

GRILLSKA GYMNASIET Rapport  
Teknikprogrammet 2016-05-03  
Gymnasiearbete, 100 poäng  
HT/VT, 2015/2016

**The Legend of Grillska**

Författare:  
Erik Rimskog

Handledare:  
Alexander Alsenholt

# **1. Sammanfattning (Abstract)**

## **1.1 Svenska**

Detta projekt är ett spel som jag själv har skrivit i java från grunden. Det är ett 2D-plattformsspel med ganska många element från ”bullet hell”-genren, alltså att det är väldigt många projektiler på skärmen samtidigt. Spelet ska utspela sig på skolan jag går på, Grillska Gymnasiet Uppsala, och lärarna ska vara bossar som man ska besegra. Jag utgick ifrån en ”tutorial” på internet och det var inte den bästa ”tutorialen” som jag kunde ha tagit, men det funkade alldeles utmärkt för att göra ett funktionellt spel med.

Jag kom fram till att det funkar väldigt bra att göra spel i java och att det är ganska enkelt jämfört med andra språk. Detta projekt är dock inte tillräckligt för att avgöra ifall java är ett bra språk att göra spel i om man kollar prestandamässigt, ty grafiken är inte krävande och kan köras på alla system utan problem. Man måste göra ett spel i 3D med avancerad grafik för att kunna avgöra ifall det är bra eller inte.

## **1.2 English**

This project is a game which i have written myself from scratch. It is made in java and it is a 2D platformer with bullet hell mechanics, which means that there are alot of projectiles on the screen at the same time. This game takes place at Grillska Gymnasiet Uppsala. The teachers are bosses which you are going to defeat. I started from a tutorial from the internet, and it wasn’t the best tutorial i could have picked, but it worked out very nicely to create a game with it.

My conclusion from making this game is that it works very well to create a game with java and that it is pretty easy compared to other languages. Though this project isn’t enough to say whether java is a good language for creating games if we look at performance. The graphics are very basic and aren’t demanding at all, it can be run on any system without a problem. You have to create a game in 3D with demanding graphics to decide whether java is good or bad in that regard.

# **2. Innehållsförteckning**

[1. Sammanfattning (Abstract) 1](#_Toc450070806)

[1.1 Svenska 1](#_Toc450070807)

[1.2 English 2](#_Toc450070808)

[2. Innehållsförteckning 3](#_Toc450070809)

[3. Inledning 4](#_Toc450070810)

[3.1 Bakgrund 4](#_Toc450070811)

[3.1.1 Hur fungerar ett datorspel? 4](#_Toc450070812)

[3.1.2 Vad är programmeringsspråk? 5](#_Toc450070813)

[3.1.3 Vad är java? 5](#_Toc450070814)

[3.1.4 Java som språk för spelprogrammering 6](#_Toc450070815)

[3.2 Syfte 7](#_Toc450070816)

[3.3 Frågeställningar/hypotes 7](#_Toc450070817)

[4. Metod 8](#_Toc450070818)

[4.1 Material 8](#_Toc450070819)

[5. Utförande 9](#_Toc450070820)

[5.1 Tidsförlopp 9](#_Toc450070821)

[5.2 Förbättringar 10](#_Toc450070822)

[5.2.1 Tilesystemet 10](#_Toc450070823)

[5.2.2 Klasser för banor 10](#_Toc450070824)

[5.3 Förklaringar 11](#_Toc450070825)

[5.3.1 Klasstruktur 11](#_Toc450070826)

[5.3.2 Avståndsformler 11](#_Toc450070827)

[5.3.3 Sned sinuskurva 13](#_Toc450070828)

[6. Resultat 15](#_Toc450070829)

[7. Diskussion/analys 16](#_Toc450070830)

[8. Referenser 17](#_Toc450070831)

[9. Appendix 18](#_Toc450070832)

# **3. Inledning**

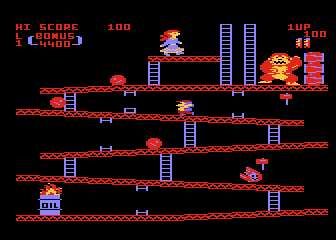
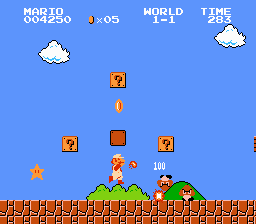
Detta projekt är ett spel (bild 1.1.2) som är skrivet i programspråket java helt från grunden. Det är ett plattformsspel med pixelgrafik och med svår svårighetsgrad. Spelet är väldigt inspirerat av så kallade ”bullet hell”-spel, vilket är en genre som rätt så mycket betyder att antalet projektiler på skärmen vid en angiven tidpunkt är väldigt stort. Exempel på dessa inspirerande spel är bl.a. I Wanna Kill the Kamilia 2 (bild 1.1.1), Rabi Ribi och Touhou Project. Utseende, spelbanor och struktur är starkt inspirerade av Super Mario-spelen samt tidigare nämnda spel. Exempel på sådana saker är att det finns en ”overworld”, alltså en plats där man väljer vilken bana man vill köra, att det finns en prinsessa man vill rädda, samt så är mycket grafik direkt taget ifrån Super Mario.

