LINKOPING UNIVERSITY

Designspecifikation

Rockblock II

Axel Blackert
Alexander Ernfridsson
Oskar Eriksson
Oscar Fredriksson

2014-10-08

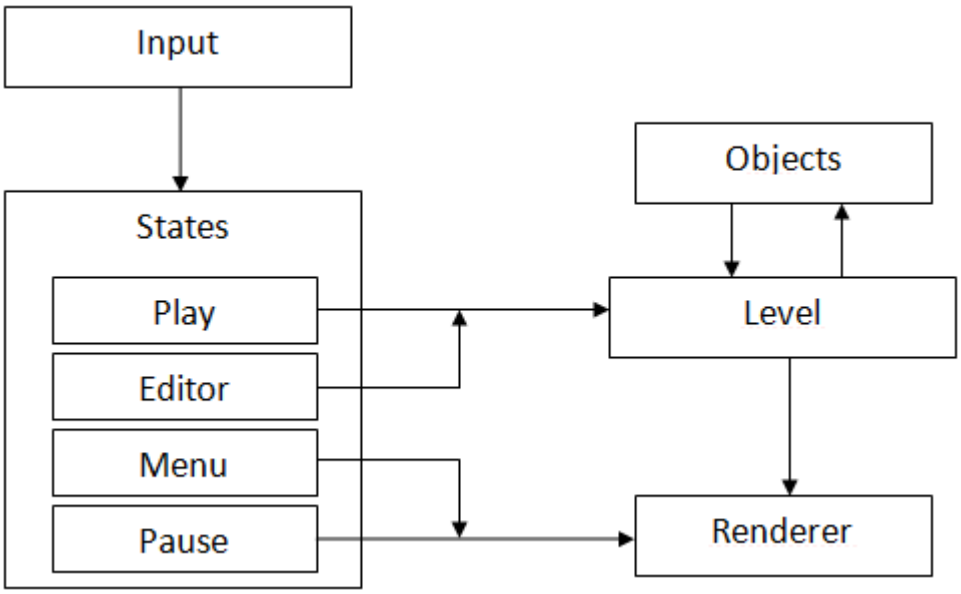
Introduktion

Detta är en designspecifikation av spelet Rockblock II. Rockblock II är ett plattformsspel i 2D grafik. Detta dokument är en genomgående specifikation av designen av spelet utifrån kravspecifikationen. Dokumentet innehåller en detaljerad beskrivning av samtliga moduler med deras funktioner, klasser och variabler samt spelets egna filformat.

Dokumentet beskriver med bilder och text hur modulerna hänger ihop och integrerar med varandra.

Innehållsförteckning

Övergripande modulkontrakt

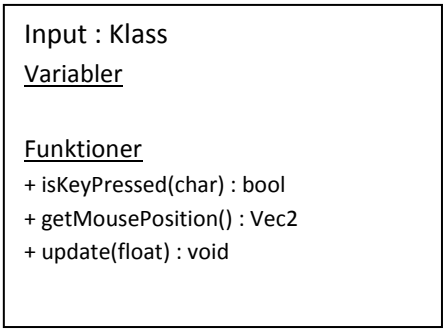


Detta är de modulkontrakt som programmet kommer följa. ”Input” ger de värden som avgör vad programmet ska göra härnäst. Programmets start-”State” kommer att vara ”Menu”. Beroende på vilket ”State” programmet befinner sig i kommer olika operationer vara tillgängliga. Då programmet är i ”Menu” eller ”Pause” behöver vi endast visa användargränssnittet och vara redo för ny ”Input”.

Då programmet är i ”Play” eller ”Editor” behöver vi ha tillgång till ”Level” för att ha tillgång till alla de objekt som spelet ska hantera som finns i ”Objects”.

I ”Play” så styr ”Input” var spelaren ska flytta sin karaktär som är lokaliserad i ”Level”->”Objects”.
I ”Editor” styr ”Input” placering och modifiering av objekten på kartan.

