

Krav- och Analysdokument för



Författare: Jenny Carlsson, Daniel Persson, Joel Hilmersson & Elin Forsberg

Datum: 2021-10-23

Version 2.0

---

# 1 Introduktion

Projektets ändamål syftar på att skapa ett tidsenligt spel inom genren "Tower defense". Bakgrunden till spelet ligger i Corona pandemin 2020-2021 som har fått utgöra ett centralt tema i applikationen då spelets fiender utgörs av Coronavirus som försöker attackera Chalmers. Målet för spelaren är därför att försöka placera ut torn som utgörs av figurer som representerar olika sektioner från Chalmers för att kunna stoppa viruset innan det når IT:s sektionslokal Hubben. Anledningen till att detta tema valdes är eftersom att många studenter är frustrerade över att inte kunna vara på skolan. Vi ville då ge dem ett sätt att kunna känna att de kan försvara Chalmers mot Coronan samtidigt som de har roligt när de spelar.

## 1.1 Definitioner, akronymer, och förkortningar

- JUnit
  - Inbyggda ramverket för enhetstestning av programmeringsspråket Java.
- (Desktopapplikation)
  - En applikation som kan köras på en dator
- (Kompilera)
  - Ett datorprogram som är en kompilator översätter programtexten(koden) som programmeraren har skrivit till ett motsvarande lågnivåprogram. Man säger att den kompilerar programtexten.
- Javadoc
  - Verktyg för att generera dokumentation i HTML-format utifrån kommentarer i källkod.
- Huvudbransch
  - Den officiella "grenen" för projektets källkod i github där själva programnets färdigskrivna kod finns.
- Power-ups
  - Funktion i spelet som ger spelarens torn en förhöjd förmåga av något slag, exempelvis extra stark i sina attacker.
- Spawn
  - Ett uttryck för att skapa eller generera något.

- Targeting
  - Uttryck för målinriktning som i detta fallet syftar på ett torns inställning att prioritera målen de skall attackera.
- UML-diagram
  - “Unified Modeling Language”- diagram är ett diagram som representerar strukturen av ett program på ett objektorienterat språk för modellering.
- IT-sektionen
  - Syftar på Chalmers universitets sektion för informationsteknik (IT-programmet).
- GUI
  - GUI (Graphical User Interface) är en typ av användargränssnitt där användaren kan interagera med programmet med hjälp av visuella indikatorer.
- Tower defense-spel
  - Ett strategispel som går ut på att försvara en slutdestination mot fiender genom att bygga torn av olika slag. Spelaren kan köpa nya torn med hjälp av pengar och om en fiende tar sig igenom banan förlorar spelaren liv.

## 1.2 Generella egenskaper i programmet

- Spelet är en desktopapplikation skrivet i programmeringsspråket Java.
- Spelet är skriven med spelbiblioteket libGDX.
- Användaren kan spela spelet för att klara uppdraget eller för att slå sitt personliga rekord.
- Användaren kan placera torn.
- Användaren kan aktivera ”power-ups”.
- Användaren kan se flödet av virus.
- Användaren kan starta och pausa en runda.
- Användaren kan ändra inställningar.

## 1.3 Programmets omfattning

- Chalmers defense använder inga externa program eller databaser, utan all funktionalitet och data hanteras av programmet självt.

## 1.4 Mål och framgångskriterier för projektet

1. Användaren skall kunna öppna applikationen och börja spela spelet.
2. Användaren skall kunna köpa och placera torn.
3. Användaren skall kunna förlora spelet.

4. Användaren skall kunna vinna spelet.

# 2 Krav

## 2.1 Användarberättelser

Användarberättelser som projektet har baserats på.

Hög prioritet	Medel prioritet	Låg prioritet
CHD05 : Tjäna valuta	CHD04 : Uppgradera torn	CHD09 : Slå rekordet
CHD07 : Förlora spelet	CHD03 : Ta bort torn	CHD13 : Läsa information om spelet
CHD16 : Klara runder	CHD06 : Vinna spelet	CHD18 : Ändra targeting mode
CHD01 : Attackera virus	CHD10 : Använda Power ups	CHD19 : Snabbspola runda
CHD08 : Förlora liv	CHD11 : Pausa spelet	CHD20 : Ändra svårighetsgrad
CHD17 : Starta runda	CHD12 : Ändra inställningar	CHD21: Ljudeffekter
CHD02 : Placera torn		CHD22: Boss virus
CHD15 : Spawna virus		CHD23: Visa progressbar
CHD14 : Starta spelet		CHD24: Hot Keys

**Figur 2.1 :** Rutor ifyllda med grönt är implementerade användarberättelser i programmet.

### # Hög prioritet :

#### CHD05 : Tjäna valuta

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag kunna tjäna ihop någon form av valuta så att jag kan köpa saker i spelet.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag få pengar av att döda virus?
- Kan jag få pengar av att klara av en runda?
- Kan jag få pengar av ett torn?
- Kan jag se att jag får pengar?

#### **Icke-funktionella**

- Lägger spelet till pengar själv?
- Får jag pengarna från de olika inkomstkällorna direkt?
- Ökar inkomsten av pengar efter varje runda?
- Kan jag få mer pengar från ett torn som är uppgraderat gentemot ett som inte är det?

## **CHD07 : Förvara spelet**

### Implementerad?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna förlora spelet så att det finns en utmaning.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag förlora?
- Kan jag se att jag har förlorat?
- Kan jag starta om efter att jag har förlorat?

#### **Icke-funktionella**

- Förlorar jag mina framsteg när jag förlorar spelet?
- Förlorar jag genom att få slut på liv?
- Kan jag förlora när som helst inom spelet?
- Kan jag förlora på flera sätt?

## **CHD16 : Klara runder**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag kunna klara runder så jag tillslut kan vinna spelet

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag klara en runda?
- Kan jag se att jag har klarat en runda?

#### **Icke-funktionella**

- Måste jag ha dödat alla virus innan jag kan klara runden?
- Kan jag starta en ny runda innan jag har klarat den gamla?

## **CHD01: Attackera virus**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag att mina torn ska kunna attackera virusen för att komma vidare.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag skada virusen med mina torn?
- Kan jag se att virusen tar skada?
- Kan jag reducera virusets hp?

#### **Icke-funktionella**

- Kan jag skada samma virus flera gånger?

## **CHD08 : Förlora liv**

Implementerad?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna förlora liv för att kunna förlora spelet.

### **Acceptanskriterier**

#### **funktionella**

- Förlorar jag liv när virus tar sig till slutet av banan?
- Kan jag se när jag förlorar liv?
- Kan jag förlora liv?
- Kan jag se hur mycket liv jag har kvar?

#### **Icke-funktionella**

- Kan jag förlora liv när som helst?
- Kan jag förlora liv på olika sätt?
- Förlorar jag olika mycket liv?

## **CHD17 : Starta runda**

Implementerad?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna starta rundor så jag kan komma vidare i spelet

### **Acceptanskriterier**

#### **funktionella**

- Kan jag starta en runda?
- Ser jag att rundan är startad?
- Måste jag starta en runda inom en viss tidsram?

#### **Icke-funktionella**

- Kan jag starta flera runder samtidigt?
- Måste jag starta rundan själv?

## **CHD02 : Placera torn**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag kunna placera ut torn så att de kan attackera virus

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag placera ett torn?
- Kan jag se vilka torn som är placerade?
- Kan jag se vart jag kommer placera mitt torn?
- Kan jag se vart jag kan placera mitt torn?

#### **Icke-funktionella**

- Kan jag placera ut ett torn vart som helst?
- Kan jag placera ett torn när som helst?
- Kan jag placera ut ett torn utan tillräckligt med valuta?
- Kan jag placera hur många torn som helst?

## **CHD15 : Spawna virus**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag att spelet ska spawna virus så att jag får ett motstånd.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag se att spelet spawner virus?
- Kan jag påverka när spelet spawner virus?

#### **Icke-funktionella**

- Kan spelet spawna virus när som helst?

- Spawnar spelet olika virus varje runda?
- Följer spelet samma mönster när det spawner virus? (Samma intervall hela tiden eller olika)
- Spawner spelet starkare virus i senare rundor?

## CHD14 : Starta spelet

Implementerad?

- Ja

### Beskrivning

- Som en användare vill jag kunna börja spela spelet så att jag kan ha roligt en stund.

### Acceptanskriterier

#### Funktionella

- Kan jag börja spela spelet?
- Kan jag se att spelet börjar?

#### Icke-funktionella

- Rensar spelet den tidigare spelplanen jag hade?
- Kan jag starta spelet när som helst?

## # Medel prioritet :

## CHD04 : Uppgradera torn

Implementerad?

- Ja

### Beskrivning

- Om en användare vill jag kunna uppgradera mina torn så att de blir bättre

### Acceptanskriterier

#### Funktionella

- Kan jag uppgradera mitt torn?
- Kan jag se grafiskt att tornet är uppgraderat?

- Ser jag att tornet gör mer skada/ har blivit bättre?
- Förlorar jag valuta när jag uppgraderar ett torn?
- Kan jag se vilket torn jag håller på att uppgradera?
- Kan jag se hur många gånger jag kan uppgradera mitt torn?

#### **Icke-funktionella**

- Kan jag uppgradera ett torn utan tillräckligt med valuta?
- Hur många uppgraderingar kan jag göra?
- Kan jag köpa samma uppgradering flera gånger?
- Kan jag uppgradera när som helst?

### **CHD03 : Ta bort torn**

Implementerad?

- Ja

#### **Beskrivning**

- Som en spelare vill jag kunna ta bort torn för att få pengar till annat.

