Designdokument levelleditor

Bomb Defuser

version: 1.0.0

2015-05-19

Handledare: Edward S. Blurock

Grupp 18

Revision History

Date	Version	Description	Author
2015-04-07	0.0.0	Lagt fundament, skrivit rubriker	Anton Björkman
2015-04-09	0.1.0	Use Case Diagram, System-Diagram	Anton Björkman
2015-04-10	0.1.1	Kontext-Diagram	Mika Lehtinen
2015-04-12	0.2.0	Skrev designhistoria	Anton Björkman
2015-04-12	0.2.1	påbörjade användargränssnittet	Patrik Nilsson
2015-05-15	0.2.2	Arbetat färdigt med klass hierarki	Anton Björkman
2015-05-17	0.3.0	Läst igenom och rättat	Mika Lehtinen
2015-05-17	0.3.1	Finslipat struktur	Patrik Nilsson
2015-05-17	0.3.1	Klassdiagram	Anton Björkman
2015-05-19	0.3.1	Rättat	Peter Lindberg
2015-05-19	1.0.0	Finslip	Patrik Nilsson, Mika Lehtinen
2015-05-22	1.0.1	Kontrolläst och rättat.	Peter Lindberg
2015-05-24	1.0.1	Korrekturläst och rättat	Peter Lindberg

Innehållsförteckning

1 Inledning

1.1 Syfte

Syftet med detta dokumentet är att lägga ett fundament för projektet som vi kommer följa genom projektets gång.

1.2 Omfattning

I detta dokumentet kommer vi att skapa en övergriplig blick på designarbetet där vi bland annat går in väldigt djupt på designarbete. Vi kommer inte gå in på djupet i funktionen av specifika klasser i systemet.

1.3 Definitioner, akronymer och förkortningar

C#, Java: Två varianter av språk man kan skriva kod i, används ofta för att utveckla system och spel.
XNA, LibGDX: Två väldigt snarlika ramverk till språken C# och Java som ger användaren tillgång till förbestämda funktioner istället för att du ska behöva skriva all kod manuellt.
Level-Editor: Ett system utvecklat med syftet att förenkla skapandet av nivåer till ett angivet spel, utvecklas ofta för att öka effektiviteten i utvecklingen.
Taser: Conducted electrical weapon (CEW) är ett pistolliknande vapen som skickar iväg väldigt starka elchocker genom två pil-liknande elektroder.
Tiles: Ett system av rutor som vi bygger upp spelet med, ofta om man talar om "tile-based" games så är spelvärlden uppbyggd i rutor av förbestämd storlek.
AI: (Artificial Intelligence) Är i korta drag intelligensen som uppvisas av en maskin eller mjukvara.
Android: Ett populärt operativsystem för smarttelefoner, detta är systemet vi utvecklar spelet till.
Icke-funktionella krav: Är ett krav som inte är direkt relaterat till vad systemet ska göra
Funktionella krav: är ett krav som beskriver vad systemet ska göra
Pixel-art: Är en bild som är uppbyggd och redigeras på pixel-nivå. Kännetecknas ofta med aningen kantigare grafik.

1.4 Referenser

Vi har använt oss av den vägledande dokumentation som funnits tillgänglig för oss studenter som till exempel mallar och exempel på tidigare studenters arbete. Vi har också använt oss av dem PDF-er vi fått tillgång till genom föreläsningarna genom kursens gång. Och slutligen så har vi använt oss väldigt flitigt av Roger Pressman's *Software Engineering, a practitioner's approach 8th edition.*

Programspråk, ramverk och annan mjukvara som varit av centralt värde för utvecklingsprocessen är utvecklade och tillhörande följande:

- www.java.com
- www.blender.org
- www.microsoft.com

1.5 Struktur

Dokumentet är strukturerat på så vis att det börjar med att djupgående förklara hur designhistoriken ser ut. Den går sedan över i mer översiktlig information med Use-case-diagram och Kontext-diagram. I slutet av dokumentet övergår det i en väldigt detaljerad representation av systemets inre struktur och uppbyggnad.

