgle i den afbildning som feltvariablen ***network*** peger på. Den tilknyttede værdi er en tom mængde ***new TreeSet<>()***.

Den anden metode opretter veje mellem de angivne byer, ved at indsætte dem i de mængder, der forekommer som værdier i den afbildning, som feltvariablen ***network*** peger på. Der oprettes kun veje der starter i det pågældende land. Hvis begge byer ligger i landet oprettes der veje i begge retninger. Hvis én af byerne ligger i landet oprettes kun den vej, der starter i landet. Hvis ingen af

byerne ligger i landet oprettes der ingen veje. Der oprettes kun veje, der har positiv længde og forbinder to forskellige byer (afgjort ved hjælp af ***equals***).

Implementér nedenstående metoder i ***Country*** klassen:

- ***Position position(City city)*** // Returnerer byens position
- ***Position readyToTravel(City from, City to)*** // Gør klar til at påbegynde rejse

Den første metode returnerer byens position, dvs. et ***Position*** objekt på formen ***new Position(city, city, 0)*** – svarende til at spilleren står i byen ***city*** uden at have valgt en vej mod en anden by. Dette er også tilfældet, hvis byen slet ikke ligger i landet.

Den anden metode returnerer et ***Position*** objekt, der repræsenterer, at spilleren er klar til at påbegynde rejsen fra ***from*** til ***to*** (men stadig står i ***from***). Hvis ***from*** og ***to*** er samme by (afgjort ved hjælp af ***equals***), returneres byens position. Hvis der ikke er en direkte vej fra ***from*** til ***to*** returneres ***from***'s position. Dette er også tilfældet, hvis ***from*** slet ikke ligger i landet.

Opgave 7

Kald klassemetoden ***download*** i ***TestServer*** klassen med parameteren "CG1". Dette vil downloade nedestående klasser og tilføje dem til jeres projekt. Mange af de nye klasser har oversætterfejl. Det skal I ikke blive forskrækket over. Det retter sig, når I laver de to rettelser, der er beskrevet nederst i opgave 7.

- ***Game*** styrer interaktionen mellem spillere, byer og lande.
- ***GUI*** indeholder den grafiske brugergrænseflade (hvor man kan se landene, byerne, vejene imellem dem og spillernes positioner).
- ***Settings*** gemmer de forskellige options, som brugeren kan sætte.
- ***Player*** repræsenterer de forskellige spillere.
- ***GUIPlayer*** repræsenterer den spiller, der styres af brugeren via musen og den grafiske brugergrænseflade.
- ***RandomPlayer*** repræsenterer en computerstyret spiller, der altid vælger en tilfældig vej.
- ***GreedyPlayer*** repræsenterer en computerstyret spiller, der altid vælger den vej, hvor destinationsbyen har højest værdi.
- ***SmartPlayer*** repræsenterer en computerstyret spiller, der vælger den vej, der på sigt giver flest penge.
- ***CGTest*** indeholder en Test Fixture til konstruktion af et simpelt netværk. Denne skal bruges i Computerspil 2.

Ovenstående klasser må I kun modificere, når I får eksplicit besked herom.

I skal nu modificere ***Country*** klassen. Tilføj en ny feltvariabel ***game*** af type ***Game*** og implementér nedenstående accessor og mutator metoder:

- ***Game getGame()***
- ***void setGame(Game game)***

Modificér dernæst ***bonus*** metoden, så de tilfældige værdier beregnes via det ***Random*** objekt, der er skabt af det ***Game*** objekt, som ***game*** refererer til. Dette kan fra ***Country*** klassen tilgås via metodekaldet:

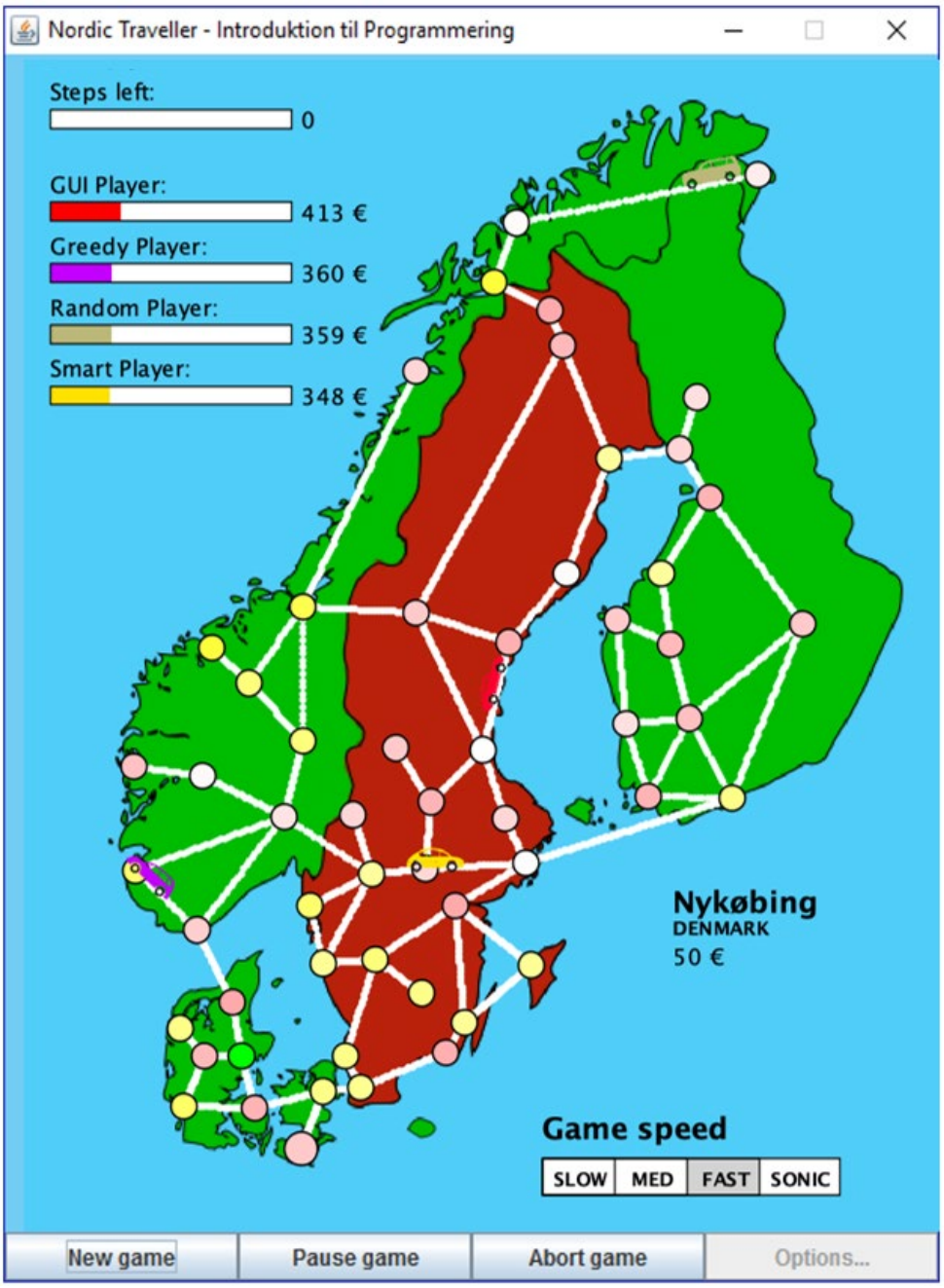
- ***game.getRandom()***

Hvis I har kodet jeres fire klasser korrekt, kan det samlede projekt nu kompileres.

Opgave 8

Kald klassemetoden ***createGameBoard*** i ***GUI*** klassen. Metoden opretter de nødvendige lande, byer og veje, og I vil nu se nedenstående vindue med et kort over Norden (nogle af vejene er færgeruter).

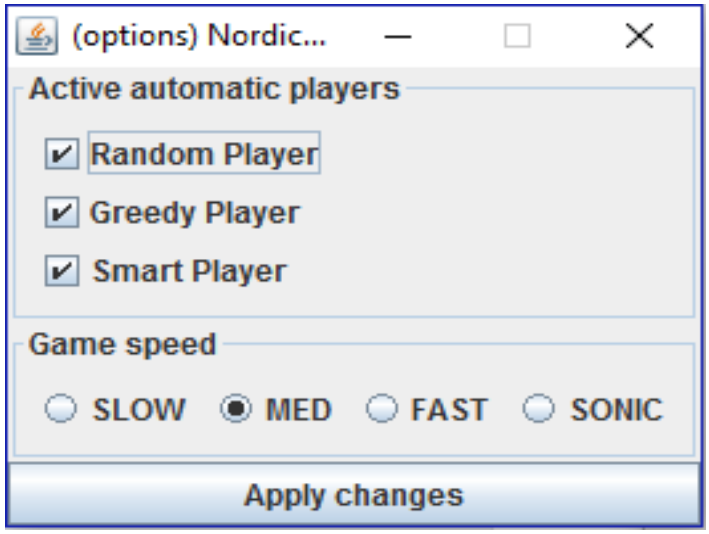
- Man starter et nyt spil ved at trykke på ***New game*** knappen.
- Spillet kan på ethvert tidspunkt pauseres ved at trykke på ***Pause game*** knappen. Teksten på knappen skifter så til ***Resume game***, og et nyt tryk på knappen genoptager spillet. Spillet kan afbrydes ved at trykke på ***Abort game*** knappen.
- Spillets hastighed (dvs. hvor hurtigt spillerne bevæger sig) kan styres ved at klikke på ***Game speed*** knapperne i nederste højre hjørne.