#### **Acceptanskriterier**

##### **Funktionella**

- Kan jag ta bort ett torn?
- Får jag tillbaka pengar när jag tar bort ett torn?
- Kan jag se att ett torn är borttaget?
- Kan jag se vilket torn jag håller på att ta bort?

##### **Icke-funktionella**

- Kan jag ta bort vilket torn som helst?
- Kan jag ta bort ett torn när som helst?
- Får jag tillbaka olika mycket pengar ifall jag har uppgraderat tornet olika?

### **CHD06 : Vinna spelet**

Implementerad?

- Ja

#### **Beskrivning**

- Som en användare vill jag kunna vinna spelet så att jag kan skryta om

det för alla

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag vinna spelet?
- Kan jag se att jag har vunnit?
- Vet jag efter vilken runda jag kommer vinna?
- Kan man fortsätta efter man har vunnit?

#### **Icke-funktionella**

- Kan jag vinna spelet utan att ha dödat alla virus?
- Hur länge kan jag fortsätta efter att jag har vunnit?

## **CHD010 : Använda Power ups**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag kunna använda power-ups för att jag lättare ska kunna slå mitt rekord

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag använda power-ups?
- Kan jag se vilken powerup som är aktiverad?
- Kan jag se hur länge till powerupen varar?
- Kan jag se när jag kan använda den igen?
- Kan jag se att mina torn blir starkare när jag använder powerups?

#### **Icke-funktionella**

- Håller spelet reda på hur länge powerupen har varit?
- Kan jag använda powerups när jag inte har råd?
- Kan jag använda flera powerups samtidigt?

## **CHD11 : Pausa spelet**

Implementerad?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna pausa spelet så att jag kan ta en paus.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag pausa spelet?
- Kan jag se att spelet är pausat?
- Får jag upp en paus meny när jag pausar?

#### **Icke-funktionella**

- Finns det en tangentbordsknapp jag kan klicka på för att pausa
- Kan jag pausa genom att klicka någon knapp på skärmen?
- Slutar spelet att "spela" i bakgrunden?

## **CHD12 : Ändra inställningar**

Implementerad?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna ändra inställningarna så att jag kan anpassa spelet till mina behov

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag ändra inställningar?
- Kan jag se att inställningarna är ändrade?
- Kan jag återställa inställningarna?

#### **Icke-funktionella**

- Behöver jag spara inställningarna?
- Sparas mina inställningar mellan spel?
- Finns det "default" inställningar?

# **Låg prioritet :**

## **CHD09 : Slå rekordet**

Implementerad?

-

### **Beskrivning**

- Som en spelare vill jag kunna slå mitt rekord så att jag vet när jag blivit bättre på spelet

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag se när jag slog rekordet?
- Kan jag se mitt tidigare rekord?
- Kan jag se när mitt rekord uppdateras?

#### **Icke-funktionella**

- Sparar spelet mitt rekord?
- Vet spelet när jag som spelare slår rekord?
- Uppdateras det visade rekordet i samband med att man ökar det?

## **CHD13 : Läsa information om spelet**

Implementerad?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna läsa information om spelet för att kunna planera strategier bättre.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag läsa information om spelet?
- Kan jag se var jag kan läsa information om spelet?

#### **Icke-funktionella**

- Kan jag läsa informationen på flera ställen?
- Kan jag läsa informationen när som helst?

## **CHD18 : Ändra targeting mode**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag kunna styra över hur mina torn väljer att skjuta så att jag kan utveckla en smart strategi.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag ändra targeting mode?
- Ser jag att tornet skjuter utefter mina val?

#### **Icke-funktionella**

- Måste jag ändra inställningen på varje torn för sig?
- Sparas inställningen för det tornet jag har valt automatiskt?
- Kan jag ändra targeting mode när som helst?

## **CHD19 : Snabbspola runda**

Implementerad?

- Ja

### **Beskrivning**

- Som en spelare vill jag kunna snabbspola runder så jag kan vinna spelet snabbare om jag vill.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag snabbspola en runda?
- Kan jag se att jag snabbspolar en runda?
- Kan jag ta bort snabbspolningen under rundan?

#### **Icke-funktionella**

- Behålls snabbspolningen när nästa runda börjar??
- Kan jag sätta på snabbspolningen när som helst?
- Kan jag stänga av snabbspolningen när som helst?

## **CHD20 : Ändra svårighetsgrad**

Implementerad?

-

### **Beskrivning**

- Som en spelare vill jag kunna ändra svårighetsgrad så jag kan spela spelet så som jag vill.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag ändra svårighetsgrad?
- Märker man att svårigheten faktiskt skiljer svårighetsgraderna åt?
- Kan jag se att jag har valt ett svårare läge?

#### **Icke-funktionella**

- Blir viruset svårare att döda med en svårare svårighetsgrad?
- Blir det fler virus per runda med en svårare svårighetsgrad?
- Väljer jag svårighetsgrad innan jag börjar spelet?
- Ändrar svårighetsgraden antalet runder jag måste klara för att vinna?

## **CHD21 : Ljudeffekter**

Implementera?

-

### **Beskrivning**

- Som en spelare vill jag att spelet har ljud för att jag som spelare skall få en bättre spelupplevelse.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag höra ljud?

#### **Icke-funktionella**

- Kan jag höra flera ljud?
- Kan jag bestämma när jag hör ljudet?

- Hör jag ljud hela tiden?

## **CHD22 : Boss virus**

Implementera?

- Ja

### **Beskrivning**

- Som en användare skulle jag vilja att spelet har boss fiender som är svårare att slå så att jag får mer spänning och variation när jag spelar.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag se att något virus är svårare?
- Märker jag av att viruset är svårare?

#### **Icke-funktionella**

- Har boss viruset många liv?
- Spawnar flera virus efter mina torn dödat bossviruset?
- Hur ofta kommer bossvirus?

## **CHD23 : Visa progressbar**

Implementera?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna se hur långt jag har kommit i spelet så jag blir stolt över vad jag åstadkommit och blir motiverad att fortsätta.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag se mina framsteg i spelet?

#### **Icke-funktionella**

- Kan jag se hur långt jag kommit?
- Kan jag se hur långt det är kvar?
- Kan jag se min framsteg hela tiden?

## **CHD24 : Hot keys**

Implementera?

- Ja

### **Beskrivning**

- Som en användare vill jag kunna använda hot keys för att jag skall kunna vara mer effektiv när jag vant mig vid spelet.

### **Acceptanskriterier**

#### **Funktionella**

- Kan jag använda tangentbordet för att utföra uppgifter?

#### **Icke-funktionella**

- Kan jag använda tangentbordet till flera uppgifter?
- Kan jag använda tangentbordet när som helst?
- Vet jag när jag kan använda tangentbordet?

## **2.2 Definition of Done**

- Koden har haft en kvalitetsgranskning.
- Alla JUnit tester passerar och får godkänt
- Applikationen går att bygga och kompilera
- Koden är dokumenterad med Java-doc
- Koden har sammanfogats till huvud "branchen" i github
- UML-diagrammet är uppdaterat och speglar kodens struktur
- Alla högprioriterade användarberättelser har blivit implementerade
- Koden följer en enhetlig standard

## 2.3 Användargränssnitt

Denna sektion kommer skapa en visuell bild av applikationen med hjälp av bilder och beskrivande text.

### 2.3.1 Startsidan

Användaren möts av startmenyn vid öppning programmet. Startmenyn består av olika knappalternativ varav den mittersta knappen tar användaren till spelvyn. Startmenyns GUI är designat i färger inspirerade av Chalmers och IT-sektionen.



Figur 2.2 : Startmenyn till Chalmers Defense.

#### 2.3.1.1 Spelinformation

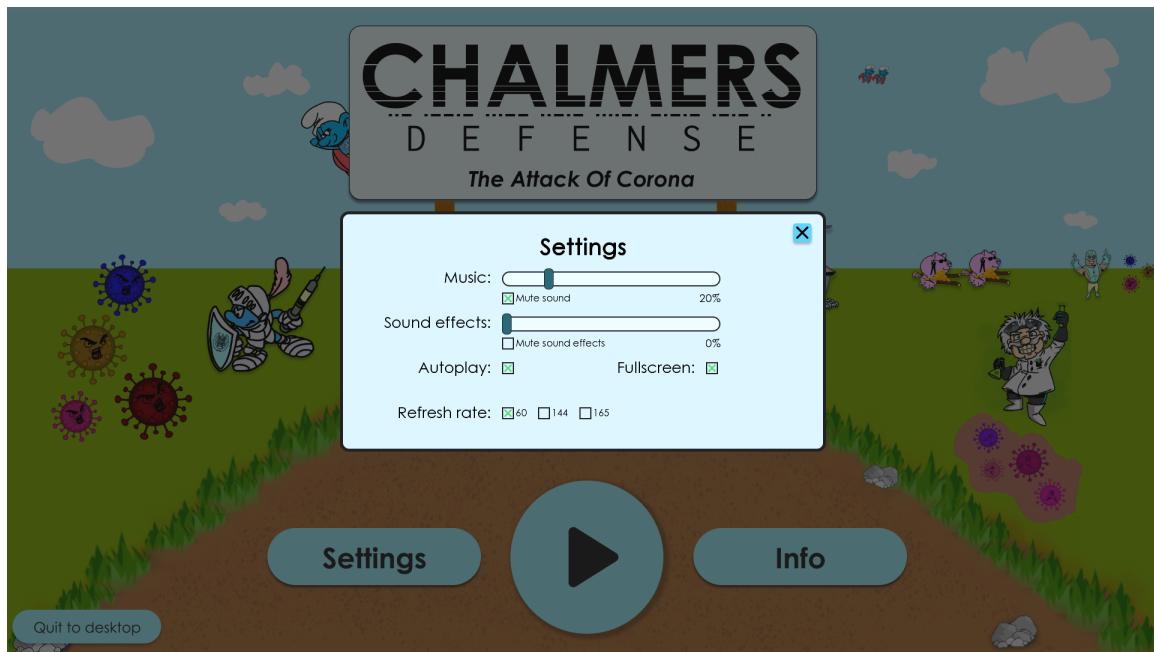
På vyn för spelinformation (knapp-benämning: "info") kan användaren läsa lite om spelet och få reda på speltips, vilka olika kortkommandon som finns i programmet.