Spelet kommer att utspela sig på skolan Grillska Gymnasiet Uppsala, och lärarna kommer att vara bossar som man ska besegra på olika vis. Huvudkaraktären kommer att vara Erik Rimskog och han ska äventyra genom skolan och se vad det finns för fasor i skolans salar samt att rädda prinsessan Moa Marklund, ordförande i skolans elevkår.

## **3.1 Bakgrund**

Anledningen till att detta projekt skapades överhuvudtaget är för att jag tycker att programmering är kul samt att spela spel är kul. Om man slår ihop dessa saker så kan det inte gå fel. Det är också så att java, programmeringsspråket som används, inte vanligvist används för att skapa stora kommersiella spel. Så detta projekt är lite av ett test för att se om det ens går att göra ett lite större spel i java. En annan stor anledning till att java valdes är för att jag är mest bekväm med java.

### **3.1.1 Hur fungerar ett datorspel?**

Ett datorspel eller ett videospel är ett interaktivt underhållningsmedium på elektronikenheter. Man har någon sorts skärm och någon sorts styrenhet för att på något sätt uppnå ett arbiträrt mål för att få underhållning och möjligtvis pengar eller kunskap. Spel är ett relativt nytt medium som kom när de första kompatibla persondatorerna kom på 80- eller 90-talet. Det började som simpla korta spel med enkel grafik och ett enkelt mål, som att ta sig högst samtidigt som man undviker hinder (bild till höger). När datorerna blev kraftfullare kunde spelen få bättre grafik och mer avancerade saker man kunde göra. Denna industri har växt till sig till att bli så stort som det är idag. Nu har man avancerad 3D-grafik med fina ljuseffekter och detaljerade objekt.

Det finns många olika spelgenrer, det finns allt från dejtingspel till första persons skjutspel (FPS). Just detta projekt är ett 2D-plattformsspel samt ett ”bullet hell”. ”Bullet hell” (bild 1.2.2) är spel som har väldigt många projektiler flygandes samtidigt. Ett plattformsspel är ett spel där man helt enkelt ska hoppa från en plattform till en annan. Super Mario (bild till höger) är ett gott exempel på ett sådant spel. 2D betyder att spelet är tvådimensionellt, alltså att man endast kan röra sig i två riktningar, oftast uppåt/neråt och höger/vänster. Super Mario är också ett bra exempel på detta. Det finns också spel i 3D, då kan man utöver axlarna som man kan gå i när man är i 2D också gå genom en tredje axel, den som går ”in” och ”ut” ur bilden. Call of Duty är ett bra exempel på ett FPS-spel som är i 3D (bild 1.2.1).

### **3.1.2 Vad är programmeringsspråk?**

För att datorer ska kunna köra program så behöver de en kod som är skriven i endast ettor och nollor, det är väldigt jobbigt att skriva kod på det sättet. Därför finns det programmeringsspråk av olika sorter. I dessa språk skriver man en kod i vanlig text (bild till höger) så att en vanlig person lätt kan förstå den. Datorn förstår dock ingenting, så därför kompilerar man, eller skriver om koden till ettor och nollor så att datorn förstår och kan köra koden. Det finns många olika sorters programmeringsspråk som är bra på olika saker och är lite annorlunda, men man kan göra precis samma saker med alla språk. Det är precis som riktiga talspråk, man kan göra samma saker, fast det låter lite annorlunda. Skillnader på programmeringsspråk kan vara hur de kompileras (mer om detta under nästa rubrik), hur de är uppbyggda (mer under nästa rubrik), vilken syntax de har, alltså vilka ord man skriver för att göra olika saker, och hur många färdiga bibliotek det finns. Ett bibliotek är en samling med användbara funktioner som man använder ganska ofta. Vissa språk har många egna bibliotek medan andra språk knappt har några alls, då får man skriva egna eller ta från tredje part. Dessa bibliotek kan göra saker allt ifrån att lagra data på ett smart sätt till att rita ut grafik på skärmen och hantera nätverk.

public class HelloWorld**{**

public static void main**(**String**[]** args**)** **{**

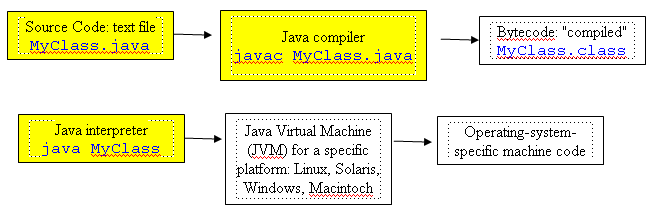
System**.**out**.**println**(**"Hello World!"**);**

**}**

**}**

### **3.1.3 Vad är java?**

(1) Java är ett programmeringsspråk som skapades 1995 av bland annat James Gosling på Sun Microsystems. På denna tid var C och C++ de dominerande språken, så när en litet projektgrupp hos Sun skulle göra ett inbäddat system (ett litet system som ska göra få simpla saker t.ex. timern till mikrovågsugnar) så kom de fram till att C++ skulle ta alldeles för mycket arbetsminne och skulle vara för krångligt. Då bestämde de sig för att de vill ha ett nytt programmeringsspråk som man enkelt kunde flytta till olika enheter. Det var så Java skapades och kom att växa till ett av de mest populära språken idag.

