Meeting Agenda

Date: 2018-04-26

Participants:

1. Vad vi måste tänka på

Gå igenom anteckningar från förra mötet.

2. Under mötet…

Bestämmer oss att ta bort TankGun klassen.

Variabeln power borde inte sparas undan i Tank - ta bort den från konstruktorn i Tank.

Leyla
- kodar shoot() och aim()
    - döper om shoot() till fire() för att inte blanda ihop med Shot klassen
    - Tar bort weapon klassen och lägger till all funktionalitet i Tank klassen
    - Tar bort TankGun klassen för att den inte gör någonting :)
    - Skriver (om) aim() och fire() i Tank

Patricia
- tar bort argumenten i fire() klassen och döper om till fireTank()
- gör ett test för fire()

Vi utökar aim så att den tar hänsyn till vinkelns värde och sätter in begränsningar.

Adam
- kodar metoder i controller så att tangenttryckningar kopplas ihop med vår moveTank() funktion
- sätter ihop en prototyp med körbar tank-bild

Carl
- jobbar med view

3. Tills nästa möte

Fortfarande oklart hur vi gör med Weapon…

Vi borde gå igenom design modellen tillsammans och diskutera eventuella förändringar och se till att alla förstår hur allting hänger ihop.

Nästa möte: 27/4 kl 8:00

MÖTE MED JOACHIM

- RAD version 1 klar tills nästa torsdag.
    - Utgå från skiss på prototyp…
    - Snapshot av RAD - ögonblicksbild - så här ser vårt GUI just.
    - Fixa den röda texten och sammanställ allt i dokumentet...

- SYSTEM DESIGN dokument ska göras sen…lägg till slutliga uml diagrammet där (kan generera ett via intellij)
- Problem i GitHub → ta bort **out/production/classes** från projektet!!!

Vi måste kunna förklara hur applikationen är konstruerad…

- desktopLauncher → controller (ärver game från libgdx så att den kan byta mellan olika skärmar/scener) och implementerar InputProcessor (den ger oss KeyEvents funktioner…)
    - Tomma metoder som vi inte använder läggs i en Adapter, genom att vi skapar en Adapter klass kan vi låta den implementera alla metoder från InputProcessor.
    - Controller får delegera (ärva?) Adapter för att på så sätt bara använda de metoder den behöver från InputProcessor.

  Controller ← → BaseScreen (**cirkulärt beroende**)
    - Screen har en referens till Controller som har en lyssnare…
    - **evt.getSource** → if(...) kan användas för att man ska veta vilka knappar förväntas
    - Controller behöver alltså inte ha en referens till BaseScreen?
    - Borde vi döpa om Controller? Låter för anonymt..kalla den GameController istället, sedan kan vi ha små Controller klasser som sköter olika grejer.
    - GameController - lyssnare som byter skärm (det har vi).
        - så måste ha en referens till en screen som byts hela tiden...

Controller → BaseScreen är det beroendet vi vill ha?

Varje skärm har sin inputProcessor - vår Controller behöver därför inte ha en?

Controller lyssnar efter något som har hänt och utifrån det gör den något.

Game-skärmen ska ha en koppling till modellen...

Modell = Domän modellen (i MVC)

Den första klassen som körs ska vara utanför mvc-packetet - den skapar det första skärmen/fönstret som syns

- global variabel som håller den aktuella skärmen
- denna klass borde heta Main...

Vi måste sätta oss in i hur man byter skärm? Med hjälp av setScreen()?

DENNA STRUKTUR BORDE VI HA:

- Launcher → Main → startScreen (implements inputProcessor)
    - Main har pilar till olika skärmar…
    - Okej att ha dubbel referenser/pilar i början till Main
- Main har setScreen som swappar skärmar
- Spelskärmen har en koppling till modellen
    - där har vi en lyssnare som känner av att man till exempel känner av att tanken flyttas och uppdaterar skärmen enligt logiken från modellen.
    - så spelskärmen ska bara vara en bild av modellen

- Ha en tillfällig options klass där du sparar undan alla set options grejer som sedan skickas där de behövs
- Ha globala variabler i Main
- Skicka set options resultatet tillbaka till Main klasse?

Ha något körbart till varje möte…

Vi måste få upp rätt struktur! Maila vid frågor