**Figur 2.3 :** Vy för information om spelet.

### 2.3.1.2 Inställningar

Via knappen "settings" kommer användaren till kontroller för inställningar. Här får hen möjlighet att modifiera ett urval inställningar, exempelvis ljud och skärmstorlek.



**Figur 2.4 :** Vy för information om spelet.

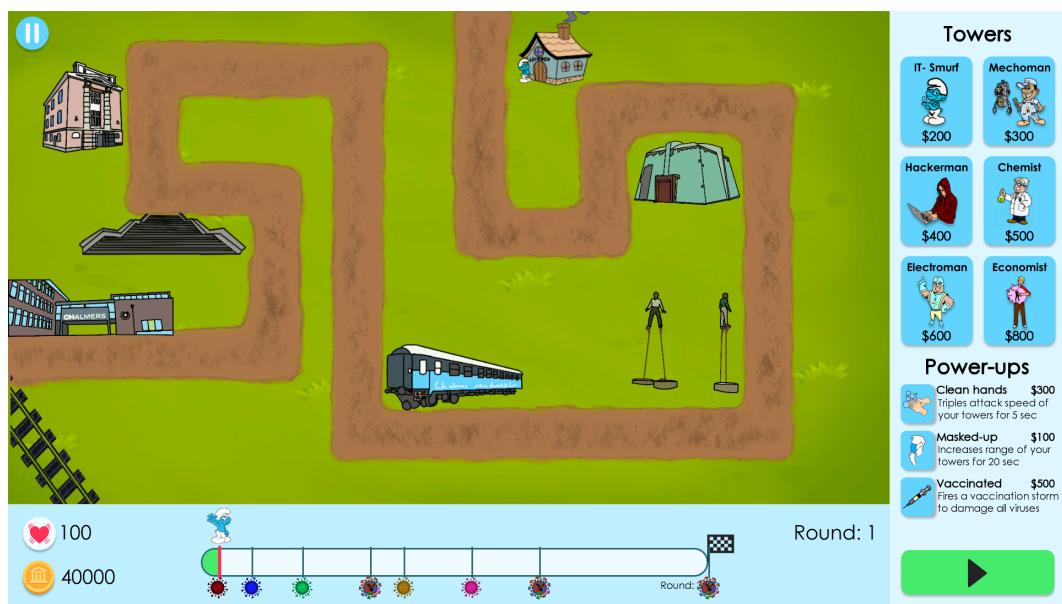
## 2.3.2 Spelvyn

När användaren tryckt på play-knappen på startmenyn dyker spelvyn upp. Denna vy består av spelplanen, panel med torn och power-ups, spelkontroller och en dynamisk informationspanel, som i grundläget innehåller en framstegsmätare, liv och pengar. Här kommer användaren befina sig under tiden spelet spelas.

Väl inne i spelvyn har spelaren möjligheten att starta spelet direkt genom att trycka på startknappen nere i det högra hörnet eller klicka på mellanslag på tangentbordet. Samma knapp fungerar sedan som en uppsnabbnings-knapp under rundorna.

Ett annat alternativ är att istället börja sätta ut torn från panelen på högra sidan av skärmen innan spelaren startar. Detta kan göras under hela spelets gång, inklusive mellan runder. Spelaren kan även använda power ups, vilka ligger ovanför startknappen, för att tillfälligt stärka tornen.

När väl spelet har startat kommer fienden börja färdas från vänster sida av spelplanen, genom slingan och försöka komma ut ur andra änden. Användarens uppgift är att eliminera fienden innan det händer.



**Figur 2.5 :** Spelvyn för Chalmers Defense.

### 2.3.2.1 Uppgradera torn

Efter att användaren köpt och placerat ett torn har hen även möjligheten att

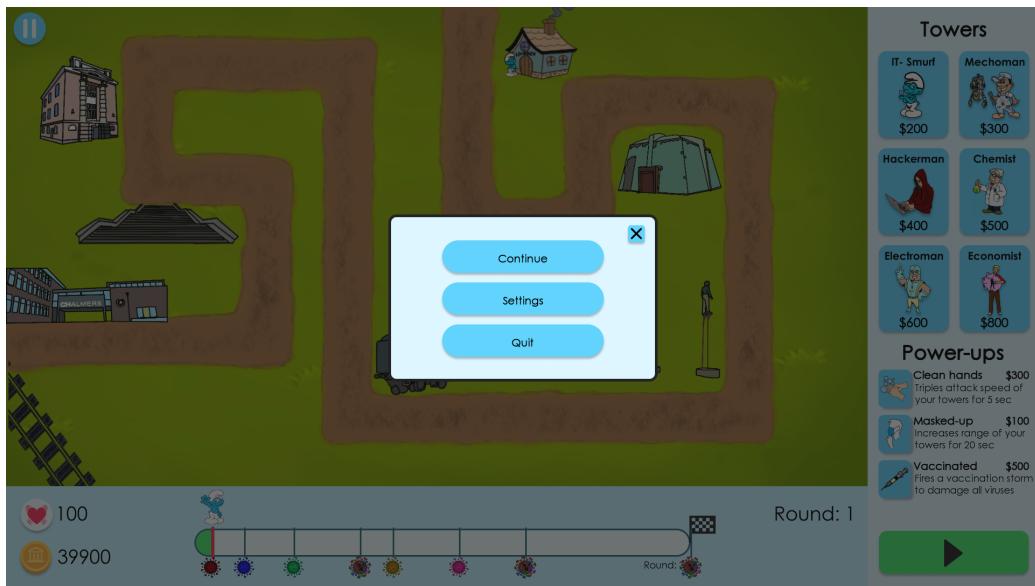
uppdatera tornet. Detta görs genom att klicka på ett placerat torn på kartan och då dyker denna vy upp med information om uppdateringen.



Figur 2.6 : Spelvyn när användaren klickat på ett placerat torn.

### 2.3.2.2 Pausmeny

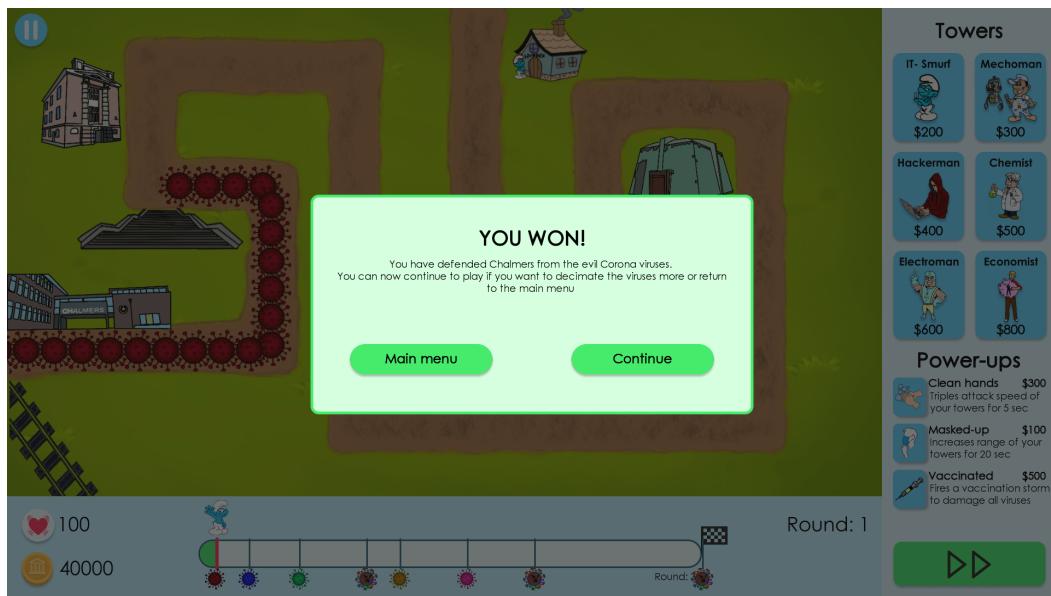
Pausmenyn nås genom att klicka på paus-knappen uppe i övre vänstra hörn i spelvyn, alternativt använda kortkommando genom att trycka på tangenten "escape". Här får användaren möjlighet att ändra inställningar, avsluta spelet eller fortsätta.