Java är som sagt ett plattformsoberoende språk, vilket betyder att man bara behöver skriva koden en gång, kompilera den och sedan kan man köra det på nästan vilka enheter man vill. Till skillnad från språk som C++ så måste man kompilera sin kod annorlunda till alla olika enheter som man vill köra sitt program på. Det som gör att java är plattformsoberoende är att den inte kompileras direkt till maskinkod (det som datorn direkt läser), utan den kompileras till någonting som kallas bytekod. Det är också ettor och nollor, men denna kod ska inte läsas direkt av datorn som den gör med maskinkod, utan denna kod ska läsas av någonting som kallas för en ”interpreter”. Det är rätt så mycket en virtuell dator som tar bytekoden och gör om den till maskinkod under körning. Det är just detta som gör att java kan köras på många olika enheter. Man skriver alltså koden, kompilerar den till bytekod, installerar en ”interpreter” som är anpassad till enheten man vill köra koden på och sedan kör koden utan att göra några specifika ändringar eller kompilera den annorlunda. Just att den måste köras genom en ”interpreter” gör att programmen generellt körs långsammare än om man kompilerar ett program direkt till maskinkod. Eftersom java funkar på detta sätt finns därför java på nästan allting. Java finns på din dator, din mobil, på internet, i din bil, i din vattenkokare, din mikrovågsugn etc. Det är miljontals enheter som kör java.

(1) Java har ett par olika versioner. Det finns java ME, vilket står för Micro Edition. Detta är java-versionen som körs på inbäddade system. Sedan finns det Java SE, vilket står för Standard Edition. Detta är versionen som vanliga persondatorer kör. Till sist finns det java EE, alltså Enterprise Edition. Denna version används för stora nätverksapplikationer och har standardlösningar för vanliga problemtillställningar.

Java har någonting som språk som C++ och C inte har, någonting som kallas för ”garbage collector”. När man skriver ett program i C måste man själv rensa de resurser av arbetsminnet som man har använt. Om man inte gör det kommer allt detta onödiga data ligga kvar och vara i vägen tills man helt stänger av datorn eller om den på något sätt förlorar strömtillförseln, det är inte så bra. Det kan vara väldigt jobbigt att behöva hålla koll på sådana saker själv, därför har java gjort denna ”garbage collector”. Det är en process som körs i bakgrunden när du kör ett java-program, och det den gör är att den aktivt letar efter resurser som du aldrig mer kommer att använda och tar bort dem åt dig. Det är väldigt händigt och gör att det är lite enklare att programmera i java.

Java är något som kallas för ett objektorienterat språk. Det är ett av många programmeringsparadigm. Ett paradigm är hur språken är strukturerade och hur de fungerar när man programmerar i dem. Det finns ett paradigm som betyder att språket börjar att läsa högst upp i filen och gör allt som står i ordning och hoppar när den blir sagd att göra så. Ett objektorienterat språk är lite annorlunda, det baseras på tanken att göra ”objekt”. Alltså att man gör ett objekt som t.ex. katt, och där kan du säga att en katt ska innehålla sin ålder, pälsfärg och ägare. Sedan kan man jätteenkelt göra hur många katter man vill och väldigt enkelt hålla koll på vad varje katt har för information. Man skapar objekt från filer som kallas för ”klasser”, i en klass står det hur objekten man skapar ifrån klassen ska vara strukturerade.

Java är ett språk som har väldigt många egna färdiggjorda bibliotek för att göra allt möjligt. Java har ett bibliotek för att göra fönster och rita ut grafik, bibliotek för nätverk, bibliotek för smarta sätt att lagra variabler på m.m.

### **3.1.4 Java som språk för spelprogrammering**

Det vanligaste språket att programmera spel i idag är C++ för att det är det språket som ger bäst prestanda. Det språket körs på ganska låg nivå och är därmed snabbt. Java har en ”interpreter” och det gör att java kommer att köras långsammare, med spel vill man att det ska köra så fort som möjligt för att se bäst ut. Java är ett språk på mycket högre nivå än C++, därför kommer det att vara mycket overhead så man kan inte påverka hårdvaran direkt lika snabbt (2). Det språket har varit en standard väldigt länge, det är många företag som använder det med färdiga system som de har använt länge. Det skulle vara jättejobbigt att byta språk nu, det skulle betyda att de skulle få göra om allting helt från grunden och buggtesta allt igen. Så därför är det mycket enklare att hålla sig till det som redan finns som man vet fungerar (2).

De flesta datorspelen görs till Windows, så att vara plattformsoberoende är inte någonting som är i stort fokus. Som sagt är det en nackdel för java att den är plattformsoberoende, det kör långsammare, vilket inte är bra.