2 Designhistoria

Vi startade designarbetet genom att titta på andra leveleditors för spel som exempel CS:GO för att se vad de hade som gjorde dem bra och mindre bra. Vi såg därför väldigt tydligt vilka funktioner som en leveleditor behövde. Funktioner som krävs är ett smidigt sätt placera ut tiles och andra objekt. Även att kunna röra karmera så man kan navigera i banan man skapat. Man måste även kunna spara, ladda och kunna köra en bana för att uppfylla kraven vi har på en smidig effektiv leveleditor.

Vi kom fram till att levelleditorn inte ska vara en del av spelet, utan ett system för sig själv. Vi var även överens om att levelleditorn bara skulle kunna användas på PC. Eftersom vi delade upp oss i två olika team, där de ena teamet skulle arbeta på spelet och de andra på levelleditorn, blev valet att skriva programet i C# med hjälv av .NET och XNA. Då utvecklingsgruppen på leveleditorn hade mest erfarenhet i detta språket.

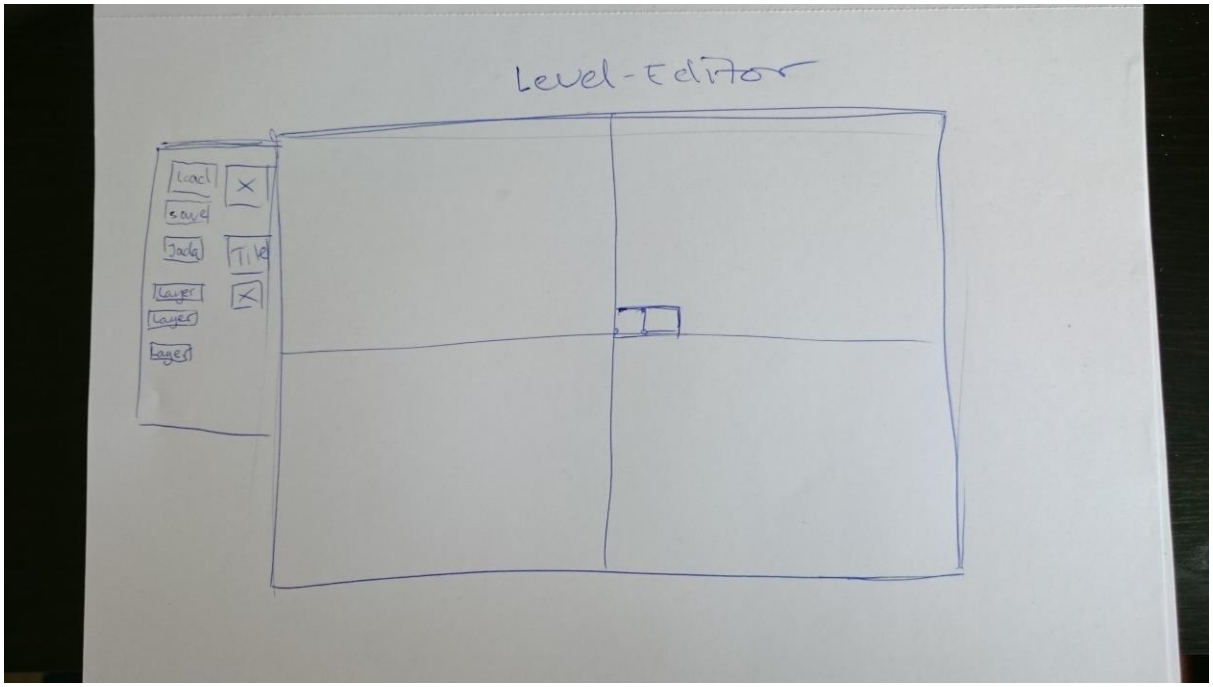
Vi bestämde oss för att göra en prototyp, där vi då kunde lära oss mer om problemen att utveckla levelleditorn. I prototypen gjorde vi användargränssnittet helt själva, det vill säga knappar och liknade. Eftersom spelet har olika lager som kollisionslager, bakgrundslager och överlager detta gjorde att vi som hel grupp fick diskutera vilka olika objekt som skulle kunna vara i de olika lagern. När vi arbetat klar vår första prototyp hade vi ett program där man kunde placera ut olika objekt men man kunde inte välja vilka lager objekten skulle ligga i. Man kunde manipulera objekten göra dem större eller mindre, flytta en eller flera samtidigt med hjälp av väljverktyget. Vi utvecklade även en kamera så att man kunde navigera sig fram, ner, upp och ner, med detta uppstod ett problem som vi redan kände till och det är att när man flyttar kameran så ändras matrisen och därför blev det att kod som vi hade skrivit för att kunde välja objekt inte längre fungera.