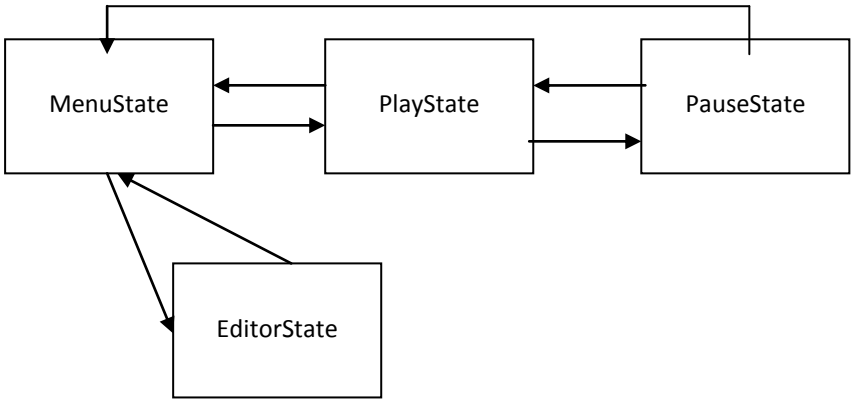
Input

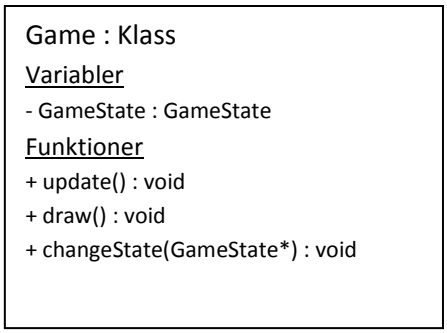


Input ska hantera input från SDL och finnas tillgänglig i alla olika states.

States

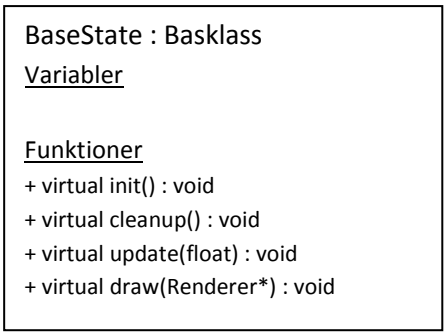
Flödesschema över de olika basestates:





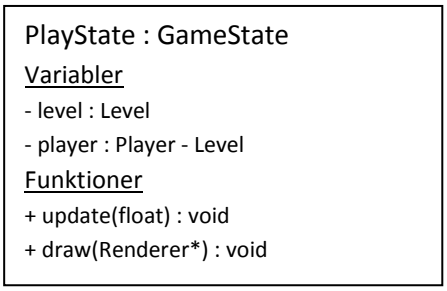
Det är Game som ska hantera den nuvarande staten och ansvara för att initiera och rensa upp den när den byts. Den innehåller även själva game-loopen som ska köras med cirka 60FPS. I game-loopen anropas den nuvarande statens update() och draw() funktioner.

BaseState



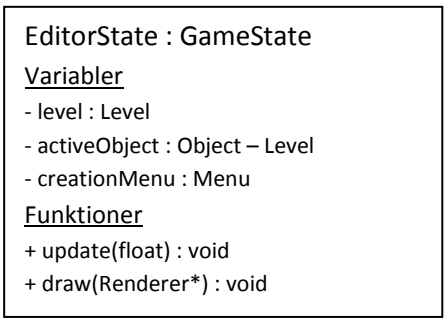
Spelet kommer kunna befinna sig i flera olika states, varje state representeras av en instans från GameState. PauseState, PlayState och MenuState ska alla härleda från GameState och innehålla de datamedlemmar som behövs. Egna versioner av medlemsfunktionerna ska också göras för varje unik state.

PlayState



PlayState är aktiv när man spelar spelet och är inne på en karta. Den har en Level med alla objekt och en pekare till Player objektet. Det är i PlayState inmatning hanteras och styrning av spelaren sker.

