Informasjonsvitenskap

INFO 233, Vår 2014

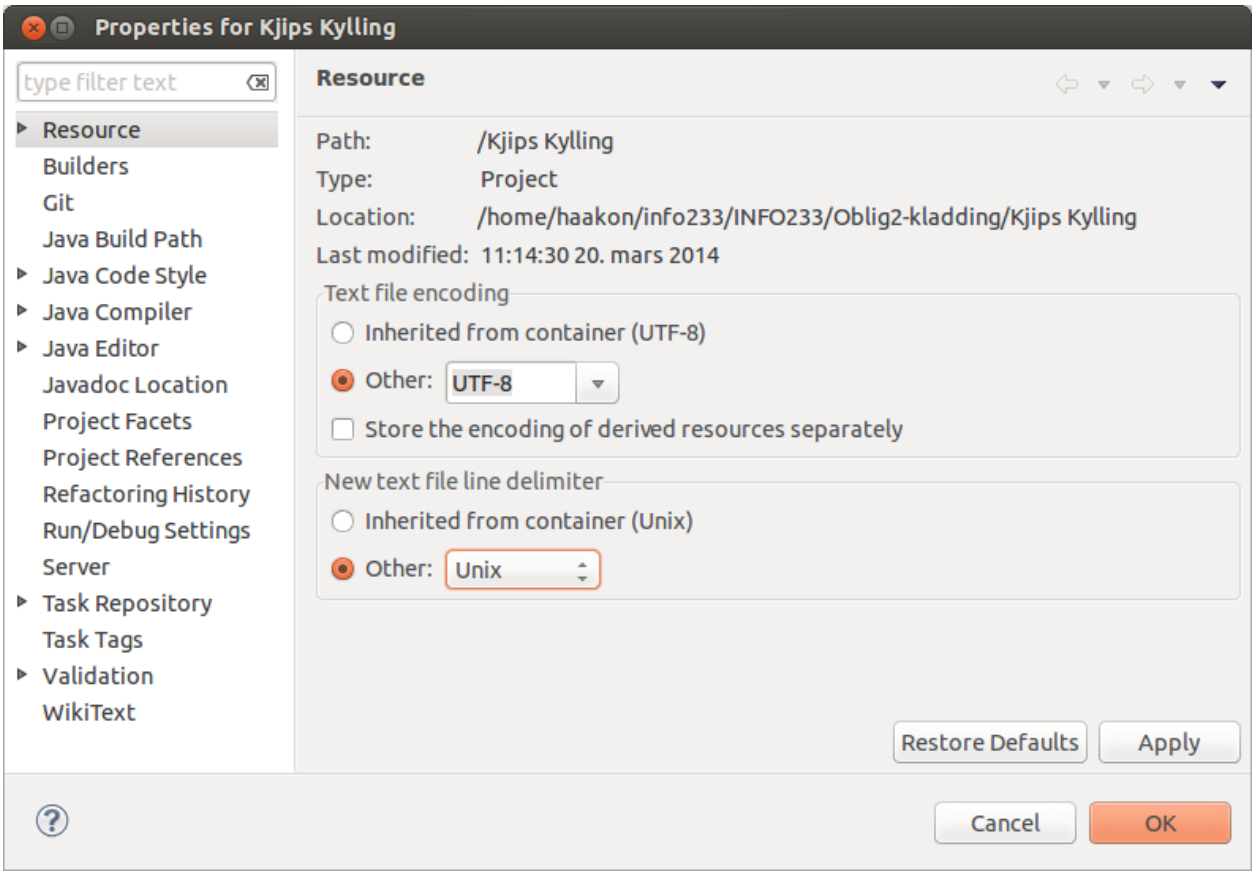
Andre Obligatoriske oppgave

Utlevert 28.mars 2014

Innleveringsfrist klokken 15.00, 7. mai 2014

1 Regler

Oppgaver vurderes til godkjent/ikke godkjent.
Dere jobber enkeltvis eller i par.
Leveringsformatet er et Eclipseprosjekt, eksportert til en komprimert fil. (En av: zip, tgz eller tar.gz. Rar-filer godtas ikke.)
Merk at prosjektet dere får utlevert er i UTF-8. Dere bør derfor etter å ha importert det, sjekke filformatet. Det gjør dere ved å høyreklikke på prosjektmappen, velge properties, og passe på at det ser ut som dette:



2 Oppgaven

Den andre obligatoriske oppgaven omhandler en utvidelse av et spill, kalt Kjips Kylling, som er en forenklet klon av Chip's Challenge[1]. Spillet har allerede en enkel grafikkmotor, og dere skal utvide spillet. Lyd er ikke en del av oppgaven, og dere kan ignorere alt som har med lyd å gjøre.
Før dere begynner å kode anbefaler vi å lese godt igjennom oppgaven, se hvordan relevante kodebiter fungerer, og tenke gjennom før en begynner å kode. Husk å bruke tiden godt. Det er nok å gjøre, og mye blir lettere å gjøre hvis dere har en plan først.