Att java har en ”garbage collector” kan också vara ett problem (2). När den börjar leta efter skräp kan den göra så att spelet hackar lite, och det vill man inte. Därför är det bättre att ha ett språk som inte har detta, som C++. Då kan man istället rensa minnet exakt där man behöver och därmed inte behöva något som söker efter det åt en.

## **3.2 Syfte**

Syftet med detta projekt är att studera ifall java verkligen är så dåligt att göra spel i och hur enkelt det är att göra det själv från grunden, alltså att inte utgå ifrån en färdiggjord spelmotor.

Syftet är också att få ett färdigt, komplett, lite längre, och svårare spel som ska vara roligt att spela och där målgruppen (elever och lärare på Grillska gymnasiet Uppsala) ska känna igen sig.

## **3.3** **Frågeställningar/hypotes**

Fråga:  
*Kan jag programmera ett mer komplicerat, större och dugligt spel i java?*

# **4. Metod**

Projektet kommer att programmeras i java med hjälp av IDE:n Eclipse, vilket är ett program som kommer att underlätta programmerandet med hjälp av inbyggda verktyg som gör livet enklare. Det kommer att programmeras mest hemma hos mig för att den datorn har störst skärm och är allmänt enklare att göra stora projekt på. Spelet kommer att göras ifrån grunden, för att underlätta och för att få en bra grund kommer spelet att utgå ifrån en ”tutorial” (en guide som visar hur man gör något) från internet.

Allt ljud kommer att komma från diverse källor ifrån internet, för det mesta youtube. Ljuden kommer sedan att redigeras i programmet Audacity, vilket är en bra, gratis ljudredigerare.

Grafiken kommer dessutom att komma ifrån diverse sidor på internet. De kommer att redigeras så att de passar i Photoshop. Vissa saker som inte finns på internet för att de är originalkoncept kommer att göras själv.

Alla bossar och banor kommer att designas själv, möjligen med lite hjälp av andra parter.

## **4.1** **Material**

Material som har används är följande:

* Java IDE Eclipse Mars.1 v. 4.5.1
* JDK 1.8.0\_60
* En bärbar dator Lenovo ThinkPad L430 med en 2-kärnig Intel Core i5-3230M på 2.60 GHz, 8 GB RAM, 64-bitars Windows 8.1 Enterprise, en 300 GB HDD och en 14-tums skärm med upplösningen 1366x768.
* En stationär dator med en 6-kärnig AMD Athlon II X6 på 2.60 GHz, 12 GB RAM, en ASUS NVIDIA GT 630, 64-bitars Windows 7 Home Premium, en optisk CD-läsare, en extern HDD på 2 TB, en HDD på 1 TB, en HDD på 3 TB, en skärm av typen BenQ på upplösningen 1920x1080 och en äldre Philips-skärm med aspect ratio 4:3.
* Adobe Photoshop CS6 x64 v. 13.0.1
* Audacity v. 2.1.2
* GeoGebra 5.0. 214.0-3D
* ICA A4 rutat kollegieblock.
* Stiftpenna 0.7mm med Fyrisskolans logga.
* HP Prime grafritande räknare.
* Google Chrome version 49.0.2623.87 m

# **5. Utförande**

## **5.1 Tidsförlopp**

Projektet börjades med att utgå ifrån en ”tutorial” ifrån youtube (3). Man fick ett fungerande spel med en drake som kunde gå runt, slå och skjuta eldbollar på en snigelliknande fiende. Man fick dessutom ett simpelt program som instruktören själv hade gjort för att göra banor med. Det var inte perfekt och det var många saker som kunde förbättras. De första veckorna gick åt till att förstå all denna nya kod och till att förbättra saker.

Därefter började processen att lägga till nya användbara saker för att se hur de fungerade. Spikar är en sak som lades till, de består av likbenta trianglar och kan skada spelare ifall han nuddar vid dem. Flygande plattformar var en annan sak, de är platta rektanglar som flyger runt samt att spelare kan ställa sig på dem och följa med dit de åker.

Nu när spelmotorn har testats och experimenterats med så börjar processen med att göra det slutgiltiga spelet. En extern person anlitas för att rita huvudkaraktären ty jag är inte kapabel. Planering för vilka banor som ska göras och vilka bossar som ska göras och vart de ska vara någonstans gjordes. ”Overworld:en” påbörjades och ett ”tileset” till den gjordes. Den första bossen gjordes men blev mycket simplare och tog längre tid att göra än senare bossar, ty jag var oerfaren. Slutresultatet blev en ganska kort svår boss där man ska överleva 4 olika attackmönster för att vinna. Temat var Bowser med Alexander Alsenholts ansikte på. Feedback av klasskamrater sade att denna boss var alldeles för svår för att ha som första boss, därmed gjordes en enklare variant. Denna enklare boss gick ut på att man skulle undvika mycket simpla attacker och vänta tills bossen hoppade, då man skulle springa under honom och trycka på en knapp på andra sidan för att sedan vinna.