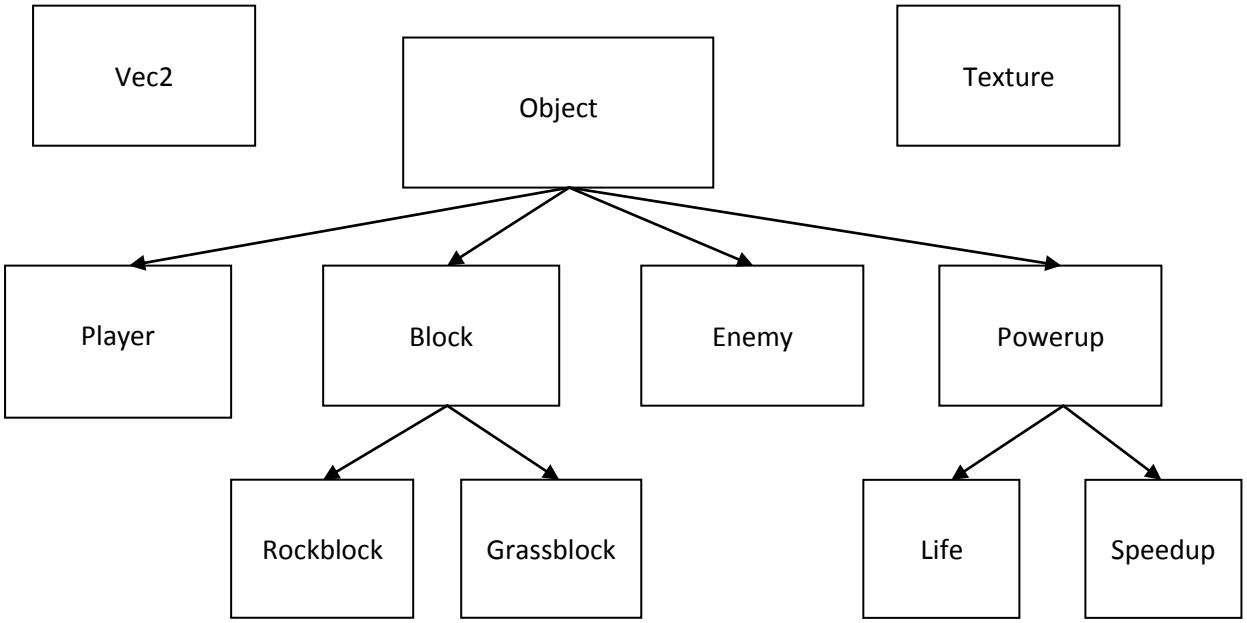
EditorState



EditorState har en Level och en pekare till ett objekt från Level som motsvarar det markerade objektet. Ifrån creationMenu ska man välja olika typer av objekt och placera ut dem på kartan.

Objekt

Objekthierarki



Här ser vi ett blockdiagram på klasshierarkin för Objectklassen.

Object

Object : Basklass

Variabler

- id : ObjectId
- texture : Texture
- position : Vec2
- width, height : float

Funktioner

+ getPos() : Vec2
+ getWidth() : float
+ getHeight() : float
+ getId() : ObjectId
+ getTexture() : Texture
+ virtual update(float) : void
+ virtual draw(Renderer*) : void
+ virtual handleCollision(Object*) : void

Object är basklassen för alla objekt som kommer finnas på kartan. Den innehåller medlemmar och funktioner som är gemensamma för alla de härledande objekten.

Funktionerna update() och draw() är virtuella så det är genom dem som man definierar hur de olika typerna av objekt kommer bete sig.

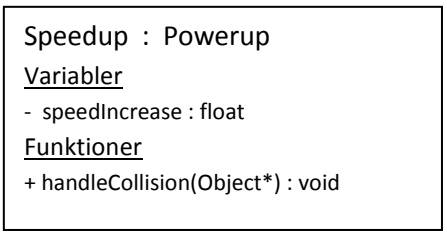
Player

Player är det objekt spelaren styr på kartan. Input som avgör hur spelaren ska röra sig hanteras i PlayState där funktionera jump() och move() anropas.

Enemy

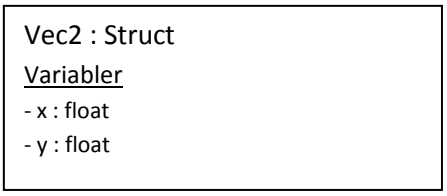
Fiender ska patrullera framåt och tillbaka på kartan. Medlemsvariablerna startPos och endPos är ändpunkterna där den ska vända håll.

Speedup



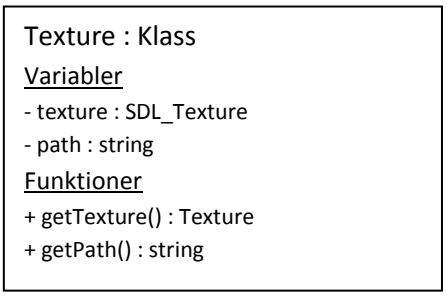
Speedup är en powerup som vid kollision ger spelaren ökad hastighet under en kort period.

Vec2



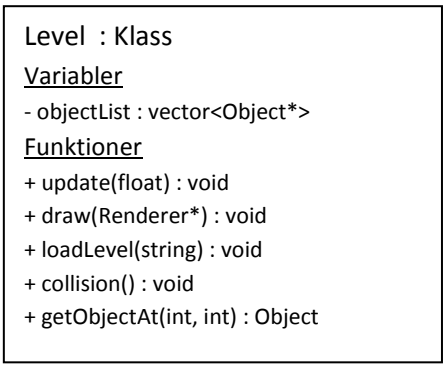
Vec2 ska vara en liten struktur som bara innehåller x och y. Istället för att hantera x och y separat i koden ska Vec2 användas där det passar. Objektens position representeras t.ex av en Vec2 medlem. Det är även ett argument av typen Vec2 som Renderer tar i sina draw-funktioner.