**Figur 2.7** : Pausmenyn.

### 2.3.2.3 Vinstvy

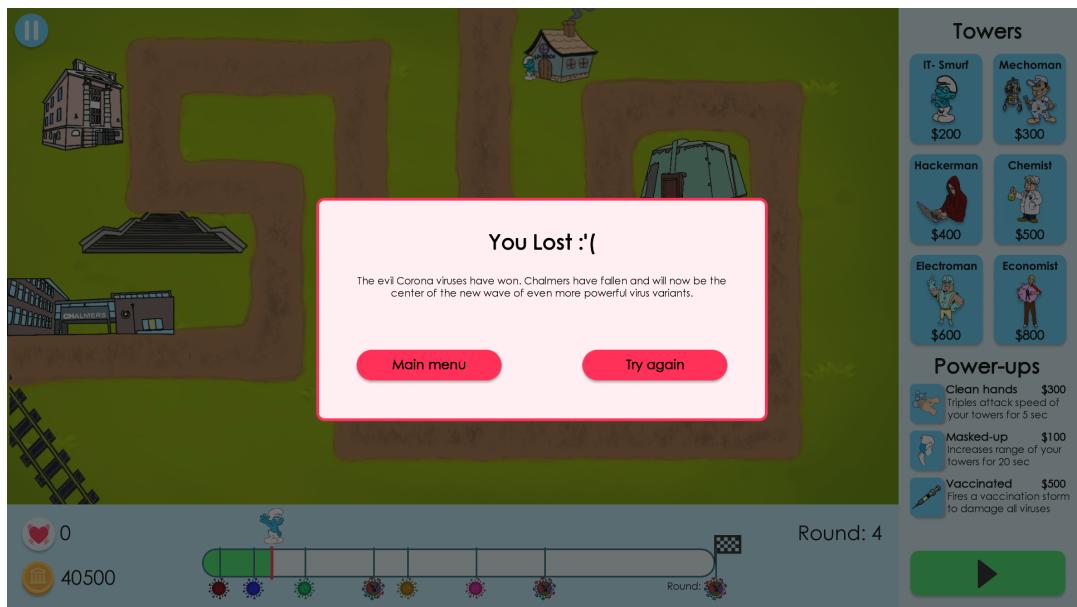
En runda är slut när det inte finns några fler virus på banan. Nästa runda kan därefter startas genom att återigen trycka på startknappen. Detta upprepas genom ett flertal runder tills spelaren har överlevt till sista runden där runden avslutas med en vinnarvy där användaren kan välja att fortsätta spela eller avsluta. Denna vy dyker upp när spelaren vunnit spelet.



**Figur 2.8** : Ruta för när man vunnit spelet.

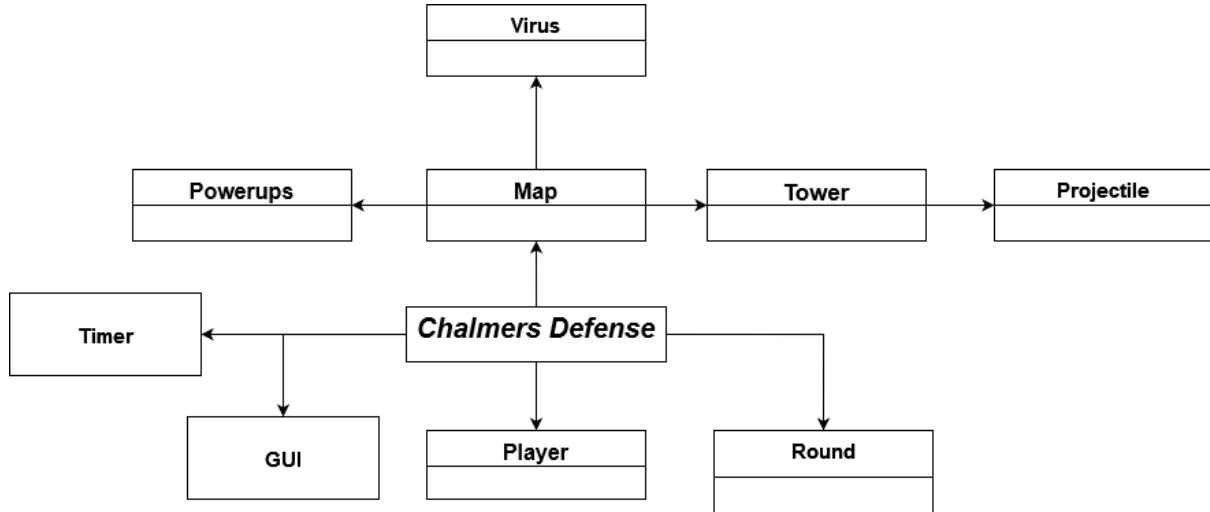
### 2.3.2.4 Förlustvy

Om spelaren skulle förlora alla sina liv vid någon tidpunkt så kommer det istället upp en förlorarvy där hen kan antingen starta om eller avsluta spelet.



**Figur 2.9 :** Ruta för när man förlorat spelet.

# 3 Domänmodell



**Figur 3.1** : Domänmodellen över programmet.

## 3.1 Ansvarsområden i designmodellen

Förklaringar av de olika huvuddelarna och deras ansvarsområden inom domänmodellen.

### Map

Map's ansvarsområde är spelplanens objekt. Detta innefattar torn, projektiller och virus främst då det är objekt som kommer att vara placerade på kartan och ha en position. Utöver detta har Map även hand om powerups, eftersom powerups bli applicerade på tornen och virusen.

### Tower

Tower har som ansvar att ha hand om tornrelaterade saker, det vill säga funktionen att skjuta projektiller och att hålla reda på vilket målläge den ska använda för att sikta.

### Projectile

Projectile ansvarar för projektilverrelaterade saker. Exempelvis vilken vinkel projektileten har, hastighet, och hur mycket skada de ska göra när de träffar ett virus.

## **Virus**

*Virus* har ansvaret att hantera fienderelaterade saker för programmet. Detta innehåller hur de ska röra sig för att följa banan och hur mycket skada de gör till spelaren om de lyckas tas sig igenom försvaret.

## **Powerups**

*PowerUps's* ansvar är att hålla reda på om de är aktiva eller inte, samt nerkylingstiden innan man kan aktivera dem igen. De har även som ansvar att lägga till grafiska element till skärmen som visar att en powerup har blivit aktiv.

## **Timer**

*Timer* har ansvaret att uppdatera modellen/ChalmerDefense med en konstant klocka. Denna kan man sedan öka och minska i hastighet för att göra spelet snabbare eller långsammare under runder.

## **Rounds**

*Rounds* hanterar "rund"-relaterade saker, såsom vilken runda spelaren befinner sig på och vilken runda som man vinner på.

## **Player**

*Player* har som ansvarsområde att hålla reda på spelarens fornödigheter. Detta innehåller hur mycket liv spelaren har och andelen pengar spelaren har samlat på sig.

## **GUI**

*GUI* i domändiagrammet står för hela View modulen som programmet använder sig av. Detta paket står för själva renderingen av applikationen och är det som visas till användaren.

# 4 Referenser

## 4.1 Verktyg

- Github
  - Webbaserad och centraliserad lagring av versionshistorik för programvaruutvecklingsprojekt.
- Slack
  - En kommunikativ plattform för att diskutera intern information gällande projektet.
- Figma
  - Webbaserad plattform för att designa gränssnitt.
- Zoom
  - Verktyg för digitala möten.
- Google Drive
  - Textredigeringsprogram, som tillåter delning och parallellt arbete med live-uppdateringar.
- Trello
  - Projektplaneringsverktyg
- Diagrams.net
  - Webbsida för att skapa och redigera UML-diagram
- JUnit
  - Java bibliotek samt verktyg för att testa skriven kod.
- Travis CI
  - Verktyg för att automatisera byggandet och testandet av JUnit tester.
- Javadoc
  - Ett system som autogenererar kommentarer från koden på ett standardiserat sätt till en sammanställning.

## 4.2 Bibliotek

- JUnit
  - Är ett ramverk som programmerare i Java använder för enhetstestning.
- libGDX
  - Är ett gratis plattformsoberoende Java-spelutvecklingsramverk. Det är ett javabibliotek som ger stöd för bland annat grafik, inmatningar och ljud.

Systemdesign dokument för



Författare: Jenny Carlsson, Daniel Persson, Joel Hilmersson & Elin Forsberg

Datum: 2021-10-23

Version 2.0

---

# 1 Introduktion

Målet med denna rapport är att beskriva strukturen av mjukvarusystemet bakom spelet *Chalmers Defense*. Detta kommer att göras genom UML-diagram och beskrivande text för att ge en god bild över programmet.

*Chalmers Defense* är en desktopapplikation i form av ett spel. Spelet är en typ av "tower defense" spel men med ett Chalmers-tema där målet är att eliminera coronavirus av olika varianter innan de tar sig till IT-sektionens sektionslokal Hubben. Om man lyckas försvara Chalmers i 30 runder så vinner man och om man inte gör det förlorar man.

Länk till projektets GitHub finns bland bilagor.

## 1.1 Designmål

Målet med designen är att vi skall ha en applikation med hög sammanhållning och få sammanbindningar. Vi vill att applikationen skall vara testbar och ha goda möjligheter för expansioner. Till sist vill vi ha en design som återanvänder kod och har få beroenden.