Den andra världen påbörjades med att göra två banor. Den ena banan var en helt vanlig bana där man ska ta sig från början till slutet, denna bana blev den svåraste i hela spelet. Den andra banan blev lite annorlunda, den består av 3 frågor som man ska besvara. Varje fråga är uppbyggd av en stor skylt med en matematisk fråga på och 3 olika hål med olika svar på. Tar man fel svar måste man börja om, tar man rätt kommer man till nästa fråga. Den andra bossen gjordes härnäst och det blev med temat av Mattias Karlsson som en blå grafräknare (TI-82). Denna boss sköt matematiska konstanter och grafer mot spelaren. På den sista fasen skulle spelaren undvika olika sorters noter på exakt en minut. Desto mera tiden gick desto svårare blev det. Anledningen till att musiknoter användas var för att Mattias hade varit musiklärare en gång i tiden.

På grund av tidsbrist blev det inga banor till den tredje världen, så det blev endast en boss. Denna boss fick läraren själv designa. Johannes Hultin Knutas ville att hans boss skulle ha temat av bossen Lavos från spelet Chrono trigger. Denna boss, jämfört med de två tidigare bossarna, gick inte ut på att överleva attackmönster och sedan magiskt vinna. Utan på denna boss ska man skriva ord som läraren inte tycker om för att göra skada på bossen. Man gör detta genom att nudda bokstäver som fritt flyger runt bland alla skadliga projektiler som bossen skjuter ut. Efter att man har skrivit 3 succesivt längre ord vinner man striden.

På den sista bossen är Alexander tillbaka för revansch, nu ska man besegra honom och rädda Moa för att vinna spelet. Här möter man den svårare bossen som gjordes allra först. Denna boss som det sista gjorde spelet antiklimaktiskt ty bossen var kort och inte lika avancerad som de tidigare. Detta löstes genom att lägga till en ny fas på den sista bossen. Alltså att man ska möta den tidigare bossen, och sedan direkt efter möta en helt ny boss med helt ny design och annat utseende. Bossen på andra fasen blev av temat ”helvetet” och Alsenholt förvandlas till sin sanna form, en cyborg-version av sin själv. Bossen består av ett svävande huvud och två svävande händer. Målet med striden är att slå på händerna när de väl är inom räckhåll för att minska hans hälsa. Hans attacker är allting ifrån att skjuta massor av röda projektiler i olika mönster till att försöka slå spelaren med de svävande händerna. Denna boss blev ungefär 10 minuter lång och är väldigt svår jämfört med de andra bossarna. När man besegrar denna boss har man tillslut vunnit spelet och man får rädda Moa. Beroende på hur många gånger man har dödat bossen innan så kommer alla dialoger med Moa och med själva bossen att vara annorlunda.

## **5.2 Förbättringar**

### **5.2.1 Tilesystemet**

C:\Users\Erri0401\AppData\Local\Microsoft\Windows\INetCache\Content.Word\tileset.pngEn av de saker som inte var optimala var hur banorna och ”tileset”:sen sparades. Ett ”tileset” är en fil där all grafik för hur rutorna (”tile”), som banorna byggs utav, ska se ut. Från början lagrades alla ”tiles” med siffror i stigande ordning (bild till höger). Banan sparades sedan som en textfil i ett rutnät-liknande vis. Varje rad i textfilen blev en rad på banan och varje siffra blev en kolumn i banan (bild 2 till höger). Detta funkar bra, men detta system har en stor nackdel, man kan inte expandera sitt ”tileset” utan att förstöra de tidigare banor som man redan har gjort. Om man lägger till fler ”tiles” kommer alla siffror på den andra raden att ändra sig och därmed kommer banorna att peka på helt andra ”tiles” än vad de gjorde när de skapades. Detta problem löstes genom att ge varje ”tile” en koordinat på formen (x, y) istället för ett index. Alltså övre vänstra hörnet skulle ha koordinaten (0, 0), den bredvid till vänster skulle vara (1, 0) etc. Den enda nackdelen med detta system är att textfilerna nu måste lagra två siffror per ”tile” istället för endast en.



Banskaparprogrammet som man fick med var inte så bra gjort, det fanns inga intuitiva menyer, allt för kortkommandon, det var omöjligt att göra någonting utan ha ett fuskpapper. Det fanns inga verktyg, det enda som fanns var att placera ut block och att ta bort block. Programmet var dessutom dåligt programmerat för att den var anpassad till det gamla sättet att spara banor på samt att man inte kunde expandera funktionaliteten på ett enkelt sätt. En sådan grej som man inte kunde göra var att lägga till en ny rad på sin ”tilesets”, programmet fungerade bara om man hade två rader. Ett nytt program skrevs där alla dessa problem fixades, nytt gränssnitt som var intuitivt, anpassat för de nya systemet att spara banor på och nya funktioner som underlättar byggandet av banor. En sådan funktion var att man kunde markera flera block och flytta runt på dem. Innan var man tvungen att placera ut varje block var för sig om man ville flytta på stora områden.

