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.)

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.

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.

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

[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

- 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.

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.

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 IOStuff for lesing av filer og slikt, 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.

[4]Dette ligger i game.controller
[5]Og dette ligger i game.view

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.

TODO: Få dette renskrevet og inn ordentlig:
En kan også dele koden opp i kode dere skal modifisere, og resten.
Skal røre: game.view.GameWindow, game.controller.Game, game.entity.TileLevel
Skal lage: game.io.ResourceLoaderSQL, game.entity.monster.OppNed TilVenstre TilHøyre, VenstreHøyre, Målsøkende, PatruljeMonster

5 Oppgaven

Oppgaven dere skal gjøre kan deles inn i to hoveddeler: Databasebruk og Monstre. I tillegg er det noen underting som også skal legges til etterpå.

5.1 Utvidelse til bruk av SQL

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

5.2 Monstre

Dere skal lage monstre som går snur seg når de treffer veggen, som er målsøkende og følger en patruljerute.

5.3 Til slutt

Alle brett skal ha en makstid, utvid brett med en maksimal tid de må fullføre et brett på. Hvis en spiller ikke er i mål når tiden går ut har han tapt.

– Introduksjon til oppgaven – ResourceLoaderSQL (sql/maps!) – Highscores! (SQL) – Monstre (Kø! (Prioritetskø?)) SELECT score, name from HighScoreWHERE level = ? ORDER BY DESC LIMIT 10;

6 terminologi

6.1 Game-loop

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

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.

[6] entry-point

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.[7] For eksempel har monstrene i eksempelmonster fire sprites, og kan dermed se ut på fire forskjellige måter. En kan selvsagt gjøre ting mer komplisert (tenk på for eksempel animasjoner og slikt), men vi holder oss til enkle ting.

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.

[7] I eldre spill som på den første Nintendoen, var det sterkte begrensede ressurser. En tok derfor å tegnet monstre og delte bilde opp i mindre deler, siden prosessoren på Nintendoen ikke kunne tegne hele bilde på en gang. En har siden utvidet teknikken slik at deler av en karakter som er logisk adskilte kan tegnes sammen, slik at du kan skyte av en arm fra et monster, etc. Vi bruker bare et bilde på en entitet.