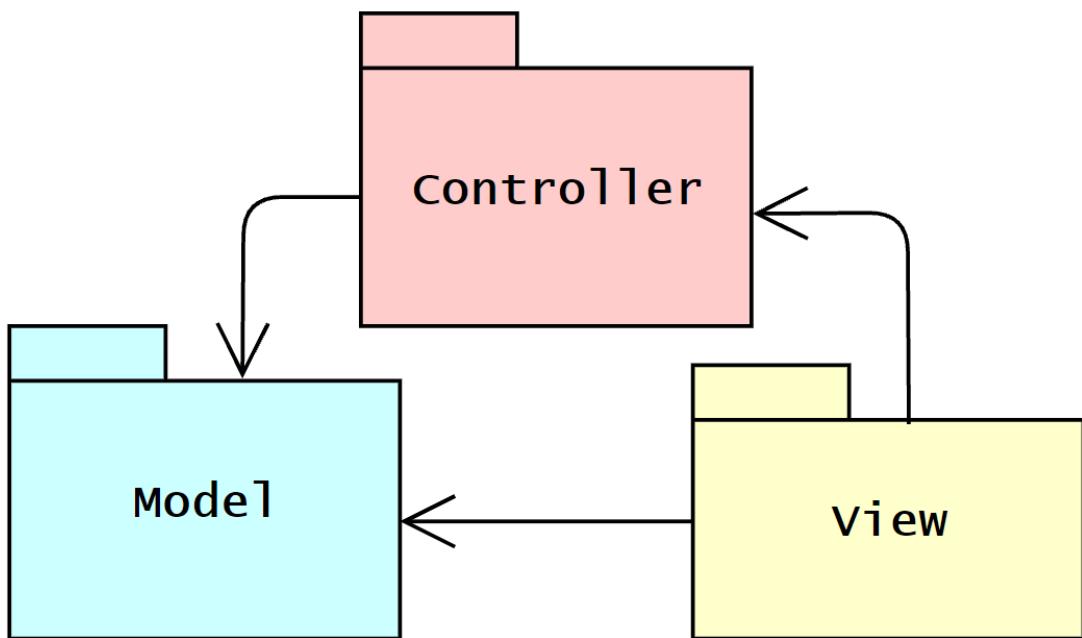
## 1.2 Definitioner, akronymer, och förkortningar

- JUnit
  - Inbyggda ramverket för enhetstestning av programmeringsspråket Java.
- Chalmers Defense
  - Namnet på vårt program.
- JSON
  - Javascript object notation.
- Javadoc
  - Verktyg för att generera dokumentation i HTML-format utifrån kommentarer i källkod.
- Huvudbransch
  - Den officiella "grenen" för projektets källkod där själva programnets färdigskrivna kod finns.
- Power-ups
  - Funktion i spelet som ger spelarens torn en förhöjd förmåga av något slag, exempelvis extra räckvidd till sina attacker.
- Spawn / Spawna

- Ett uttryck för att skapa eller generera något.
- Rendera
  - Används för att beskriva när programmet visar något på skärmen.
- Targeting
  - Uttryck för målinriktning som i detta fallet syftar på ett torns inställning att prioritera målen de skall attackera.
- UML-diagram
  - “Unified Modeling Language”- diagram är ett diagram som representerar strukturen av ett program på ett objektorienterat språk för modellering.
- Dependencies
  - Man säger att en modul har en “dependency” till en annan modul om den är beroende av den modulen på något sätt. Kan exempelvis vara så att en modul behöver använda en annan modul för att kunna göra sitt jobb.
- IT-sektionen
  - Syftar på IT-programmet på Chalmers tekniska högskola.
- SMART
  - All domän logik bör ligga i modellen, därför brukar den kallas för smart.
- DUM
  - En vy ska inte utföra egna beräkningar, bara rendera baserat på direktiven från andra. Därför kallas den ofta för dum.
- TUNN
  - En kontroller ska endast hantera extern inmatning, som oftast är inmatning från användare. Det bör vara ett tunt lager mellan användare och program, därför kallas kontroller för tunn.
- Runda
  - En runda i Chalmers Defense börjar med ett klick på startknappen. Då börjar virus spawna in och åka igenom banan. Det finns alltid ett fixerat antal virus per runda, så rundan är avklarad när alla har spawnat och antingen dött eller kommit till slutet.
- Separation of concern
  - Är en designprincip som innebär att ett modul ska endast ha en väldefinierad uppgift.
- High Cohesion, low coupling
  - Designprincip som säger att det skall vara hög sammanhållning mellan relaterade komponenter och få sammanbindningar till komponenter som inte är relaterade.
- GUI

- GUI (Graphical User Interface) är en typ av användargränssnitt där användaren kan interagera med programmet med hjälp av visuella indikatorer.
- HUD
  - I spel så är HUD (Heads-up display) en metod för att visa visuell information till användaren som en del av spelets GUI.
- Overlays
  - I Chalmers Defense används en overlay för att visa vinstvyn och förlorarvyn. En overlay innebär att man lägger något visuellt ovanpå den aktuella skärmen men täcker inte allt. I detta fall läggs vinn- och förlorarvyn som en ruta i mitten av skärmen som användaren kan interagera med.
- libGDX
  - Vårt primära bibliotek för att rendera spelet på skärmen.
- LabelStyle
  - En libGDX class som ger varje textelement en specifik stil beroende på font storlek, font stil och färg.
- Sprite
  - En libGDX klass som håller information över position, rotation samt storlek över en viss bild (kallad "Sprite") som kan renderas till fönstret.
- Custom exception
  - En egenskapad undantags klass för användning inom ett visst syfte när javas standard undantag inte passar.
- Tower defense-spel
  - Ett strategispel som går ut på att försvara en slutdestination mot fiender genom att bygga torn av olika slag. Spelaren kan köpa nya torn med hjälp av pengar och om en fiende tar sig igenom banan förlorar spelaren liv.

## 2 Systemarkitekturen



Figur 2.1: Arkitekturen av programmet

Programmet är uppdelat i fyra delar enligt MVC:

- Model: är SMART och står för logiken för programmet & klasserna. Denna är oberoende av de andra.
- View: är DUM och inkluderar det som presenteras utåt, det vill säga gränssnitt etc. Den hämtar endast listor från Model och agerar på dessa.
- Controller: är TUNN och hanterar användarinput som sedan direkt skickas till Model för vidare behandling inom programmet.

### 2.1 Programflödet

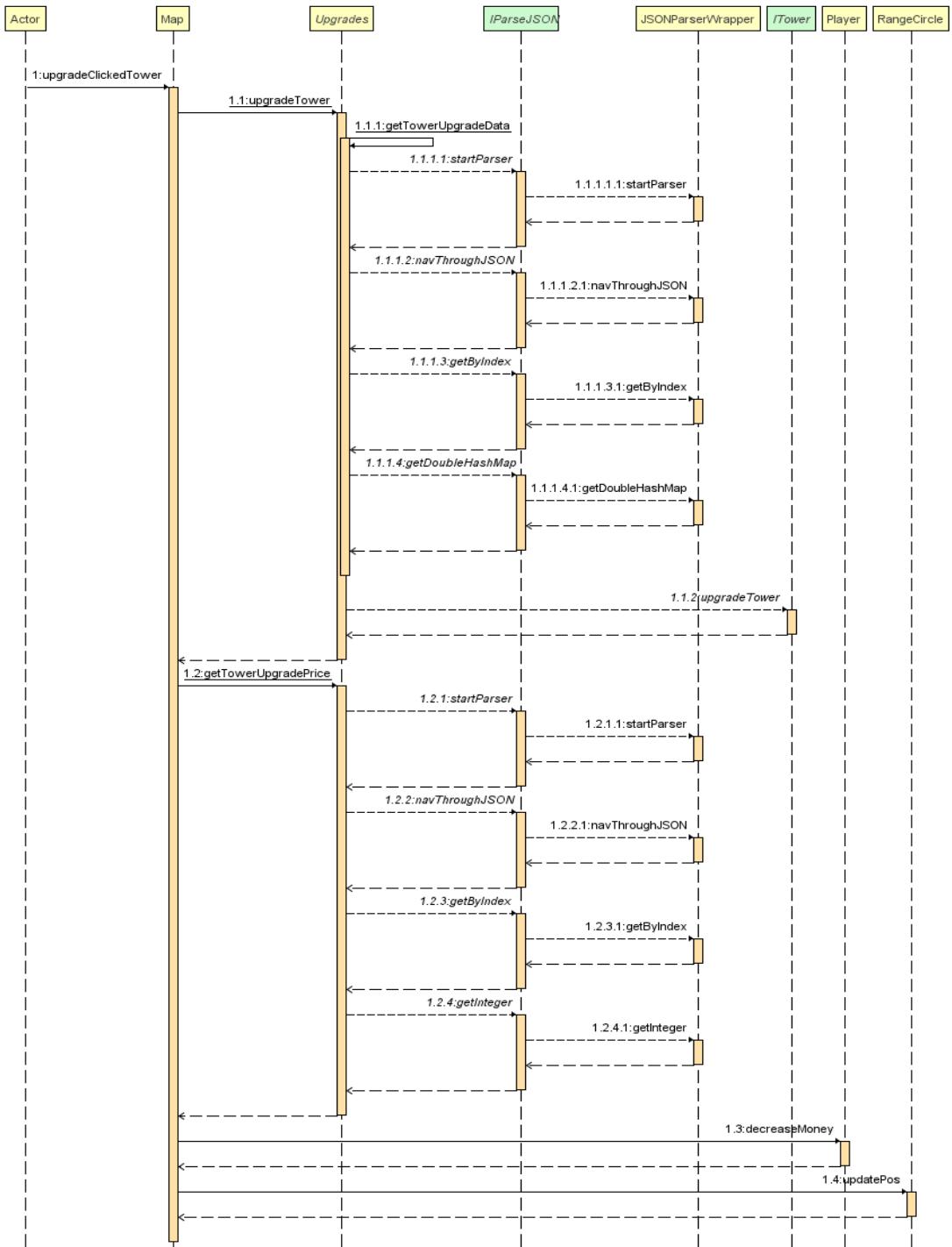
Programmet startas genom klassen *ChalmersDefense*. Den initialiseras allt som behövs för att få upp den första vyn som spelaren ser, vilket är huvudvyn, samt de andra vyerna som man i spelet kan komma åt. De individuella klasserna bygger i sin tur upp sina komponenter, såsom knappar och bilder som uppbygger vyn. Klassen skapar även de olika kontroll-klasserna som utgör Controller modulen, samt Model klassen som vidare initierar alla objekt som tillhör Model modulen. Från huvudvyn som spelaren initialt befinner sig på kan spelaren klicka på en startknapp, vartefter programmet går vidare genom att lägga upp spelvyn så att man kan börja att spela.

