**Spec – p-uppgift ”Medrun”**

Spelet Medrun går ut på att ta sig så lång som möjligt i flera sammanlänkade banor, poäng ges hela tiden till spelaren baserat på hur snabbt man springer, och hastigheten på spelaren ökar desto längre spelaren kommer, alltså får spelaren mer poäng desto längre spelaren kommer.

Spelet börjar på ett hustak i en stad, spelaren står still i ungefär 2 sekunder.

Efter det så börjar spelaren att springa till höger, med en stadigt ökande hastighet. Hastigheten jämnas ut efter ca 5 sekunder, men ökar fortfarande sakta desto längre spelaren kommer.

När spelaren springer så springer han på hustak. På vägen kommer han därför att möta mellanrum mellan huset han är på och andra hus, det är även olika höjdskillnader mellan olika hus som han måste hoppa upp eller ner för, mellanrummen mellan hustaken är varierande. Olika hus är också olika formade; vissa hus är små medan andra är stora, de har olika typer av hinder byggda på sig, och det kan ibland vara svårt att komma vidare utan att behöva hoppa över hinder på själva husen.

Spelaren kan hoppa genom att trycka på ”space” knappen på tangentbordet. Om spelaren håller in knappen så kommer karaktären att tappa höjd långsammare, vilket resulterar i att karaktären hoppar högre och längre. Om spelaren fortsätter att hålla in space-knappen så kommer karaktären att hoppa igen så fort han nuddar marken.

Om spelaren trycker på ”A”-knappen på tangentbordet så kommer karaktären att minska sin hastighet och springa lite saktare än vad kameran rör sig, detta ger effekten av att karaktären rör sig till vänster på skärmen. Karaktären kan röra sig till vänster till han kommer till det fördefinierade stoppet som ser tills att karaktären inte kan komma ur bild.

Om spelaren däremot trycker in ”D”-knappen så kommer karaktären att röra sig snabbare än vad kameran rör sig på skärmen, detta ger effekten av att karaktären rör sig till höger på skärmen. Karaktären kan röra sig till höger tills han kommer till det fördefinierade stoppet som ser till att karaktären inte kan komma ur bild.

Ifall spelaren råkar springa in i en vägg så är spelet över, karaktären kommer att tumla ut ur bilden och kameran kommer de-accelerera till dess att den stannar.

Det är även ibland låga hålrum som han måste glida under och igenom för att inte springa in i en vägg och dö. Spelaren kan glida på marken genom att hålla in ”S”-knappen på tangentbordet och kommer då glida i ungefär två sekunder, detta visas genom att animationen byts ut till en ”glidar” animation. Därefter kommer karaktären att ställa sig upp och spelaren kommer inte kunna glida på ungefär en sekund.

Spelet är över om spelaren krockar in i en vägg, eller ramlar ned mellan två hus.

Ett GUI visas på skärmen under spelets gång, på GUI:t finns en poängräknare, en klocka som visar den tid man sprungit och en längd-mätare som visar hur långt man sprungit.

När man startar spelet så kommer man till en huvudmeny.

två knappar finns i huvudmenyn;

Start & exit:

Trycker man på exit-knappen avslutas spelet.

Trycker man på startknappen så starar spelet.

Om man trycker på ”escape-knappen”. Under spelets gång så kommer man till paus menyn. I paus menyn så ser användaren en ”fryst” bild från där karaktären befann sig när spelaren pausade spelet. Det står även med en vit text i mitten ”paused”. Under ”paused” texten står det tre texter brevid varrandra med vit text, det står: ”E - unpause”, ”R” - Restart” och ”ESC – Main menu”. Om spelaren trycker på E-knappen i pausmenyn så startar spelet igen. Om spelaren trycker på R-knappen så startar spelet om. Om spelaren trycker på ”Escape-knappen” så går spelet till spelets huvudmeny.

**Klassen och Funktionerna i ”main” klassen:**

*”public class Medrun extends StateBasedGame”:*

den bit text om visar att här börjar klassen, den förlänger den abstrakta klassen ”StateBasedGame” från slick2d för att spelet ska kunna dra fördel utav slicks inbyggda funktioner och spellägen.

*”public Medrun(String name)”:*

Konstruktorn till klassen. Den tar in ett namn som argument och skickar namnet till super klassen. Den initierar sedan de fyra spellägena som används i spelet. Efter det så försöker den sedan läsa inställningsfilen genom att köra ”readSettings” men om det inte fungerar så körs ”writeSettings”.

*”public static void main(String[] args)”:*

körfilen som automatiskt startar när man kör projektet. I denna så skapas ett Medrun objekt och även en ”AppGameContainer” som är ett slicks där spelets inställningar kan ändras på.

*”public void initStatesList(GameContainer gc)”:*

funktionen som initierar alla spellägen. Den plockar alla spellägen från Medrun objektet och kör en funktion kallad ”init” på alla. Denna funktion körs automatiskt när man startar projektet, så den behövs inte startas från ett annat ställe i filen.

*”public static void readSettings() throws IOException”*:

Funktionen som försöker läsa in alla spelets inställningar från en fil. Om inte går så kastas ett ”FileNotFound exception” och då körs ”writeSettings” istället.

*”public static void writeSettings()”:*

Funktionen som skapar settings.ini filen där inställningarna läses in i spelet, och där inställningarna kan ändras på utanför spelet. Den skapar en fil vid namn ”sttings.ini” och skriver sedan alla rader till filen med standard inställningarna som vid ett senare tillfälle kan läsas in av ”readSettings” funktionen.

*”public static boolean isFullScreen()”:*

En funktion som returnerar ”true ” om ” AppGameContainer” har fullskärm aktiverat, annars returneras ”false”.

*”public static void setFullScreen(boolean fullScreen)”:*

En funktion där man ställer in fullskärmsläget beroende på vad man stoppar in för argument till funktionen.

public static boolean isvSync()

En funktion som returnerar ”true ” om ”AppGameContainer” har v-sync påslaget, annars returneras ”false”.

*”public static void setvSync(boolean vSync)”:*

Funktionen där man kan ändra på v-sync inställningen beroende på vad som stoppas in som argument.

public static boolean isDisplayFPS()

En funktion som returnerar ”true” om ”AppGameContainer” visar fps mätaren, annars returneras ”false”.

*”public static void setDisplayFPS(boolean displayFPS)”:*

Funktionen där man kan aktivera eller avaktivera på fps-mätaren beroende på vad som stoppas in som argument.

*”public static int getGameWidth()”:*

reurnerar bredden på spelfönstret.

*”public static void setGameWidth(int gameWidth) ”:*

ställer in bredden på spelfönstret baserat på argumentet.

*”public static int getGameHeight()”:*

returnerar höjden på spelfönstret.

*”public static void setGameHeight(int gameHeight) ”:*

ställer in höjden på spelfönstret baserat på argumentet.

*”public static int getTargetFramerate()”:*

returnerar den bildfrekvens som spelet ska hålla sig under, vanligtvis är detta 60 bilder per sekund, men det är justerbart.

*”public static void setTargetFramerate(int targetFramerate)”:*

Funktionen där man kan själv ställa in hur många bilder per sekund spelet kan ha som max.

*”public static int getMinUpdateTime()”:*

*returnerar dne kortaste tidsintervallet som kan vara mellan två updateringar i spelet.*

*”public static void setMinUpdateTime(int minUpdateTime)”:*

*Sätter det minsta uppdterings intervallet*

*”public static int getDifficulty()”:*

*returnerar den nuvarande svårighetsgraden.*

*”public static void setDifficulty(int difficulty)”:*

*sätter svårighetsgraden till input värdet.*