[1] Se `http://en.wikipedia.org/wiki/Chip%27s_Challenge` for mer om spillet

3 Spillet

Spillet Kjips Kylling består av flere brett, en spiller og monstre. Målet med spillet er å klare alle brettene. For å klare et brett må du komme deg fra begynnelsen på brettet til slutten. Dersom du går tom for tid, kommer borti et monster, eller tråkker på en dødelig rute i spillet dør du. Når du dør kan du prøve brettet på nytt. Brettet består av forskjellige ruter. Noen kan gås på, andre ikke. Noen er dødelige, andre ikke. Brettet har også forskjellige monstre. Disse monstrene beveger seg etter faste mønstre. Monstrene kan ikke dø i utgaven dere skal lage.[2] Når du har nådd slutten på et brett begynner neste brett. Det følger med tre eksempelbrett, men dere står fritt til å lage fler. Se beskrivelse av filformatet i slutten av oppgaven for detaljer.

4 Om koden

For å å få til deres versjon av spillet må dere utvide/endre kode som alt er laget og finnes tilgjengelig. Dere får denne koden, og må sørge for å legge den til i prosjektet. Husk å implementere interface der hvor slike er gitt, som for eksempel Monster-interfacet.

4.1 Introduksjon til kodebasen

Koden er lagt opp etter MVC-mønsteret. Det betyr at koden kan sies å være representert i tre lag:

- Ett lag som representerer modellene, dvs. forskjellige entiteter i spillet. I koden ligger dette i pakken game.entity
- Ett lag som representerer kontrollerene, dvs. kode som er ansvarlig for å binde modellen sammen med presentasjonen. Ligger i game.controller
- Til slutt ett lag som håndterer presentasjon av informasjonen i modellene. Ligger i game.view

I tillegg er det IO-pakker som leser fra disk, snakker med andre programmer, etc. Her ligger blant annet grensesnittet ResourceLoader med sine implementasjoner, ResourceLoader har ansvar for innlasting av ressurser som monstre, brett, ruter, grafikk etc.
Koden som dere får er sortert under forskjellige pakker, som følger:

game.controller Inneholder klassen *Game*, som er en kontroller for spillet. Her finner du blant annet en game-loop, som kjører kontinuerlig og som er ansvarlig for å gi ut ticks. (Se seksjon 6.2.)

game.controller.input Er hvor input til spillet, som tastatur, mus, gamepads, osv. ligger. Koden dere får har kun en enkel tastaturklasse som tar hånd om input. Dere trenger ikke røre noe kode her inne.