Programmet väntar nu på signaler från kontroller klasserna. När spelaren interagerar med en av knapparna i GUI:t så kommer kontroller klasserna att reagera genom att delegera till *Model* som tar in vad som har blivit klickat på och agerar därefter. Antingen hanteras det själv av *Model* eller mer troligen delegeras det vidare till andra klasser inom modulen *Model*.

En runda startas genom att användaren klickar på startknappen. När detta händer skickar Controller startknappaanropet till *model* för vidare hantering. *Model* startar upp uppdateringsklockan i klassen *GameTimer* som kontinuerligt kallar på uppdateringsmetoden i *Model* så att alla subkomponenter kan uppdateras. Direkt efter klockuppstarten anropas också *SpawnViruses* klassen med den aktuella rundan från *Player* till att börja spawna en förprogrammerad sekvens av virus. Dessa virus börjar sedan att röra sig utefter banan som finns i spelet.

En runda avslutas när det inte finns några virus kvar på banan. När rundan är avslutad säger *Model* till *GameTimer* att sluta kalla på metoden "updateModel" vilket resulterar i att allt pausas. I samma skede kallas ytterligare metoder för att göra spelet redo till starten av nästa runda.

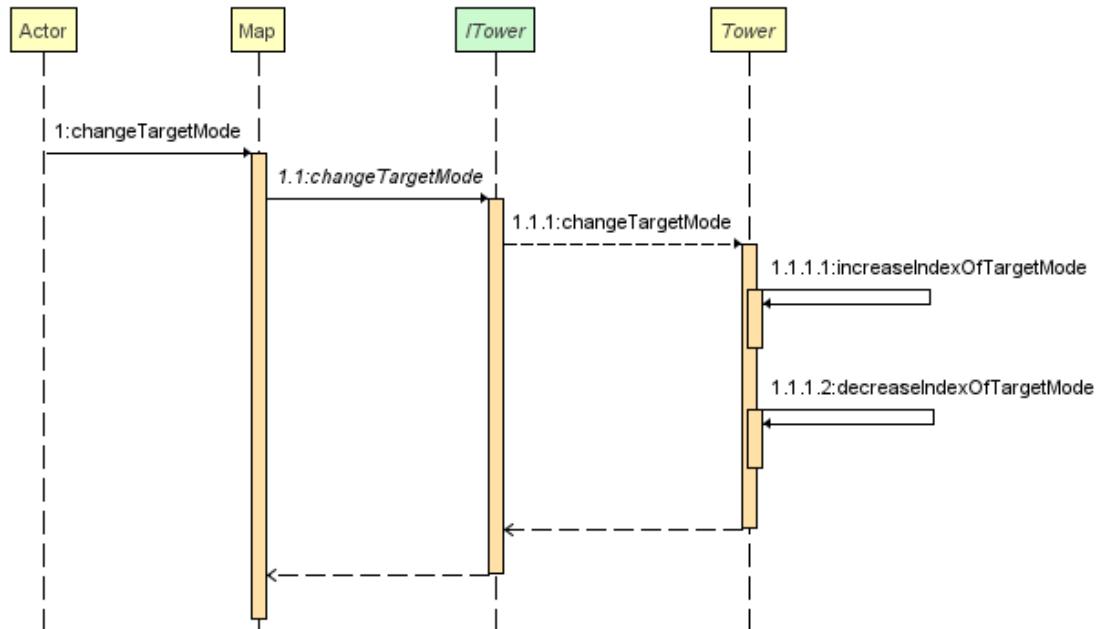
Spelaren kan både under runder och mellan runder när som helst välja att dra ut ett nytt torn från högermenyn och placera dessa på spelplanen. Tornen kommer sedan beroende på om rundan är aktiv eller ej att direkt börja fungera som tänkt utan några problem.



Figur 2.2: Ett sekvensdiagram av metoden "upgradeClickedTower"

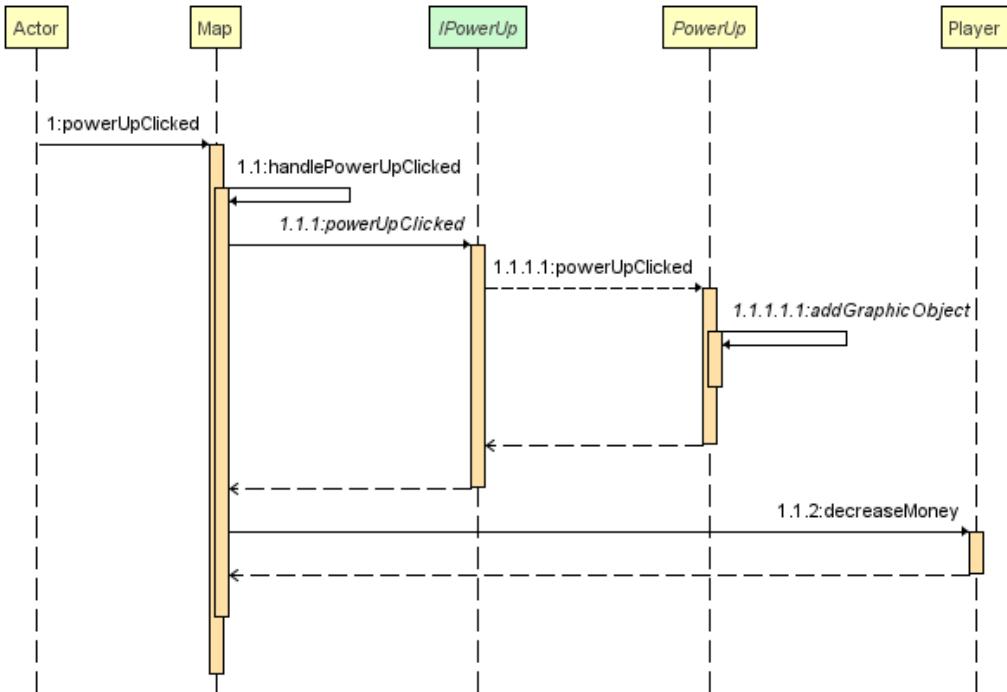
De utplacerade tornen kan sedan bli påverkad av spelaren genom att uppgradera tornet från uppgraderingsmenyn som uppkommer när man klickar på tornet. Ovanstående sekvensdiagram visar flödet för denna funktionalitet samt vilka underliggande metoder som kallas för detta.

Övrig manipulation av tornens funktionalitet kan göras genom ändring av target mode och aktivering av power ups.



Figur 2.3: Sekvensdiagram som visar metodanropet "changeTargetMode" med dess två möjliga utfall.

För att ändra target mode används samma panel som för att upgraderar tornet i men att spelaren istället klickar på en av två knappar som byter target mode i en karusell. Beroende på vilket håll användaren klickar åt snurrar denna karusell åt det hålet. Detta speglas i Figur 2.3 där man längst in ser de två utfallen beroende på vilken av knapparna man klickat.



Figur 2.3: Sekvensdiagram som visar metodanropet "powerUpClicked" för att aktivera en power up.

För aktivering av power ups använder spelaren knappar som ligger under tornmenyn till höger. Spelet erbjuder tre stycken val varav två är en tidsbaserad effekt som påverkar tornens funktionalitet medan den tredje direkt påverkar virusen. Sekvensdiagrammet ovan visar hur en normal sekvens ser ut efter trycket för aktivering av power up.

När spelaren känner att man antingen kört tillräckligt eller vunnit/förlorat kan spelaren avsluta spelet när som helst genom att navigera tillbaka till huvudmenyn och därifrån klicka på avsluta spelet, vilket tar användaren tillbaka till skrivbordet.

För renderingssystemet till applikationen används *libGDX* biblioteket. Klasser relaterade till detta ligger i View modulen och körs oberoende av Model-modulen. Detta innebär att renderingen av skärmelementen och uppdateringen av spelets inre funktionalitet görs separat vilket gör att antalet renderade bilder per sekund inte påverkar uppdatering av modellen tillsammans med en rad andra fördelar.

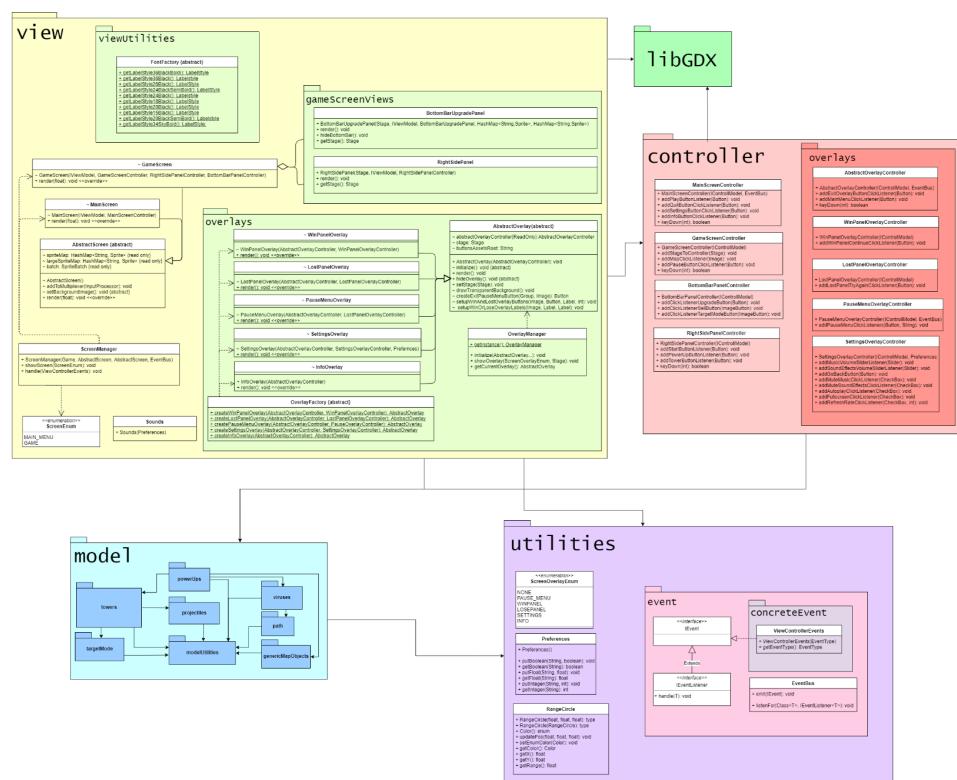
För att få detta system till att fungera har MVC:n satts upp genom att tillåta View fråga Model efter de objekt som ska renderas, därefter Model skickar den relevanta datan till View för att bli visad. Detta resulterar i att View under normala körförhållanden frågar Model efter objekt att rendera 60 gånger per sekund medan Model oberoende av nuvarande uppdateringshastighet/hastighet på spelet skickar