Med nya kunskaper om hur kameran fungeraroch hur den påverkar programmet gjorde detta att vi skrotade den första prototypen och påbörjade en ny som skulle likna den riktiga versionen. Vi märkte att den strukturen vi hade valt inte var helt optimal, därav blev valet att göra en ny prototyp. I den andra prototypen hade vi den första som en utgångsmall och ändrade på det som vi visste skulle bli ett problem, som till exempel med kameran. Vi såg till att det fanns två kameror där en var på själva världen där vi skapade objekten och en kamera

som vi hade på användargränssnittet. Efter att ha gjort menyer och knappar åter en gång insåg vi att använda oss .NET istället skulle underlätta och skulle inte ta så mycket tid heller. I den andra prototypen kunde vi göra allt som vi kunde i första prototypen men däremot var allt fixat så att objekten kunde väljas även då man hade flyttat på kameran. Man kunde även lägga objekt i olika listor/lager. Men där fattades funktioner som att kunna spara, ladda och köra en mapp.

Med detta i bagaget gick vi vidare mot den riktiga versionen, och eftersom som vi kände att vi ville göra egna knappar och användargränssnitt, gjorde vi den riktiga versionens knappar med .NET. Sen därefter var det bara att bygga programmet efter föregående prototyp. Dock återstod det att spara, ladda och köra en bana som nyss gjorts. Eftersom denna textfil skulle tolkas i spelet för att kunna bygga upp banan gjorde vi detta sist. Med hjälp av medlem från spelteamet fick vi hjälpas åt så att textfilen sparades korrekt. Efter detta återstod det bara att köra den mappen som sparades och det gjordes med hjälp av spelteam också.

3 Användargränssnitt



Denna bilden representerar det enda utkastet vi har på användargränssnittet. Sidomenyn kommer användas för att navigera i levelleditorn, där väljer du vilka lager du vill använda