Texture



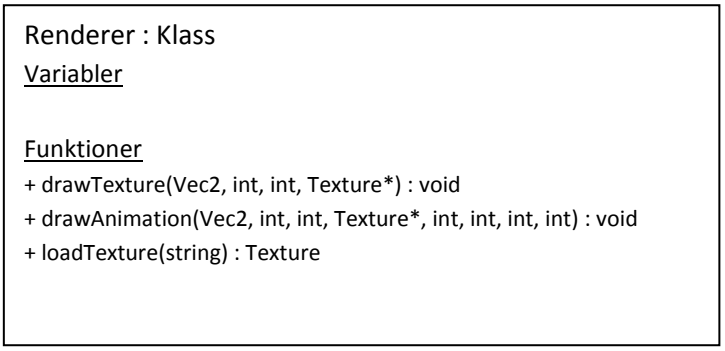
Texture är en liten wrapper klass runtom SDL_Texture.

Level



Innehåller en lista över alla objekt härledda från Object. Det är denna lista som utgör vad som finns på kartan och när spelet börjar fylls listan med objekt skapade från den lagrade datan i sparfilen. Den har en Update och Draw funktion där objekten i listan itereras över, och deras egna Update och Draw funktioner anropas. Level ansvarar också för kollisionstesterna mellan spelaren och alla andra objekt.

Renderer



Klassen Renderer hanterar utskrift på skärmen och inladdning av texturer.

Filformatet

När man skapat en bana i editorn ska alla objekts attributer sparas till en fil. Det är från denna fil banor laddas när man ska spela dem. Alla objektens attributer läses in och objekt placeras ut på banan utifrån deras sparade positioner och typ.

I filen ska även koordinater på spelarens start position och banans mål finnas.

Olika värden i filen skiljs med mellanslag och varje rad förutom de två första motsvarar ett sparat objekt.

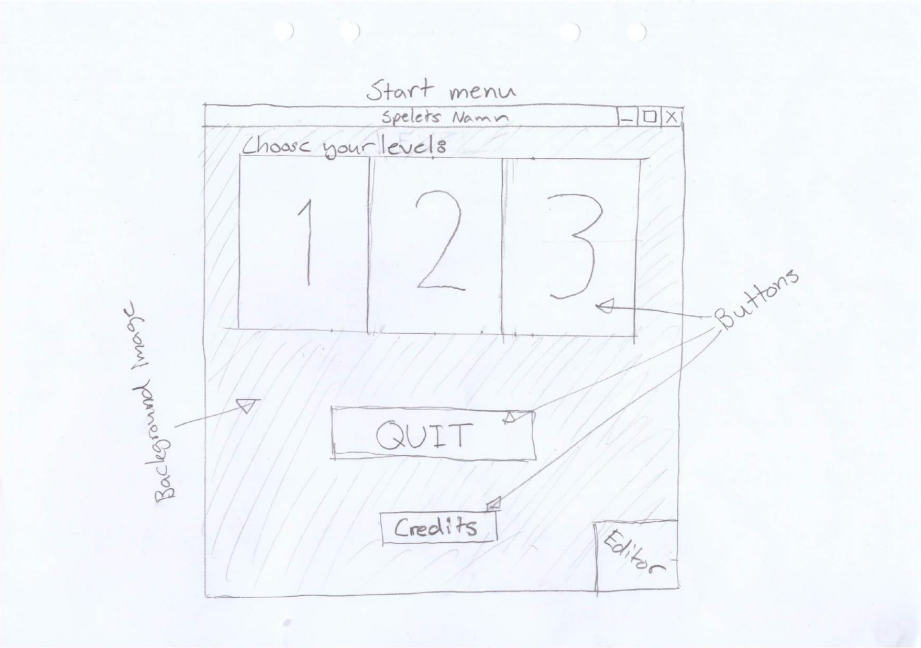
```
start_x start_y
end_x end_y
type x y width height texture
type x y width height texture
type x y width height texture
```

Vissa objekt som t.ex Enemy måste spara mer data till filen. Start position, slut position och hastighet för de fiender som patrullerar.