objekten direkt till View:n för rendering. Detta gör det också mycket lätt att öka antalet bilder per sekund som spelet uppdateras i under inställningarna utan att övriga Model modulen behöver uppdatera snabbare då samma hastighet kan användas.

# 3 Systemdesign

Programmet Chalmers Defense följer designmönstret MVC för arkitekturen, vilket står för Model-View-Controller. Det är ett designmönster för att dela upp en visuell applikation i tre moduler och det skapar en god modulär design för applikationen, samt minskar komplexitet.

## 3.1 MVC



Figur 3.1: Paket-diagramm över programmet

Arkitekturen är designad för att skapa ett modulärt program och därmed följa designprincipen "Separation of Concern". Detta genom att uppdelningen i de tre

huvudsakliga modulerna baseras på vad som hanteras och vad för typ av innehåll det är, exempelvis View-modulen som har hand om allt GUI relaterat eller Controller som enbart består av olika "Listeners" som lyssnar och skickar metodenrop till modellen.

### 3.1.1 Relationer mellan modulerna

Relationerna till model sker främst via gränssnitt. Både View och Controller använder ett eget gränssnitt för att komma i kontakt med Model, vilka Model implementerar. Detta gör att man gömmer funktionalitet som ej används bakom gränssnitten så att de ej kan nås. Gränssnitten gör det också lättare att kunna byta ut konkreta lösningar i framtiden om så skulle viljas utan att behöva ändra övrig implementation inom applikationen.

Mellan View och Controller modulerna valdes det att inte ha några gränssnitt eftersom de olika kontroller klasserna främst innehåller egna metoder som bara används av en viewklass. Det hade därför varit opraktiskt att skapa egna gränssnitt för varje kontroller klass som sedan varje enskild viewklass ska använda sig av när metoderna i gränssnittet då enbart hade varit exakt samma som för den konkreta kontrollerklassen som implementerar gränssnittet.

Inuti huvudmodulerna och framförallt Model modulen utgörs de inre relationerna mellan delpaket av beroenden till gränssnitt och Factory-klasser. Ett bra exempel på detta är den inre paketet "towers" i Model modulen där enbart gränssnittet *ITower* och fabriksklassen *TowerFactory* är synliga för utomstående användare av klasserna.

## 3.2 Model

Model är en oberoende modul. Det innebär att model kan klara sig på egen hand och dess View och Controller är utbytbara. Modellen står för spelets logik och hur de olika inre delarna samspelear med varandra för att utgöra spelet.

### 3.2.1 Intern arkitektur

Den interna arkitekturen i Model är i sig indelad i paket, samt har några självstående klasser och gränssnitt. Detta är för att fortsätta på en modulär design, även i lägre nivåer av programkoden. Processen där man delar upp ett system i flera små bitar kallas för "decomposition"(upplösning). Modulära system som denna är uppdelade genom "decomposition" i flera olika nivåer, som exempelvis klasser, komponenter, paket och subsystem. Detta leder till att systemet följer principen om abstraktion,

vilket innebär att visa så lite information som möjligt för att lösa upp de olika dependencies som existerar i programmet.

### 3.2.2 Skapandet av objekt

Inom modellen finns objekt som står för programmets funktionalitet. Huvudingången sker genom klassen *model* som delegerar arbetet vidare till andra klasser inom modulen. De övergripande klasserna inom modellen är *Player*, *Rounds*, *Map* och *SpawnViruses*. *Player* tar hand om spelarens liv och pengar genom metoder som kan modifiera detta. *Rounds* håller koll på vilken runda spelet befinner sig på samt vilken runda som man vinner vid. *Map* tar hand om spelplanens objekt, vilket består av projektiller, torn, *GenericMapObject* och virus. Utöver dessa saker har *Map* en referens till ett *IPath* objekt som används för kollision checkar när man placerar ut tornen på kartan samt en lista över de power-ups som finns. *SpawnViruses* används för att spawna korrekta sekvenser av virus som sedan åker ut på banan.

Objekten i Model modulen skapas främst vid uppstart av programmet då de relevanta klassernas objekt skapas i samband med att *Model* klassen skapas.

Undantag av detta utgörs av Torn, Projektiller, *GenericMapObject* och Virus objekt som alla skapas efter uppstarten av programmet. Torn objekt skapas genom att spelaren drar ut torn från högermenyn i spelvyn medan Projektiller objekt skapas av Torn klassen när en runda är aktiv. *GenericMapObject* skapas av när de olika powerupsen aktiveras. Virus objekten skapas av *SpawnViruses* klassen när en runda har startats.

### 3.2.3 Objekt och positioner på spelkartan

Objekt som ska ha en position på spelplanen implementerar gränssnittet "*IMapObject*". Detta gränssnitt består av metoder som hämtar relevant information såsom position, höjd och bredd samt rotation från objektet för att senare användas av View modulen för rendering. Viktigt i gränssnittet är också en metod som hämtar vilken nyckel som ska användas för att hitta korrekt Sprite som representerar objektet i View:n.

För att påverka objektens position används olika sub-gränssnitt till "*IMapObject*" som bidrar med metoder för att kunna uppdatera relevant information inom objektet där position ofta ingår. Objekten hanterar då uppdateringen genom att på sitt eget sätt uppdatera positionen på sig själv utefter eget sätt.

Detta beroende på sub-gränssnitt för uppdatering av de enskilda objekten, samt ett huvudgränssnitt som används för positionering, gör att man tar bort möjligheterna för View modulen att kunna ändra på objektets data. Detta gör att View följer designprincipen separation of concern eftersom det bidrar till att hålla isär saker som på ett abstraktionsplan inte har med varandra att göra. Detta tillsammans med att alltid skicka en kopia av alla *IMapObjects* till viewn gör det svårare att av misstag råka ändra och modifiera Modellen från View modulen av programmet.

### 3.2.4 Utåtgående dependencies från Model

Model har två utåtgående dependencies. Den första är till libGDX. Funktionalliteten som används är *GameTimer* och *JSONParserWrapper*. Dessa är abstraherade från libGDX så långt det går genom gränssnitt och att libgdx klasserna är omslutna i egna objekt.

Den andra utåtgående dependency vi har är till ett Utilities-paket. Utilities används för att underlätta för Model att göra vissa saker. Dessa verktyg är skapade av oss och är inte externa. Model har även ett intern utilities-paket vid namn *modelUtilities*, som endast innehåller klasser model använder. Däremot är den utilities ligger externt är en annan och denna innehåller klasser som fler moduler kan använda.