### **5.2.2 Klasser för banor**

Det finns en klass för varje bana som man spela (mer om klasstrukturen på 5.3.1). Det som man fick direkt ifrån tutorial:en (3) hanterade dessa klasser på ett mindre optimalt sätt. I dessa klasser finns det funktioner för hur alla objekt ska ritas ut, hur de ska uppdateras, vart de ska sättas ut någonstans etc. Problemet var att det var mycket av samma kod på alla dessa klasser som gjorde exakt samma sak. Att det är lite mer text som man måste spara är inte det som var huvudproblemet. De största problemen med detta var att det såg väldigt oorganiserat och fult ut samt att det är väldigt träligt att ändra på någonting. Om man ska ändra hur en fiende uppdateras generellt för alla banor måste man in och ändra det på alla filer som man har gjort. Detta löstes genom att göra en superklass som alla banor kommer att ”extenda” (alltså att rätt så mycket låna kod från superklassen). I superklassen skrevs alla saker som alla banor har gemensamt, t.ex. att uppdatera och rita ut alla fiender. Det enda som nu finns i klasserna för banorna är saker som; vilket ”tileset” den ska ha, vart alla fiender ska vara någonstans etc. Nu om man ska ändra någonting grundläggande så behöver man endast ändra det på ett ställe, i superklassen.

## **5.3 Förklaringar**

### **5.3.1 Klasstruktur**

Detta är en illustration på hierarkin av GameState-klasserna. Ett GameState är någonting som säger vad som ska hända och hur allting ska hanteras. I t.ex. MenuState står det hur en meny ska uppdateras och hur den ska ritas och i LevelState står det hur en bana på själva spelet ska uppdateras och ritas. GameState är abstrakt och innehåller endast en update-funktion och en draw-funktion. För att hantera vilken state som är aktiv och för att kunna byta mellan dem finns det en annan klass som heter GameStateManager. Denna klass har tillgång till alla states och har funktioner för att hantera dem. När huvudspelfönstret ska rita ut och uppdatera den nuvarande aktiva state:en går den till GameStateManager och kallar på exempelvis funktionen draw. GameStateManager i sin tur kollar vilken state som är den aktiva och kallar på draw på den.

### **5.3.2 Avståndsformler**

För att beräkna hastigheter och sträckor med acceleration finns det inom fysiken dessa formler:

Där s är sträckan i meter, v0 är starthastigheten m/s, v är hastigheten i m/s, a är accelerationen i m/s2 och t är tiden i sekunder.

Dessa formler utgår ifrån att man är i den riktiga världen där all rörelse sker kontinuerligt. I spel är det inte så, där är all rörelse diskret, alltså massor med små teleporteringar. Detta betyder också att det inte finns några halva tidsenheter, det kan endast förekomma hela positiva tidsenheter. Tidsenheten i spelet kallas för ”ticks”. Formeln testades med några enkla värden på a och v0. S(t) plottades på en grafräknare där den testades hur långt man ”hoppade” mellan varje tidsenhet, alltså

.

Detta betyder ”sträckan – sträckan innan”, alltså så mycket man ”hoppade” från den tidigare tick:et till den nuvarande. ”hoppen” från spelet testades fram i själva spelet. I spelet funkar det genom att först addera accelerationen till hastigheten och sedan ”hoppa” så långt. Vid t=0 har ingenting hänt än, så då är hoppet 0, vid t=1 har accelerationen adderats en gång, vid t=2 har den adderats 2 gånger totalt, osv.

Nedan följer en tabell med hur stora alla hopp är vid en viss tidpunkt med olika värden.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Vart: | a | v0 | y(0) | y(1) | y(2) | y(3) | y(4) |
| formel | 1 | 0 | 0 | 0.5 | 1.5 | 2.5 | 3.5 |
| spel | 1 | 0 | 0 | 1 | 2 | 3 | 4 |
| formel | -2 | 8 | 0 | 7 | 5 | 3 | 1 |
| spel | -2 | 8 | 0 | 6 | 4 | 2 | 0 |

I det första fallet ser vi att formeln alltid är 0.5 mindre, och i det andra exemplet är formeln alltid 1 mer. Denna skillnad visar sig vara . För att få den totala sträckan adderar man alla steg,

I det första exemplet kommer formeln säga att den totala sträckan blir 8, medan spelet säger att det blir 10. Vi ser att vi ”hoppar” gånger där vi vid varje steg är fel från det riktiga svaret. För att kompensera för detta kan vi addera till den ursprungliga formeln.