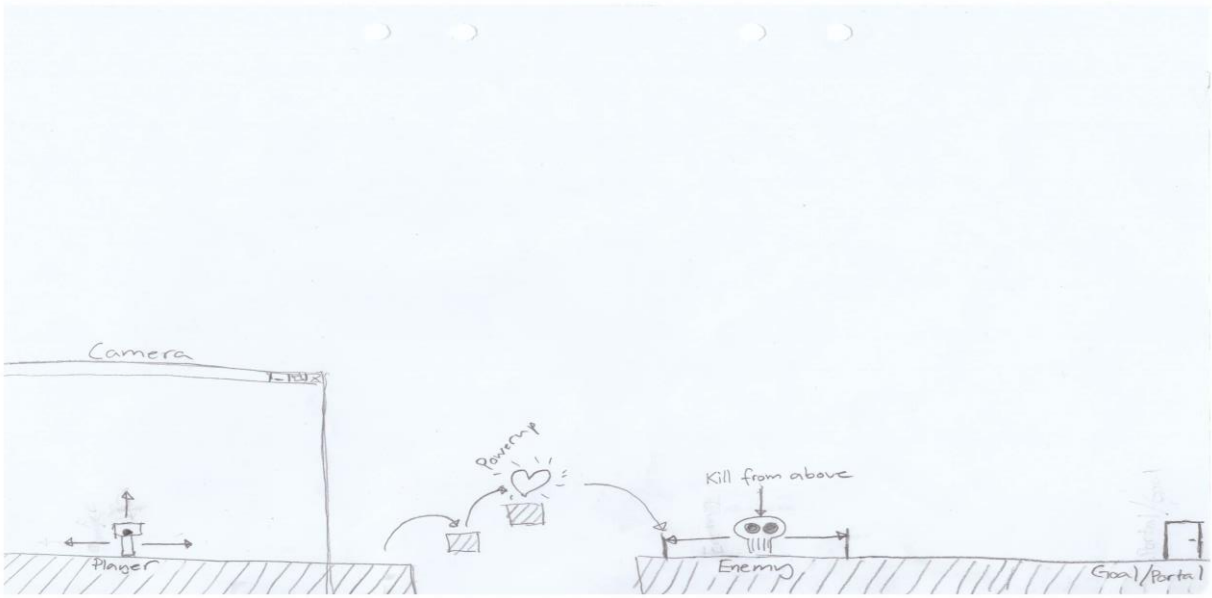
```
type x y width height texture start_x start_y end_x end_y
```

Det är klassen Level som har funktionerna loadLevel() och saveLevel() som hanterar inläsning och utskrivt till sparfilerna.

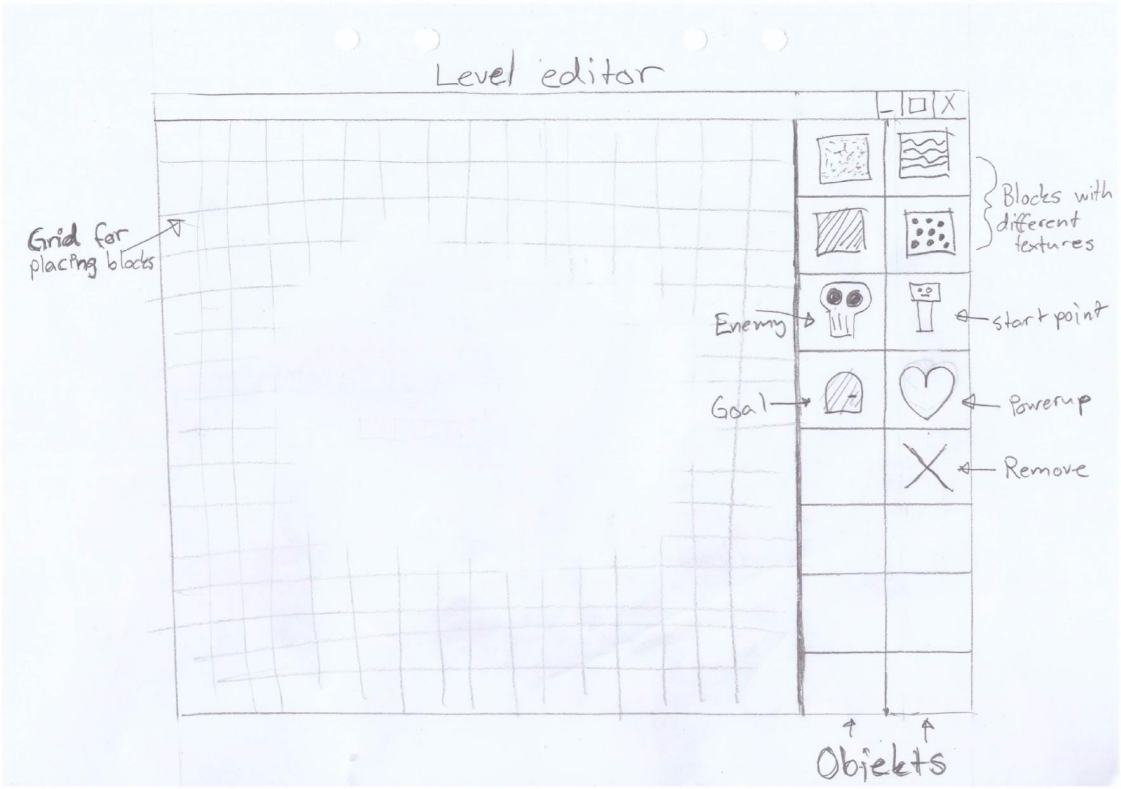
Användargränssnitt



Bilden illustrerar det gränssnitt vår meny ska ha. Härifrån kan du välja banor, avsluta, gå till editorn och visa information om skaparna. Menyn kommer också ha en bakgrundsbild. De olika alternativen i menyn kommer att representeras av knappar med text på. Knapparna kommer att ha en enkel animering vid nedtryckning.