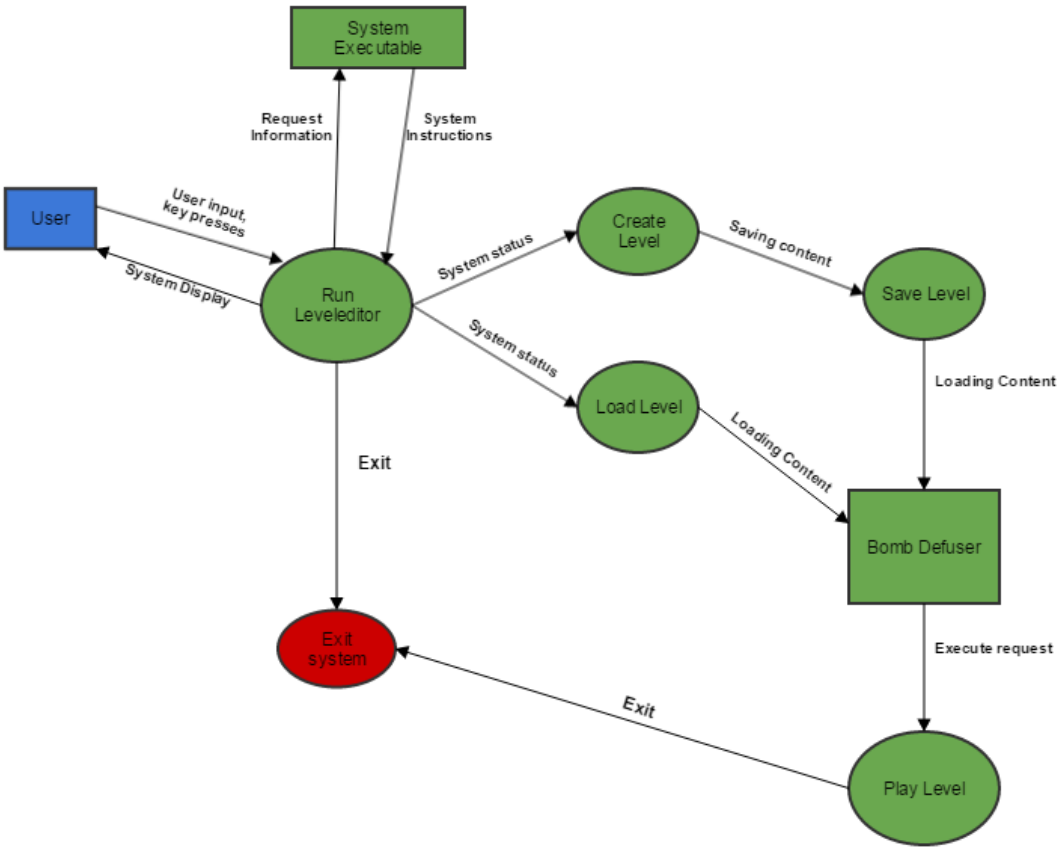
samt vilka tiles du kommer att använda. De utritade axlarna på skärmen är till för att representera mittpunkten av skärmen.

I olika objekt skall man kunna få upp en egenskapsruta där man kan redigera värden för olika objekt. I tilesen skall man till exempel kunna ändra till en exakt position samt ändra storlek.

När man skapar en ny bana skall det alltid finnas en startposition för spelaren och bomb. Dessa kan man heller inte ta bort, man kan endast flytta deras position. Detta gör varje nivå måste ha en spelare och en bomb.

Fläktarna har även en egen egenskapsruta som man kan välja hur starkt fläkten skall blåsa, åt vilket håll fläkten skall blåsa, samt hur många fläktar det skall finnas.

4 Kontext-diagram



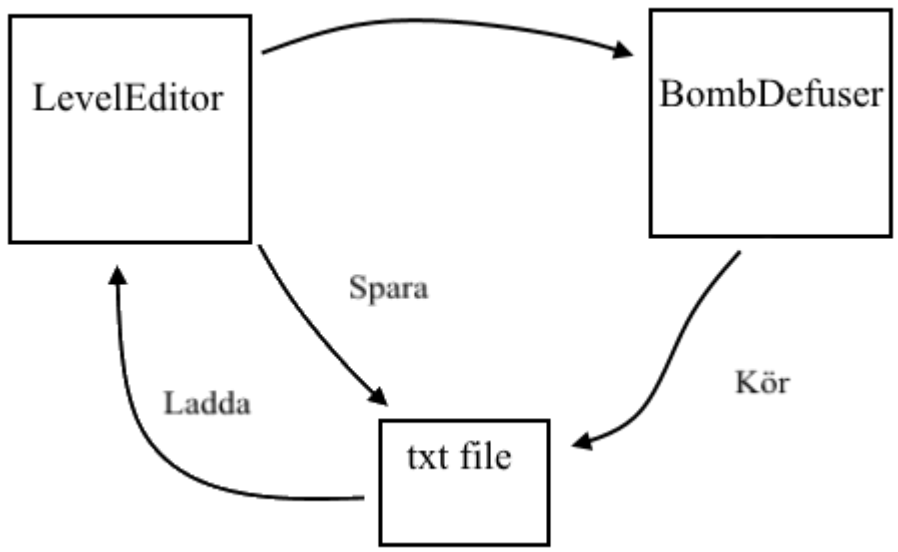
Kontext-diagrammet ovan visar med användaren som utgångspunkt hur man interagerar med leveleditorn. Blå rutan representerar den fysiska användaren, gröna bubblorna och rutorna