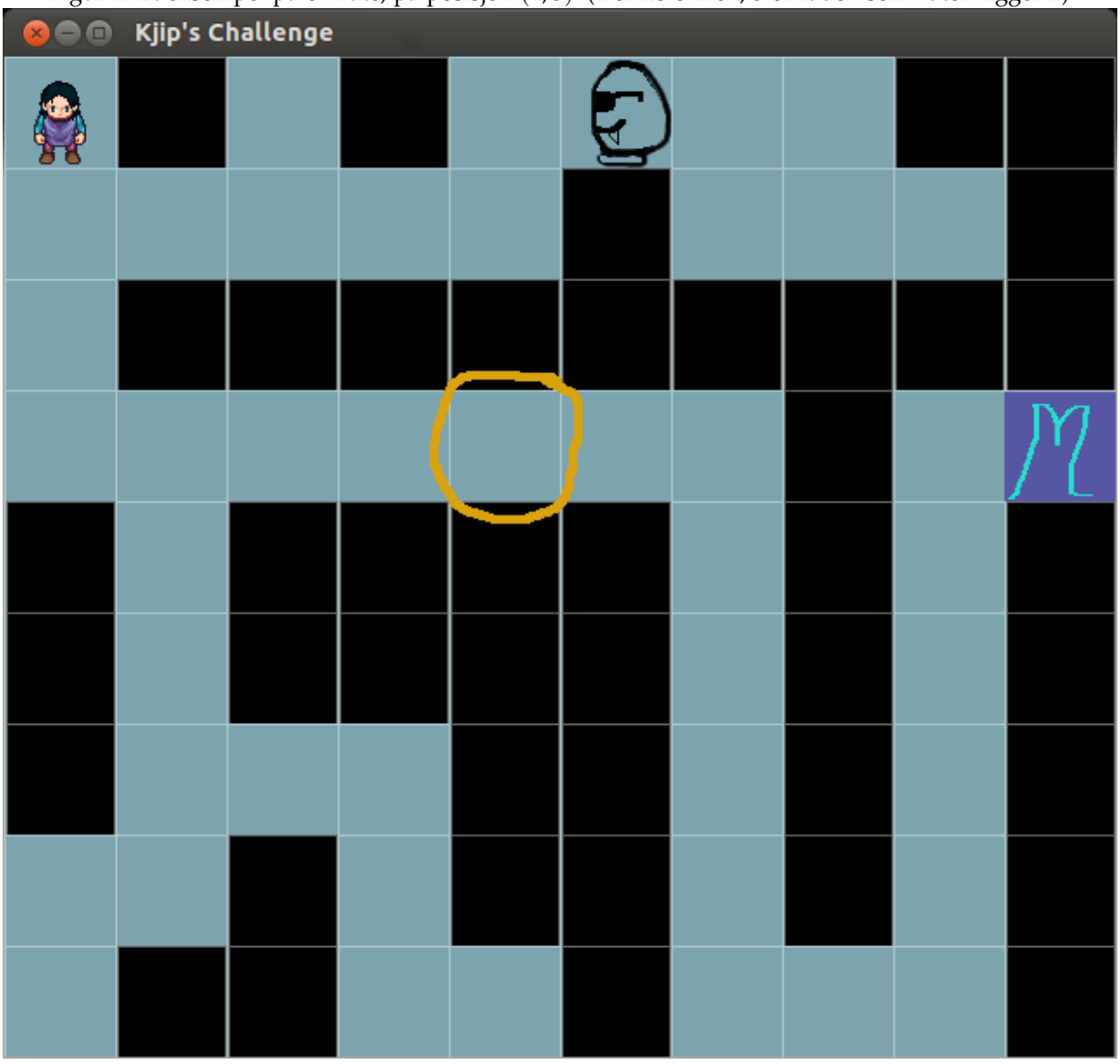
game.entity Har SimplePlayer som representerer en spiller, og TileLevel som representerer et brett.

[2] I originalen kunne du drepe et monster ved å lure det til å tråkke på en bombe.

game.entity.monster Inneholder forskjellige monstre. Når dere får den inneholder den AbstractMonster som er et forenklet monster som ikke har tick() metoden sin implementert, og ExampleMonster som er et monster med alt implementert, slik at dere ser hvordan en kan gjøre det.

game.entity.tiles Forskjellige klasser som representerer ruter (tiles), dvs. en del av brettet, samt hjelpeklasser for å gjøre ting enklere.

Figur 1: Et eksempel på en rute, på posisjon $(4, 3)$. (4 er kolonnen, 3 er raden som ruten ligger i.)



game.entity.types De forskjellige typene entiteter. Her er kun interface, og ikke noen implementerende klasser.

game.io Har klasser som snakker med disk og evt. andre programmer som en SQL-database.

game.util Har forskjellige hjelpemetoder, som Direction for retninger, og Movement for hjelpemetoder til bevegelser.

game.view Har GameWindow-klassen som er en JFrame som holder på et lerret[3]. Et lerret er en klasse for å tegne til en flate, som for eksempel en JFrame. I tillegg til metoder for popups, kan du også legge til kode om du vil ha mer GUI.

game.view.gfx Hvor grafikken blir tegnet. Det er også en klasse for å laste bildefiler inn til grafikkmotoren. Dere trenger ikke endre noe her.

game.main Inneholder utelukkende Main, en klasse som er inngangspunktet[4] for programmet. Det er bare denne klassen i koden dere får som har en main-metode.

5 Oppgaven

Oppgaven dere skal gjøre kan deles inn i to hoveddeler: Databasebruk og Monstre. Det er også noen ekstrating dere skal gjøre.