Denna bild illustrerar hur spelet kan komma att se ut när man spelar en bana. Man ser spelaren ifrån sidan som i andra klassiska 2D plattformsspel.



Detta är level editorn där man kan skapa sin egen bana. Till höger ser du de objekt som kan placeras ut på banan. Här ska du också kunna testa banan direkt, spara banan, ta bort objekt, samt kunna gå tillbaka till menyn.

Bilagor

1.) Kravspecifikation

2.) Idéskisser 1-3

Björnligan

Kravspecifikation

Project TDDI02

Skapades datum: 11-09-2014
Senast ändrad: 18-09-2014

Gruppmedlemmar:

Blackert Axel (axebl383)
Eriksson Oskar (osker098)
Ernfridsson Alexander (aleer760)
Fredriksson Oscar (oscfr819)

Bilagor:

Appendix med tre skisser.

Introduktion

Denna kravspecifikation är ett dokument som ska exakt förklara hur Björnligans projekt i programmering 2014 har planerats och skall utföras. Detta dokument kommer agera som underlag vid vidare beslutstagning då projektet börjar utvecklas.

Definitioner och förkortningar

I denna kravspecifikation använda följande definitioner:

- Bana/spelplan/karta/map/level – Syftar på den yta som spelaren kommer att röra sig på.
- Spelare – Syftar på den karaktär användaren kommer kontrollerar på spelplanen.
- Plattform – Syftar på terräng på spelplanen.
- Kollision – Syftar på då objekt i spelet krockar.
- Fiende – Syftar på de datorkontrollerade karaktärer som jobbar för att döda spelaren.
- Liv/dör – Syftar på spelarens tillstånd i spelet. Spelaren dör då den träffas av t.ex. en projektil från en fiende.
- Hjälpmedel – Syftar på föremål på kartan som ger speciella effekter när spelaren plockar upp dem. T.ex. hjärta ger liv och skor ökar hastigheten.

Projektets syfte

Målet med detta projekt är att skapa ett fungerande spel i samma anda som de gamla Super Mario- spelen. Då alla inblandade i projektet spelat dessa spel då de var yngre ansågs det som ett bra val att få förnya detta spel i vår egen bild.

Målen med att göra detta spel är:

- Få en djupare förståelse för C++.
- Få en större bild av hur det är att jobba inom mjukvara-utveckling.
- Lära oss jobba mot ett gemensamt mål som projektgrupp.
- Lära oss om processen för projektutveckling.

Krav

Skall-krav

S1. Fysikmotor (Box2D)

- S1.1. Spelaren skall kunna röra sig framåt, bakåt, hoppa.
- S1.2. Kollision skall existera i spelet. Detta innebär att kollision med plattformer, fiender etc. skall hanteras av motorn.

S2. Det ska finnas olika typer av objekt på kartan

- S2.1. Fiender som patrullerar
- S2.2. Hinder
- S2.3. Hjälpmedel som underlättar för spelaren

S3. Level-editor. (Level-skapare)

- S3.1. Ett separat läge då användaren skapar egna banor.
- S3.2. Spelaren skall kunna placera ut objekt på kartan, så som fiender, hinder, hjälpmedel och plattformar.
- S3.3. Banorna skall kunna sparas och laddas från en separat fil med eget filformat.

S4. Ett användargränssnitt som innehåller:

- S4.1. Startmeny där man väljer karta att spela
- S4.2. Pausmeny.
- S4.3. Level-editor.

S5. Då spelaren ”dör” i spelet så skall banan laddas om från start. Spelaren har endast 1 liv.

S6. En komplett och spelbar bana skall finnas från spelstart.

S7. Bakgrundsmusik skall finnas från spelstart.

S8. Spelet ska vara presenteras i 2D grafik med hjälp av grafikbiblioteket SDL.

S9. Spelet ska ha en smart kamera som följer spelarens rörelse utan att spelaren behöver styra denna.

S10. Spelobjekt med olika egenskaper skall finnas på banan för att ge spelaren hjälpmedel/hinder.

S11. När spelaren klarar en karta låses en ny karta upp.

Bör-krav

B1. High-scores ska sparas efter avslutat spelsession.