representerar dem olika stegen du tar genom levelleditorn och röda bubblan representerar avslut. Den fysiska användaren kan starta levelleditorn med ett knapptryck och levelleditorn kommer försöka visa användaren en tom mapp, samt en verktygslåda. Programmet kommer att kontrollera att det är körbart, om detta är fallet kommer den begärda datan att visas.

När användaren väl startar programmet kan användaren välja två vägar. Antingen att skapa en ny bana eller ladda en gammal. Om användaren väljer att skapa en ny bana kommer programmet först kolla att banan är sparad innan den kan köras. Om användaren väljer att ladda banan behövs det inte att banan sparar utan programmet kan köra banan direkt.

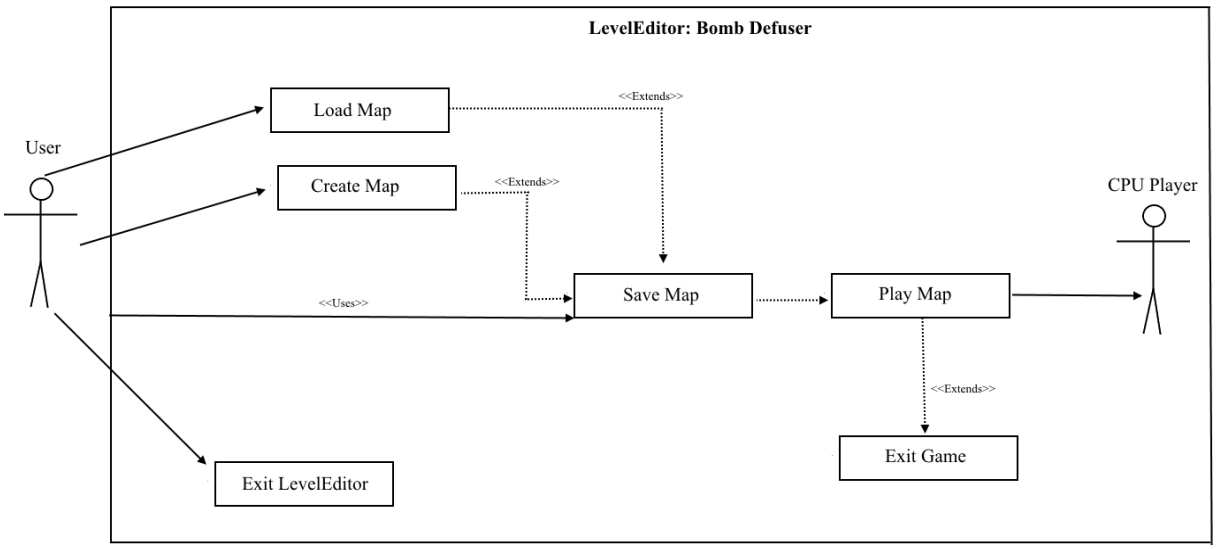
När användaren kört banan kan användaren enbart välja att stänga av systemet.

5 System-diagram



Detta är ett övergripande diagram hur levelleditorn kommunicerar mellan de olika system. Programmet skapar och sparar en text fil och sedan kan man köra den bana man precis skapat. Man startar spelet från levelleditorn och spelet går då in i ett annat läge som göra att spelet bara kommer köra den mappen man precis laddat och sparat i levelleditorn. Det som händer är att BombDefuser bara kör textfilen och när detta sker behöver spelet inte ladda menyer eller likande.