5.1 Utvidelse til bruk av SQL

Dere skal bruke databaser i dette spillet, nærmere bestemt Apache Derby, som kjører som en del av programmet deres, så dere slipper å tenke på innlogging og slikt. Bibliotekene dere trenger kommer med prosjektet, og er lagt til under lib/ mappen, og satt opp i prosjektet.

5.1.1 ResourceLoaderSQL

Prosjektet har et interface ResourceLoader som beskriver hvordan en klasse kan holdes ansvarlig for å laste inn ressurser fra disk, som blir implementert av klassen ResourceLoaderCSV. ResourceLoaderCSV leser dataene fra 4 spesifikke .csv[5] filer og henter inn data derfra. Vi vil at dere skal implementere en ny klasse ResourceLoaderSQL som bruker en SQL basert database til å hente ut data fra. For at dette må til bør dere gjøre følgende

1. Se igjennom .csv filene. Dere kan åpne dem i teksteditorer som Vim og Emacs, eller regneark som MS Excel eller LibreOffice Calc.
2. Finn ut av hva datatypene er, og hvordan disse kan overføres til SQL.
3. Finn ut av hva relasjonene er, og hvordan disse blir vedlikeholdt.[6]
4. Når dere nå har modellert en database, da, og først da bør dere skrive koden.
5. Og når dere har skrevet koden, sjekk at den virker.
6. Så kan dere begynne å legge dataene inn i databasen.

[3] Canvas
[4] entry-point
[5] CSV - Comma Separated Values
[6] For eksempel, se på alias.csv og standard-tiles.csv

7. Og nå når dataene er i databasen, og dere vet at den virker, da bør dere skrive ResourceLoaderSQL klassen.

8. Nest sist, bytt ut ResourceLoader loader = new ResourceLoaderCSV(); med ResourceLoader loader = new ResourceLoaderSQL();

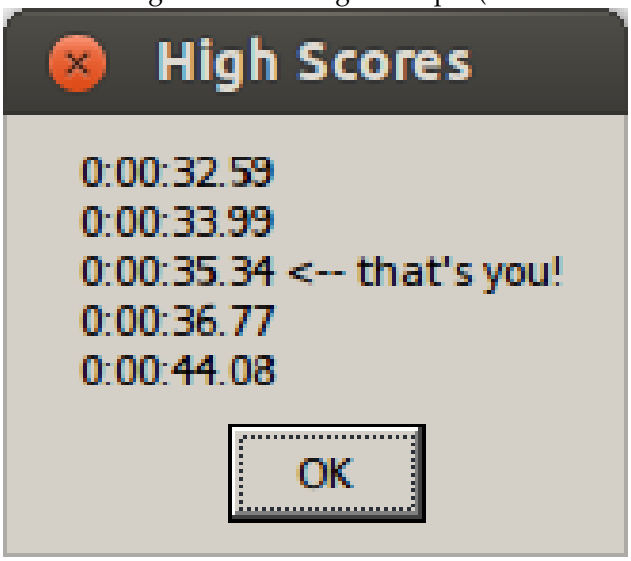
9. Til slutt, fiks alle feil som kommer.

Det er noen viktige ufravikelige krav til ResourceLoaderSQL: Når dere starter opp spillet skal alle data som trengs fra databasen lastes inn med en gang. Dere må derfor lagre data i kjøretidsminnet for å unngå at dataene må lastes på nytt. Dere bør absolutt bruke Map[7] til å holde på disse dataene, og se på hvordan ResourceLoaderCSV gjør det. Dere trenger ikke å lagre brettene i relasjonsdatabasen, de kan lagres som tekstfiler slik de er nå. Bildene skal heller ikke ligge i databasen. (Se avsnittet om brettene 6.5 for mer info.)

5.1.2 HighScore

HighScore tabell skal lagres, og når du kommer i mål skal den skrives ut, med spilleren sin plassering vist.

Figur 2: En mulig måte å vise highscore på. (Tatt fra SkiFree)



Score i denne sammenhengen er ikke mer komplisert enn antall tidels sekunder brukt, jo færre jo bedre. Dere må altså lage en ny databasetabell som holder orden på slikt, og tilhørende spørringer for å hente ut informasjon.