B2. Fiender ska få mer funktionalitet.

- B2.1. Fiender ska kunna skjuta projektiler.
- B2.2. Fiender ska kunna ha mer komplex rörlighet.

B3. Tillägg av ljudeffekter.

- B3.1. Ljudeffekt vid projektiler.
- B3.2. Ljudeffekt vid kollisoner.

B4. Lägga till fler funktioner för spelaren.

- B4.1. Spelaren bör kunna ducka inuti spelet.
- B4.2. Spelaren bör kunna använda vapen inuti spelet.

B5. Destruktion av objekt (hoppa sönder plattformar underifrån).

B6. Flytta runt vissa objekt (lådor och liknande).

B7. Fuskkoder.

B8. Plattformar som automatiskt rör på sig.

B9. Aggressiva fiender som jagar spelaren.

B10. Stegar som man klättrar vertikalt på.

B11. Trampoliner.

Osäkerheter och risker

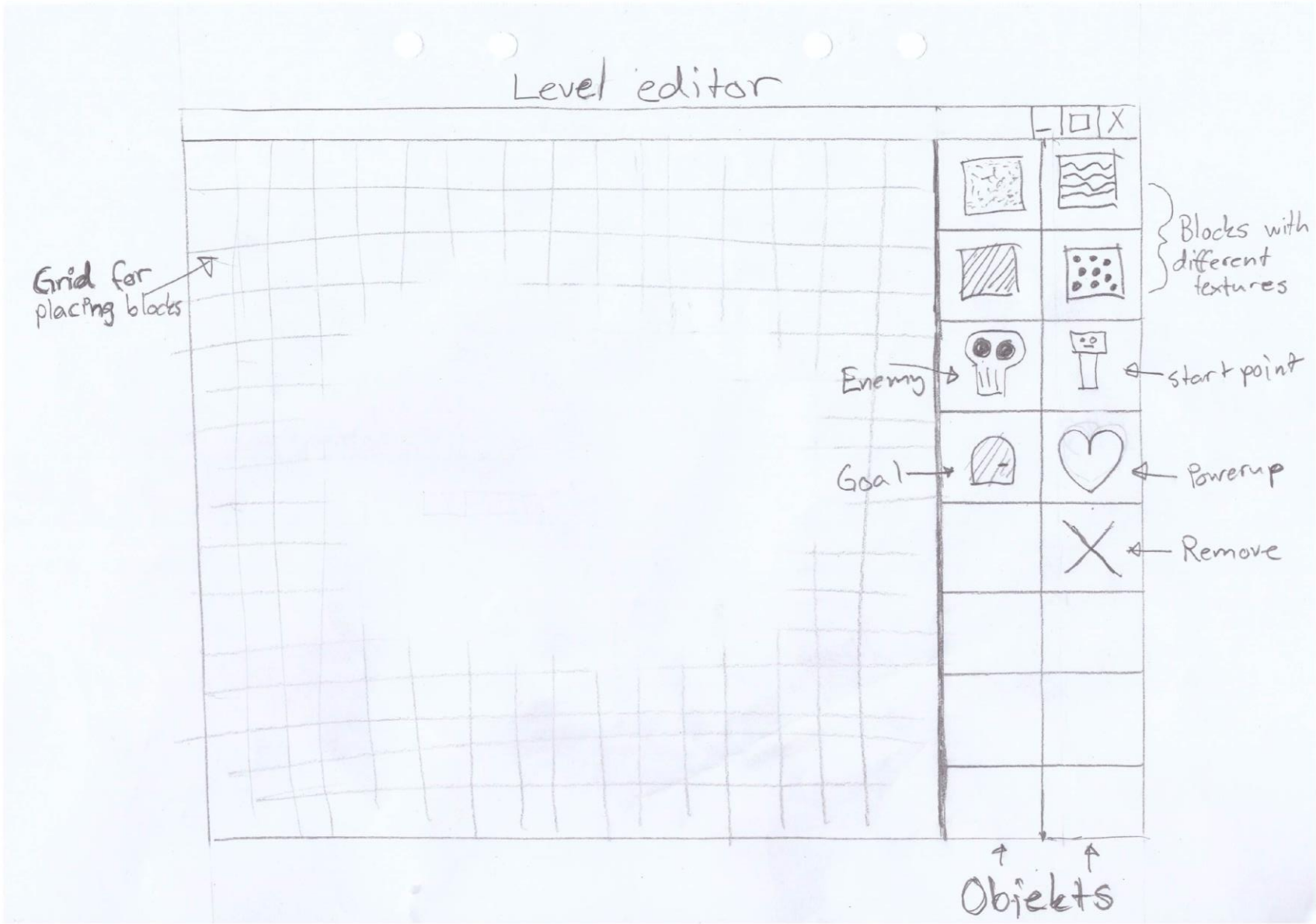
1. Inte lyckas integrera de externa biblioteken på det sättet som planerat.