6 Use Cases

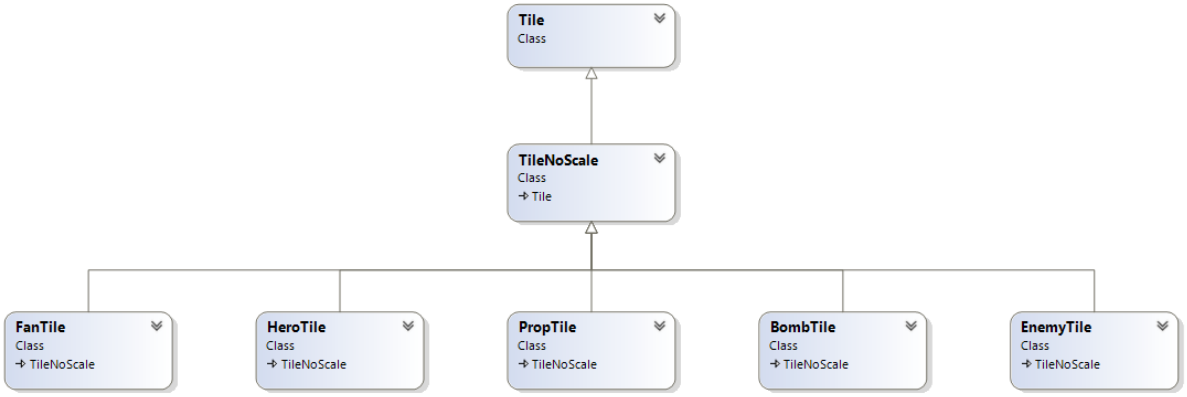


Användaren som i detta use-case diagrammet är vår huvudaktör som har som mål att skapa, spara och köra den banan man skapat. För att detta ska kunna gå att genomföra är det en förutsättning att användaren har en Windows-dator.

Den huvudsakliga uppgiften vår huvudaktör gör för att nå sitt mål är att trycka på knappar på skärmen. Vid uppstart av programmet måste aktören trycka på knappen ‘Setup’ som i sin tur gör att man måste välja BombDefuser.jar filen, för annars kommer användaren inte kunna köra sin level. Därefter kan aktören välja mellan att trycka på knapparna ‘New Map’ eller ‘Load’, antingen om man vill skapa ny bana eller ladda en gammal.

När väl användaren har skapar eller redigerat en bana kan aktören att trycka på knappen ‘Play’ som i sin tur initierar och laddar den bana användaren skapat. När väl användaren testat sin bana kan man välja mellan att trycka på knappen ‘Save’ eller forsätta redigera. Om användaren trycker på ‘Save’, kommer systemet spara ner banan på din hårddisk.

7 Klasshierarki



Eftersom vi vet om att allt ska vara uppbyggt av olika sorters tiles, känner vi att ha en basklass Tile som sedan andra klasser ärver av borde vara en bra uppbyggnad av systemet. Bassklassen Tile kommer vara objekt som går att går att dra i hur man vill. Därefter ser vi att TileNoScale ärver av Tile som sedan ärver många andra klasser av TileNoScale. Detta förekommer för att de andra objekten kan också manipuleras men det är under mer bestämda former. Det är alltså enbart ett fåtal saker som ska ändras i objekten såsom bredden, färg och likande. Vissa objekt får bara förekomma en gång (t ex karaktären och bomben) alltså kommer dessa underklasser ha vissa särskilda funktioner.

7.1 Klassdiagram

Klassdigrammet visar alla klasser som finns i levelleditorn. Dock visar klasshierarkin det viktigaste i leveeditorn. Utöver det finns det en Selektor-klass som hanterar de olika objekten. Samt olika “enums” som hanterar information som är nödvändigt för levelleditorn. Det finns även “forms” som inte beskriver kod utan är mer en design utav de egenskapsrutor som finns i levelleditorn.