Dessa formler funkar nu till detta spel och är de som har använts för att räkna ut diverse saker. Ett exempel är för att kolla vilka begynnelsehastigheter man behöver för att kunna dubbelhoppa en viss höjd på ett bättre sätt an vad som fanns innan. Det andra hoppet var alldeles för litet jämfört med det första, så erfarne Simon Ernofsson Spengs sade att det skulle bli bra ifall dubbelhoppet var två tredjedelar av det första hoppet. Detta ger att:

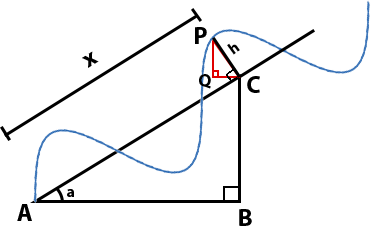
Där y är den totala höjden för hoppen och x är höjden på det första hoppet. Eftersom den nuvarande höjden var bra användes den totala höjden 102,9. Detta ger ett x-värde på 61,75 vilket blir höjden av det första hoppet, höjden på andra hoppet blir då 41,166. För att toppen av hoppet ska bli den högsta punkten måste hastigheten vid den punkten vara 0. Gravitationen är en acceleration som i detta fall är lika med -0,15 (positiv riktning är uppåt). Detta ger oss ekvationssystemet:

Där y är höjden vi vill ha för hoppet. Löser vi detta för de båda olika y-värden vi har kommer vi att få två olika v0 som kommer att vara hastigheterna vi ska utsätta gubben med för att kunna hoppa de önskade höjderna.

### **5.3.3 Sned sinuskurva**

*Detta avsnitt innefattar mycket matematik och är främst avsett för personer som är intresserade av att lära sig mer om detta.*

En annan grej som behövde åstadkommas var att kunna rita upp vinklade sinuskurvor som inte gick parallellt med x- eller y-axeln. Detta användas för att skjuta projektiler på ett stiligare sätt än att skjuta dem i räta linjer.



Bilden illustrerar en sned sinuskurva (den blå), en startpunkt A, lutningen a och sträckan X som vi har färdats på den sneda linjen. AB och QC är parallella mot x-axeln dessutom är BC och PQ parallella mot y-axeln. Detta är allting som vi vet, det som vi vill bestämma är punkten P, vilket kommer att vara positionen på sinuskurvan. Sträckan h får vi av en valfri sinusfunktion.

c, d och b är valfria konstanter. För att bestämma punkten C använder vi oss av den rätvinkliga triangeln ABC, där hypotenusan AC = x och vinkeln a är kända. Vi får längderna AB och BC genom trigonometri.

För att tillslut kunna få punkten P måste vi veta sidorna QC och PQ. Eftersom vi redan vet h måste vi veta någon vinkel annan än den räta i triangeln PQC. Det kan vi göra på detta vis:

Per definition är rät. Detta betyder att:

Utifrån detta kan vi:

Nu har vi en vinkel i triangeln, vi kan skriva den andra som:

Nu kan vi få de sista två längderna:

Nu vet vi allt för att kunna beräkna punkten P:

# **6. Resultat**

Resultatet blev ett fungerande spel i java med totalt fyra bossar (bild 1.1.2). Elever och lärare kände igen skolan och vilka bossarna skulle föreställa när de spelade spelet. Det blev en välfungerande meny med tre olika sparfiler, inställningar för att ändra kontrollerna och eftertexter där man kan se alla som var med och hjälpte till med spelet. Fyra världar skapades med en boss i varje. I vardera av de första två världarna finns det dessutom två ytterligare vanliga banor. Den första bossen är av Alexander Alsenholt som har temat av Bowser från Super Mario, denna boss har en enkel svårighetsgrad. Målet med bossen är att springa under honom för att trycka på en knapp för att sedan vinna. Nästa boss blev en av lite svårare svårighetsgrad, Mattias Karlsson som en grafräknare. På Denna boss ska man endast överleva alla attacker han skjuter mot en på tid. Den tredje bossen blev Johannes Hultin som en bosskaraktär från spelet ”Chrono Trigger”. För att vinna över Johannes ska man skriva tre olika ord som han av olika anledningar inte tycker om. Den sista bossen är en tvåfasig boss där Alsenholt är tillbaka för revansch. Här är temat ”helvetet” och denna boss är väldigt svår. I den första fasen ska man undvika attacker på tid och på den andra fasen ska man slå på två stycken svävande händer för att göra skada. När man klarar den sista bossen räddar man Moa, beroende på hur många gånger man har dödat den sista bossen innan så kommer dialogen med henne att vara annorlunda.

Det visade sig att det var ganska enkelt att göra ett spel från grunden med en tutorial. Det tar dock längre tid och man får inte mer avancerad funktionalitet som kan finnas på färdiggjorda spelmotorer. Detta simpla 2D-spel är inte tillräckligt för att avgöra ifall java är dåligt eller bra att göra spel med ty denna typ av spel inte alls är krävande på datorn. Man skulle ha kunnat göra detta spel i vilket språk som helst och de hade inte varit någon skillnad prestandamässigt. Skillnaden skulle märkas ifall spelet skulle ha varit i 3D med avancerad grafik. Dock är det enklare att göra spel i java än i andra språk ty java är ett enklare språk generellt. Det finns ”garbage collector”, inga pointers och massor med inbyggda bibliotek, inkluderat ett för att göra fönster. Så man behöver inte installera tredjeparts program som man behöver för att programmera ett GUI i C++.

Alltså, helt prestandamässigt är detta projekt inte tillräckligt för att avgöra det. Det är enklare att programmera än i andra språk. Så java är helt dugligt, i alla fall om man ska göra ett spel med icke-krävande grafik.