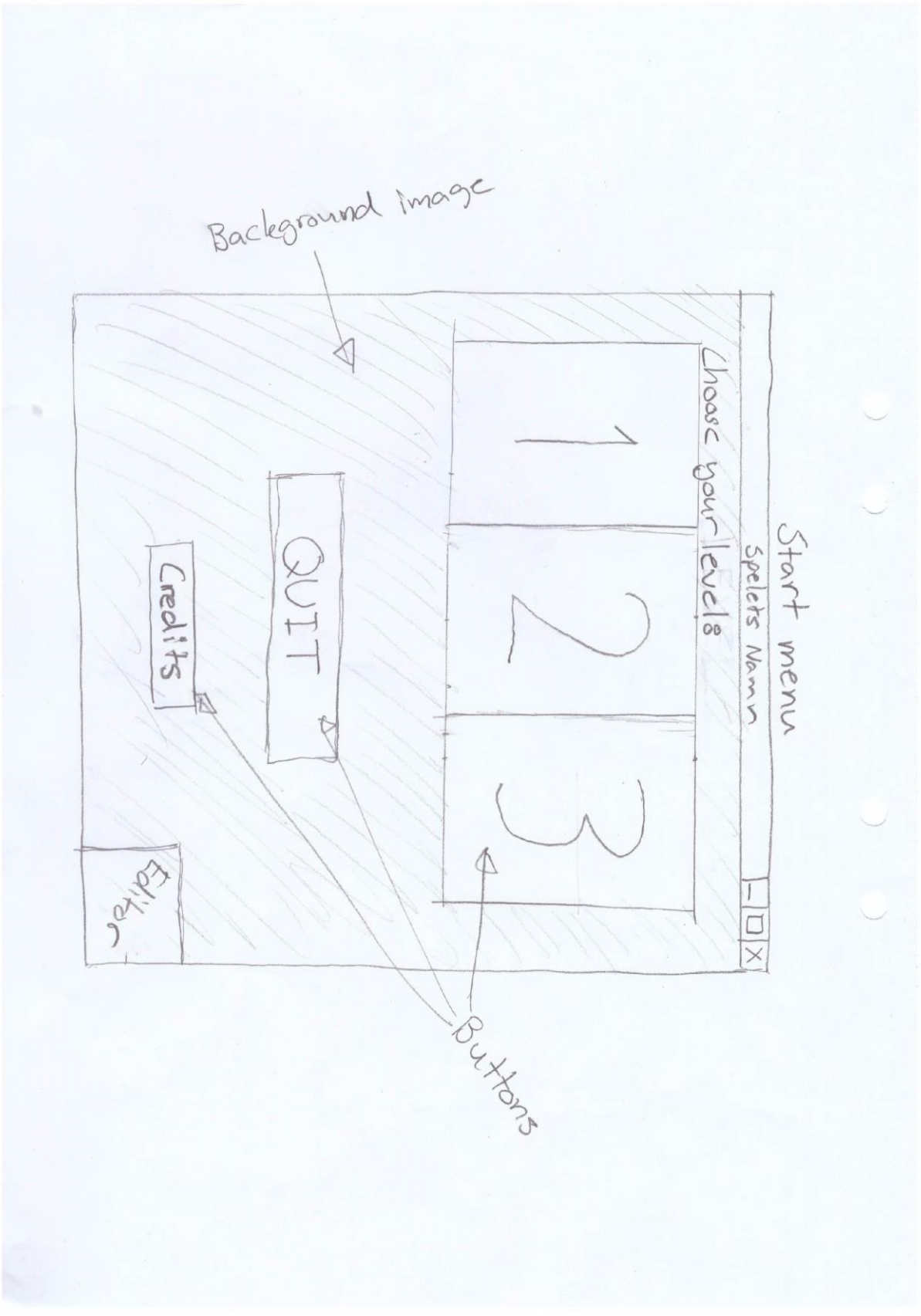
2. Brist på tid till att implementera alla funktioner.

3. En dåligt strukturerad kod bas som gör det svårt att bygga vidare på den.

4. Att spelet inte kan köras tillräckligt snabbt på alla maskiner för att få det spelbart.

Player
Camera
Powerup
Enemy
Kill from above
Goal/Portal


Level editor
Grid for placing blocks
Blocks with different textures
Enemy
start point
Goal
Powerup
Remove
Objekts


Background image
Start menu
Spelets Namn
Choose your levels
1
2
3
QUIT
Credits
Editor
Buttons