5.2 Monstre

I prosjektet nå er det kun et monster som går en tilfeldig retning. Dere skal lage flere av de klassiske monstrene. I tillegg til å designe klasser for monstre skal dere også legge til en måte å lagre monsterinfor-

[7] se `http://docs.oracle.com/javase/7/docs/api/java/util/Map.html`

masjon på i databasen. Dere må både lage en tabell for monstertypene, og en som binder monstre til brett i en mange-til-mange-tabell. Denne mange-til-mange-tabellen må si hvilket monster det er snakk om, hvor på kartet den skal være og hvilket brett det er snakk om. Monstrene dere skal lage[8]

Monstrene dere skal lage er:

VenstreMonster Dette monsteret går oppover til det treffer et hinder. Da snur det til venstre, og går den veien istedenfor.

HøyreMonster Dette monsteret går oppover til det treffer et hinder. Da snur det til høyre og går den veien istedenfor.

OppNedMonster Går opp til det treffer en hindring. Da går det ned til det treffer en hindring. Og slik fortsetter det.

VenstreHøyreMonster Går til venstre til det treffer en hindring til det treffer en hindring. Når monsteret treffer en hindring snur det 180 ° og går den veien istedenfor.

MålsøkendeMonster Går alltid ett steg nærmere spilleren. Skal ikke planlegge en rute men bare gå den ruten som tar den nærmere spilleren.

PatruljeMonster Følger en gitt patrulje. En patrulje er en streng med en av fire tegn i. N for nord, W for vest, E for øst og S for sør.[9] Når den har fulgt alle stegene i patruljen begynner den forfra igjen. En patrulje på "NNEESSWW" skal gå rundt i ring, for eksempel.

Dessuten skal dere lage en måte å representere monstre på i databasen, og dere utvide må utvide TileLevel til å behandle flere monstre på brettet.
I tillegg må dere kalle metoden tick() på alle monstre i TileLevel. Når dere kaller tick() kan monstrene gjøre en handling. Tick blir kalt 120 ganger i sekundet. Dersom dere gjør noe ved hver mulighet, vil monstrene bevege seg altfor fort. Se eksempelmonster på en mulighet til å begrense hastigheten. Når dere kaller tick, skal monstrene bevege seg i prioritert rekkefølge. Det betyr at dersom du har to monstre med forskjellig prioritet, skal den med høyest prioritet gå først. Dersom de har samme prioritet betyr ikke rekkefølgen noe.
Her bør dere bruke en prioritetskø[10]. Når dere har gjort alt dette, skal dere legge til to monstre på hvert brett. Bruk systemet dere har laget.

6 terminologi

6.1 Game-loop

En game-loop er en uendelig løkke, og er tradisjonelt delt inn i to:

[8] Dere kan godt begynne med monstre, og putte dem inn i TileLevel istedenfor ExampleMonster. Se linje 119 i game.entity.TileLevel
[9] Siden resten av koden er på engelsk følger vi samme praksis her.
[10] se `http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html`

oppdatering Oppdaterer tilstanden til alle objekter i spillet. Blant annet ting som å sjekke om spilleren har dødd, lese ting fra tastaturet, og la datastyrte entiteter som monstre bevege seg.

tegning tegner verdenen spilleren skal se etter opppdatering.

I spillet dere skal utvide er løkken forenklet, ved at tastaturinput og tegning er gjort for dere. Det er altså kun oppdatering som skjer i løkken, tegning skjer i en annen tråd.

6.2 Tick

Et tick er en sjanse for en skapning til å velge å gjøre en ting. Dersom du for eksempel får 4 ticks i sekundet kan ikke en AI[11] reagere raskere enn 1/4 sekund. Bare fordi du får mange ticks betyr ikke at en AI må bruke alle. Den kan utmerket godt kun bevege seg annenhver tick, eller sjeldnere. I vårt spill har alle klasser som implementerer Tickable-interfacet en tick() metode som er void, og lar objekter av den klassen oppdatere seg selv når metoden blir kalt.

6.3 Sprite

En sprite er et mulig bilde av en entitet eller en del av en entitet i et spill.[12] I Kjips Kylling er for eksempel spilleren som ser ned en sprite. Det er tre andre sprites som brukes om spillerfiguren, nemlig når den ser opp, til venstre eller til høyre. En gulveflis har bare en sprite, siden den ikke kan snu seg til venstre.