# **7.** **Diskussion/analys**

Jag tycker att spelet blev väldigt bra. När projektet först planerades var tanken att väldigt många fler bossar skulle finnas, jag totalt underskattade tiden det tog för att göra en boss, så väldigt mycket är ändrat. Tanken var att det skulle finnas kanske 10 bossar med 2 banor till varje, det skulle sluta med Omar och det skulle utspela sig på hela skolan. Det slutade med att det blev 4 bossar med endast 4 eller 5 ytterligare banor. Slutbossen blev inte en 3-fasig Omar, det blev Alex som kom tillbaka för revansch. Det var endast den översta våningen som kom med som ”overworld”. Att göra en boss tar lång tid, man måste rita nya sprites, planera alla attacker och faser in i minsta detalj, det tar lång tid att testköra bossen och balansera den samt så hittar man alltid på någonting oplanerat som man vill göra eller något problem som tar väldigt lång tid att lägga till eller fixa. Det var speciellt faktorn om vad bossen skulle göra som gjorde att det tog längre tid än vad jag först hade trott. Det var väldigt svårt att hitta på någonting från fantasin som bossen skulle göra, mycket tid gick just åt att tänka ut vad som skulle hända. Om jag hade varit mer erfaren i denna sorts spel så hade det nog gått snabbare med mer varierande attacker, så en eller två bossar till hade nog kunnat göras. Jag var dessutom väldigt omotiverad och trött på att göra spelet vid vissa tillfällen, så om jag inte hade varit det hade jag kunnat göra mer.

Om jag skulle ha gjort någonting annorlunda eller om jag ska göra ett nytt spel i framtiden så skulle jag nog utgå ifrån en annan tutorial (3). Den var inte så bra, videorna som han gjorde var mer som en ”code along” än någonting som skulle visa hur man gjorde. Han förklarade väldigt lite och skrev ganska många fel. Ett exempel på en grundläggande sak som han inte förklarade var hur positionerna för alla objekt var någonstans på objektet. I vanliga fall brukar man utgå ifrån att positionerna ligger i övre vänstra hörnet, men han hade placerat dem i mitten av objektet, och förklarade inte att han hade gjort det. Det var lite jobbigt. Det är dessutom ett avrundningsfel i ”main”-loopen. Det är tanken att den ska köras i 60 gånger per sekund, men den körde 63 gånger per sekund. Jag upptäckte detta försent när redan 3 bossar var gjorda som berodde väldigt på uppdateringshastigheten, så det blev kvar. En annan sak som inte är så bra är att det är väldigt svårt att ändra storleken på själva fönstret (ändra upplösningen). Om man gör det kommer all grafik att se väldigt konstig ut för att algoritmen för att ändra storlek på saker är väldigt simpel. Man kunde ändra hur den gjorde för att ändra storlek, men då blev allting suddigt istället. Detta är något som jag antar skulle bli bättre om man använde en välgjord spelmotor. Det fanns inte heller ett enkelt sätt att göra kollisioner med saker som inte var ”tiles” som låg i rutnätet. Han hade inte gjort det och jag kunde inte komma på ett bra sätt att göra det på, detta skulle också kunna vara något som en bra spelmotor hade kunnat göra bättre. Men utöver dessa saker så tyckte jag att allt blev bra i slutändan och allt var bra strukturerat så det var enkelt att förstå. Om jag ska göra ett nytt spel kommer det nog att vara väldigt likt denna i hur allt är uppbyggt, men allting som jag har klagat på skulle jag försöka fixa.

Annars finns det några designval som jag skulle ha gjort lite annorlunda. Det enda som jag har att klaga på här är att jag skulle ha gjort gubben mindre och spelfönstret större så att jag kunde få plats med fler projektiler.

Jag är väldigt nöjd med arbetet och tyckte att det för det mesta var väldigt kul att göra.

# **8. Referenser**

1. **Java (programspråk). *Wikipedia.* [Online] https://sv.wikipedia.org/wiki/Java\_(programspr%C3%A5k).**

**2. Why are there so few games written in java. *Stackoverflow.* [Online] http://stackoverflow.com/questions/1034458/why-are-only-a-few-video-games-written-in-java.**

**3. ForeignGuyMike. Programming platformer tutorial. *Youtube.* [Online] https://www.youtube.com/watch?v=9dzhgsVaiSo.**

# **9.** **Appendix**

|  |  |
| --- | --- |
| (bild 1.1.1)  https://i.ytimg.com/vi/7qh_z_ziYP8/maxresdefault.jpg  I Wanna Kill the Kamilia 2, sista bossen | (bild 1.1.2)  Bild på den första bossen av detta projekt |
| (bild 1.2.1)  http://mp1st.com/wp-content/uploads/2012/01/Call-of-Duty-Black-Ops-Escalation-Screens-Screenshots-10-640x360.jpgCall of Duty, ett 3D-spel. | https://gaminghistory101.files.wordpress.com/2012/03/pandemonium.jpg(bild 1.2.2)  Ett exempel på ett ”Bullet Hell”-spel. |