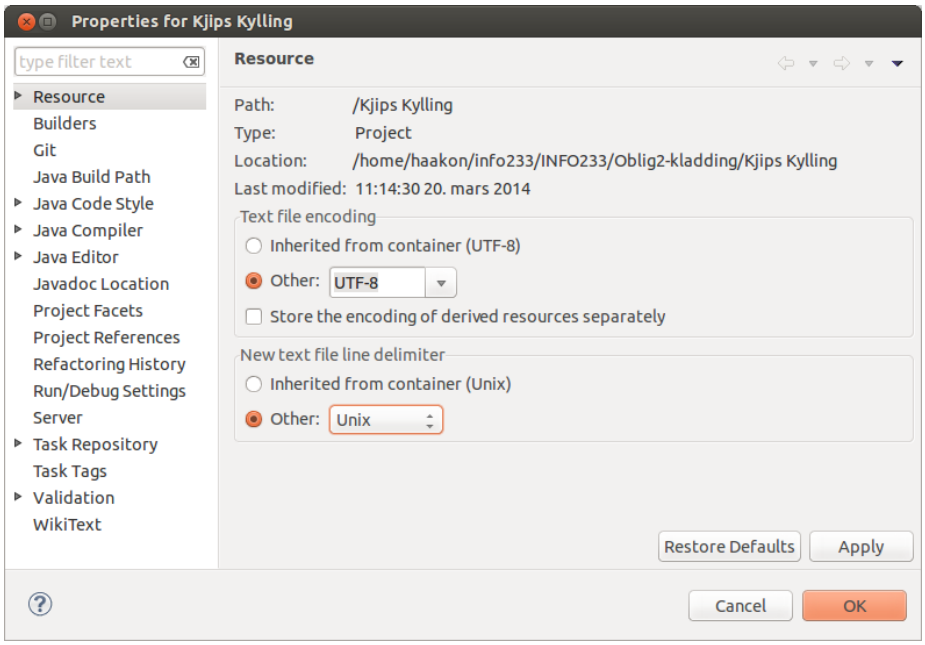
Obligatorisk oppgave info233, kladd.

1 Frister, etc.

Utlevert [dato], frist [data] klokken [klokkeslett] Etc. Antar det er samme format som forrige gang, med godkjent, ikke godkjent? Godtar vi parprogrammering?

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 obligen 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 obligen, 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, tar ani 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 denne utgaven.[2] Når du har nådd slutten på et brett begynner neste brett.

4 Om koden

4.1 Introduksjon til kodebasen

Koden er lagt opp etter MVC-mønsteret. Det betyr at koden er en av tre ting.

- Den kan representere en bit av en modell[3], dvs. forskjellige ting i programmet, som monstre, spilleren og slikt.
- Eller så kan den være en bit av en kontroller[4], dvs. noe som styrer oppførsel i programmet
- Til slutt kan den også være en del av presentasjonen[5] av programmet.

I tillegg er det IO-pakker som leser fra disk, snakker med andre programmer, etc. Her ligger ResourceLoader og implementasjoner, som abstraherer innlasting av ressurser.

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*.

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. Andre klasser har egne pakker for å holde ting relativt oversiktlig.

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.

[2] I originalen kunne du drepe et monster ved å lure det til å tråkke på en bombe.
[3] I spillet ligger disse klassene under game.entity, og dens underpakker
[4] Dette ligger i game.controller
[5] Og dette ligger i game.view

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

game.entity.types De forskjellige typene entiteter. Her er kun kontrakter, 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. 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 classe som er inngangspunktet[6] 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 en kontrakt 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[7] 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.

[6] entry-point
[7] CSV - Comma Separated Values

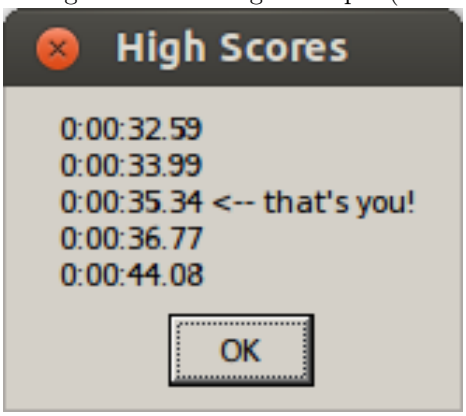
2. Finn ut av hva datatypene er, og hvordan disse kan overføres til SQL.
3. Finn ut av hva relasjonene er, og hvordan disse blir vedlikeholdt.[8]
4. Når dere nå har modellert en database, da, og først da bør dere skrive koden.
5. Og når dere har skrevet koden, sjekk at den virker.
6. Så kan dere begynne å legge dataene inn i databasen.
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 mellomlagre data slik at dere slipper å laste inn på nytt. Dere bør absolutt bruke Map[9] til å holde på disse dataene, og se på hvordan ResourceLoaderCSV gjør det. Dere trenger ikke å hive brettene over til SQL.

5.1.2 HighScore

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

Figur 1: 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 tabell som holder orden på slikt, og spørringer for å hente ut informasjon.

[8] For eksempel, se på alias.csv og standard-tiles.csv
[9] se `http://docs.oracle.com/javase/7/docs/api/java/util/Map.html`

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 monsterinformasjon 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[10]

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.[11] Når den har fulgt alle stegene i patruljen begynner den forfra igjen. En patrulje på "NNEESSWW" skal gå rundt i ring, for eksempel.

I tillegg til å lage en måte å representere monstre på i databasen, og å lage kode for dem, må dere utvide prosjektet til å ta hånd om disse monstrene. Spesifikt må dere utvide TileLevel til å holde på fler monstre, og å holde orden på hvor de er. I tillegg må dere ticke alle sammen. Når det bare er et monster er rekkefølgen opplagt, men når det blir flere vil vi at dere skal la dem tickes etter prioritet. Det vil si, at dersom to monstre har forskjellig prioritet, skal den med høyest prioritet tickes først. Ellers er rekkefølgen uviktig. En god måte å gjøre dette på er en prioritetskø.

6 terminologi

6.1 Game-loop

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

[10] Dere kan godt begynne med monstre, og putte dem inn i TileLevel istedenfor ExampleMonster. Se linje 119 i game.entity.TileLevel

[11] Siden resten av koden er på engelsk følger vi samme praksis her.

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 til en skapning til å velge å gjøre en ting. Dersom du for eksempel får 4 ticks i sekundet kan ikke en AI reagere raskere enn 1/4 sekund. Bare fordi du får n 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-kontrakten 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.

[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.