6.4 Spritesheet

En spritesheet er en bildefil som inneholder mange sprites samlet i en fil. Ble opprinnelig brukt for å spare på systemressurser, og blir fortsatt brukt på internett av den grunn. (Hvis du har mange bildefiler som må sendes over serveren må du motta og behandle mange kall. Dersom bildefilene er små bruker du plutselig mer tid på å behandle kallene enn på å sende informasjon. Men dersom du samler alle de små bildene sammen i ett bilde og sender det, og deretter sakser ut biter av bildet til nettsiden din, går alt mye fortere. Vi som har såpass små bildefiler kunne utmerket godt sluppet unna med å ha separate bildefiler, men vi valgte å ha det med, fordi det er enklere å se hvordan bildene ser ut, mindre File objekter å skyfle rundt på, for å følge tradisjonen, og fordi det er greit å ha sett hvis du i fremtiden skal lage nettsider. Se `http://css-tricks.com/css-sprites/` hvis du lurer på hvordan det brukes i nettsider.

6.5 Filformater

Dere trenger ikke røre hverken brettfiler eller spritesheets, men dere kan hvis dere vil. Her er derfor en oversikt over filformatene.

[11] AI = kunstig intelligens. Se `http://no.wikipedia.org/wiki/Kunstig_intelligens`

[12] På eldre maskiner som den første Nintendoen, var det sterkte begrensede ressurser. En fant ut at dersom en lastet inn flere bilder på en gang som ett stort bilde var det mye raskere for grafikkprosessoren. En kunne også veldig enkelt be om å kun tegne en bit av bildet. Tradisjonen med å samle bilder fra samme entitet på et sted ble da født. En kunne også modde for eksempel det første GTA spillet med å åpne opp spritesheets og tegne om på ting.

6.5.1 Spritesheets

Det er fire forskjellige spritesheets i prosjektet, og ligger i undergruppen art/.

figur.png Er spriten til hovedpersonen. Den er laget vha. `gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Ge` og dere kan lage deres egen figur der. Den har mange tilstander som kan brukes, men vi bruker bare fire av dem. Dersom du lager en ny bildefil i siden du vil bruke, bør du først gi figur.png et nytt navn, og så gi den nye filen navnet figur.png etterpå.

Sara.png Er maskoten til opengameart. Dere kan bruke den istedenfor figur.png hvis dere vil, da de har samme format.

monstre.png Er en spritesheet med flere monstre på. Hvis dere vil modde monstrene til å se annerledes ut, kan dere redigere den filen.

tiles.png Er spritesheeten for rutene på kartet. Endrer dere på dette bildet endrer dere hvordan flisene blir tegnet.

Tilesheets er forventet å legge under art/ undermappen. Selv om du kan registrere tilesheets andre steder, anbefales det sterkt at dere følger konvensjonen.

6.5.2 CSV-filene

CSV er et filformat som forenklet sett beskriver en todimensjonal tabell. Hvert felt i en rad er adskilt med komma, med en mulig overskrift som beskriver tabellen. Dette er nok til å beskrive alle csv-filene i oppgaven. Hvis dere vil lese mer om CSV, kan dere se på`http://en.wikipedia.org/wiki/Comma-separated_values` for generell informasjon, og `http://tools.ietf.org/html/rfc4180` for en spesifikasjon.

adventure.csv beskriver hvilke brett som skal spilles, med rekkefølge, navn på brettet og til slutt hvor brettfilen finnes.

spritesheets.csv beskriver hvilke spritesheets som skal lastes inn, med deres navn, plassering av filen, og størrelsen på et bilde (tilesize) i piksler.

standard-tiles.csv beskriver vanlige ruter i et brett, med navn, om spilleren skal kunne gå på dem, om spilleren dør om den betråkkes, og hvor i spritesheeten den er (kolonne og rad som to felt).

alias.csv beskriver aliaser, altså hvilke bokstaver som betegner hvilke ruter, med tegnet, og så navnet på ruten.

6.5.3 Brettfilene

En brettfil er en vanlig tekstfil, og består av følgende: Først seks heltall på en linje.

1. Hvor mange kolonner som er i brettet
2. Hvor mange rader som er i brettet
3. Hvilken kolonne spilleren begynner på

4. Hvilken rad spilleren begynner på
5. Hvilken kolonne spilleren skal begynne på
6. Hvilken rad spilleren skal ende på

Så følger en rekke linjer med enkelttegn, et tegn for hver kolonne, en linje for hver rad. Hvert tegn svarer til en rute, som definert i alias.csv. (Disse definisjonene skal selvsagt over i den relasjonelle databasen.)