- Cirklerne repræsenterer byer. Farven i cirklen angiver byens værdi med følgende indkodning:

  Når musen placeres oven på en by, kan man (over ***Game speed*** knapperne) se navnet på byen, samt dens præcise værdi.
- Stregerne mellem byerne angiver veje (eller færgeruter). Længden er en grov indikation af, hvor mange skridt, det vil tage at følge vejen (lang streg ≈ lang vej ≈ mange skridt).
- Bilerne angiver spillernes placering. Ude til venstre kan man se nogle aflange rektangler, der angiver, hvor mange penge, hver spiller har indsamlet. Farverne på rektanglerne matcher farverne på spillernes biler.
- Det mørkeblå rektangel i øverste venstre hjørne viser, hvor mange skridt, der er tilbage i spillet. Når man når ned på 0, kan ingen spillere længere bevæge sig, og vinderen er den med flest penge.
- Den røde spiller (GUI Player) styres af den person, der spiller, mens de øvrige spillere styres af programmet. Man styrer den røde spiller ved at klikke på byer. Hvis den røde spiller befinder sig mellem to byer, vil et klik på en af disse byer, få spilleren til at bevæge sig mod den angivne by. Hvis den røde spiller befinder sig i en by, vil et klik på en naboby, få spilleren til at bevæge sig mod den angivne by (forudsat, at der findes en direkte vej mellem de to byer). Alternativt kan man anvende piletasterne, idet et tryk på en af disse vælger den by, der bedst matcher pilens retning (samtidigt tryk på to piletaster giver den retning, der ligger midt mellem de to pile).
- ***Options*** knappen i nederste højre hjørne åbner nedenstående dialogboks, hvori man kan foretage forskellige valg. De aktuelle valg huskes fra spil til spil (også selv om programmet lukkes ned i mellemtiden). Man kan blandt andet vælge, hvilke automatiske spillere man vil kæmpe imod. Random Player vælger en tilfældig vej, Greedy Player vælger den vej, hvor destinationsbyen har højest værdi, mens Smart Player vælger den vej, der på sigt giver flest penge. Man kan ikke trykke på ***Options*** knappen, mens et spil er i gang.



Spil nogle spil og afprøv de forskellige faciliteter i spillet. Prøv at finde nogle gode strategier for at samle flest mulige penge sammen.

Opgave 9

I skal nu afprøve om de ting, som I har lavet Computerspil 1, fungerer korrekt ved at kalde klassemetoden ***test*** i ***TestServer*** klassen med parameteren ”CG1”.

Testserveren afprøver hver enkelt af de klasser, som I selv har skrevet, sammen med vores løsning – således at I kan se, hvilke klasser, der er fejl i. Afprøvningen foretages ved at udføre en række regression tests for konstruktørerne og metoderne i klassen.

Testserveren forudsætter, at I ikke laver unødvendige kald af metoderne i ***Random*** objektet. Hvis f.eks. ***bonus*** metoden laver to kald af ***nextInt*** for at beregne bonussen, vil de pseudo-tilfældige tal komme ud af sync, og testen vil fejle.

Hvis testserveren finder fejl, skal l gennemgå jeres kode og forsøge at rette dem.

I de næste afleveringer skal I:

- Lave automatiske regression tests for alle konstruktører og metoder (Computerspil 2).
- Bruge nedarvning til at strukturere jeres kode, således at der er flere forskellige slags lande og flere forskellige slags byer (Computerspil 3).
- Udvide den grafiske brugergrænseflade med nogle ekstra knapper, labels og tekstfelter samt en menubar (Computerspil 4).
- Optage spil ved hjælp af en ***Log klasse***, hvis objekter kan gemmes i filsystemet, for sidenhen at blive genindlæst og afspillet (Computerspil 5).