### 3.2.5 Models egna utilities-paket

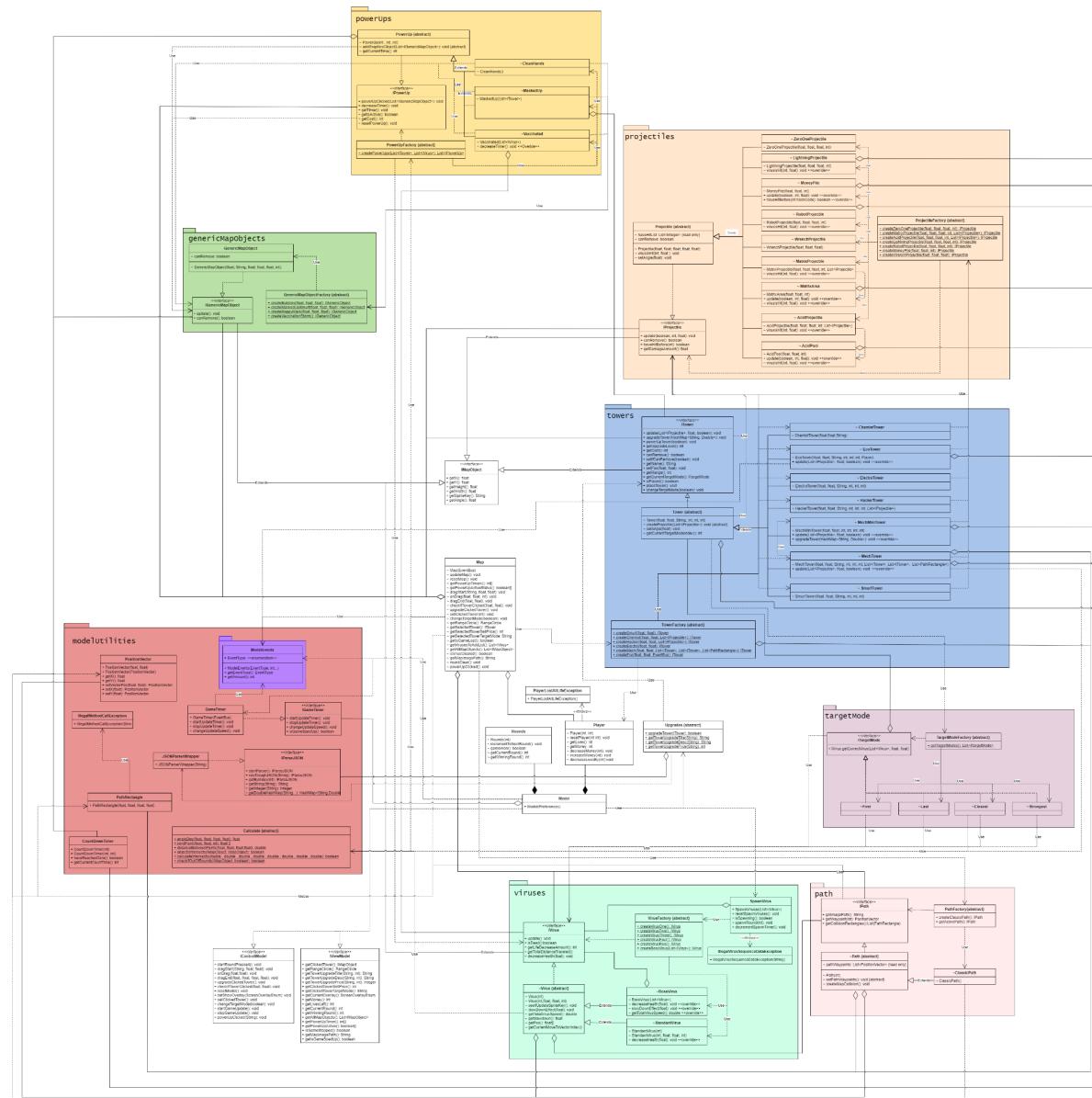
Model har som tidigare nämnt ett eget utilities-paket vid namn *modelUtilities*. Detta innehåller:

- **Calculate:** Hjälper till med diverse beräkningar.
- **CountDownTimer:** Innehåller ett tidtagarur som kan användas av alla klasser som behöver en.
- **GameTimer:** Innehåller metoder som är relaterade till spelets timer. Den omsluter ett timerobjekt i klassen. Timern kallar sedan på en uppdateringsmetod i den givna modellen.
- **IGameTimer:** Ett interface till GameTimer som Model klassen beror på.
- **PositionVector:** Representerar en punkt / vektor med hjälp av koordinater. Objekten av klassen är efter skapandet icke-muterbara och kan endast ändras via "mutate-by-copy".
- **PathRectangle:** Omsluter en javarektangel och gör den klassen icke-muterbar. Gör detta genom att ta in värdena i en konstruktor men ger sedan ingen möjlighet till att ändra dess värden.

- **IllegalMethodCallException**: Denna klass innehåller ett custom exception för PathRectangle.
- **JSONParserWrapper**: Agerar som en omslutare för libgdx JSON parser.
- **IParseJSON**: Ett interface till JSONParserWrapper som Upgrades klassen beror på.

### 3.2.6 Relationen mellan designmodellen och domänmodell

Domänmodellen och designmodellen är tydligt sammanhangande. Bortsett från GUI-delen i domänmodellen, som tillhör en annan modul, kan varje klass i domänmodellen återfinnas i designmodellen. Relationer mellan klasser i domändiagrammet återspeglas också i designmodellen. Skillnader mellan de två modellerna är att det är en stor skillnad på detaljnivå och därmed kan många fler komponenter hittas i designmodellen.



Figur 3.2: Klassdiagram över Model-modulen. Även kallat designmodellen.

## 3.3 View

View är modulen för allt som skall ”presenteras” för användaren. Detta innehåller allt användaren kan se och höra. Chalmers Defense View innehåller sju klasser och tre interna paket, gameScreenViews, overlays och viewUtilites. GameScreenViews innehåller två klasser. Overlays innehåller åtta klasser och viewUtilities innehåller en klass.

### 3.3.1 Klasser i Views modulen

ScreenManager är huvudklassen där alla vyer hanteras. ScreenManager får in alla vyer den ska hantera när *initialize* metoden körs. För att byta mellan vyer används metoden *showScreen* som kallas genom en eventbuss och byter till respektive vy.

- **AbstractScreen** är en abstrakt superklass som håller i alla gemensamma metoder för alla vyer. Klassen ärver från libGDX Stage och implementerar libGDX Screen gränssnittet. Det gör att klassen och alla subklasser kan renderas med hjälp av libGDX.
- **GameScreen** är en konkret vy som ärver från AbstractScreen och har som uppgift att visa all spelrelaterad information på skärmen. Detta inkluderar en karta över spelplanen, alla IMapObjects som ska visas på kartan och ett HUD där spelinformation visas.  
HUD:en består av tre delar. Den ena är *RightSidePanel* som nämndes ovan och det andra är den nedre delen av skärmen. När *BottomBarUpgradePanel* inte renderas visas en framstegsmätare för hur långt spelaren kommit i spelet, samt text som specifikt säger vilken runda användaren befinner sig på. Framstegsmätaren har även milstolpar som visar när nästa typ av virus kommer.
- **MainScreen** är också en konkret vy som ärver från AbstractScreen. Denna klassen är programmets ingångspunkt och innehåller fyra funktionella knappar. Via dessa kan användaren navigera sig till inställningar, spelinformation eller spelvyn, alternativt stänga ner programmet.
- **ScreenFactory** är en abstrakt klass och hanterar skapandet av olika vyer.
- **Sounds** är en klass som innehåller musiken för programmet.

### 3.3.2 Klasser i GameScreenViews paketet

GameScreenViews innehåller två paneler som används av GameScreen. Dessa paneler är uppdelade i två klasser. Den ena klassen är *RightSidePanel* och innehållar utbudet av torn, powerups och startknapp. Startknappen fungerar även som en uppsnabbnings-knapp.

Den andra klassen är uppgraderings panelen *BottomBarUpgradePanel*. Denna panelen agerar som en informationsruta för uppgraderingar, samt erbjuder användaren möjligheten att ändra tornets targetmode eller sälja tornet.

Uppgraderingar utförs genom att klicka på rutorna i denna vy. GameScreen bestämmer själv när denna panelen ska renderas utifrån en variabel *selectedTower* som hämtas från map genom model.

### 3.3.2 Klasser i overlays paketet

Paketet overlays innehåller alla olika overlays för programmet, samt klasser för skapandet av dessa. Klassen *OverlayFactory* är den som skapar själva objekten. Sedan styr *OverlayManager*-klassen vilken vy som skall visas beroende på ett *ScreenOverlayEnum*.

Overlays används för att visa en ruta mitt på skärmen med information till användaren. Detta används när en spelare vinner, förlorar, pausar eller vill läsa information om spelet. Varje overlay skiljer sig i design men har liknande funktionalitet. Därmed ärver alla dess från en klass vid namn *AbstractOverlay*, vilket innehåller gemensam funktionalitet mellan de olika klasserna, exempelvis kryssknappar och mörka bakgrunder.

De olika overlays som finns är:

- **InfoOverlay**

Håller kort information om spelet och vilka kortkommandon som finns i spelet.

- **LostPanelOverlay**

Kommer upp när spelaren förlorar spelet och ger användaren två val.

Antingen försöka igen eller gå till huvudmenyn.

- **WinPanelOverlay**

Kommer upp när spelaren vinner spelet och ger användaren två val.

Antingen fortsätta spela eller gå till huvudmenyn.

- **PauseMenuOverlay**

Visas i GameScreen när spelet pausas. Innehåller tre val för användaren. "Continue" som fortsätter spelet och stänger overlay. "Settings" som öppnar inställningar overlay. "Quit" som byter skärm till huvudskärmen.

- **SettingsOverlay**

Kan öppnas både i huvudvyn och spelvyn. Ger användaren möjlighet att ändra ljudnivå, sätta på så att runder startar automatisk, ändra uppdateringsfrekvens och byta till fullskärm.

### 3.3.3 Views egna utilities-paket

Klassen `viewUtilities` innehåller just nu endast en klass som är en hjälpklass gällande typsnitt i programmet. Om fler hjälpklasser vill implementeras för View i framtiden kan de läggas i detta paket.

- **FontFactory**

Underlättar att sätta `LabelStyle` för typsnittsanvändning. Denna klass används i View-modulen.

## 3.4 Controller

Controller-modulen behandlar all användarinput i programmet och skickar vidare till Model för hantering. Modulen innehåller fyra klasser, samt ett paket vid namn `overlayControllers`. Paketet innehåller kontroller till olika overlays i View. Dessa kontrollerklasser innehåller klick lyssnare och hjälper till att manipulera olika overlays.

De kontroller Chalmers Defense just nu har är:

- **AbstractOverlayController**

Håller gemensamma klick lyssnare och input events för alla overlays.

- **LostPanelOverlay**

Specifika klick lyssnare för LostPanelOverlay till tex försök igen knappen.

- **PauseMenuOverlayController**

Specifika klick lyssnare för PauseMenuOverlay till de tre knapparna "Continue", "Settings" och "Quit".

- **SettingsOverlayController**

Innehåller klick lyssnare för alla volymreglage, kryssrutor och övriga knappar.

- **WinPanelOverlayController**

Specifika klick lyssnare för WinPanelOverlay till tex fortsätta spela knappen.

De övriga klasserna i modulen är följande:

- **BottomBarPanelController**

Hanterar all input som kommer från bottenspanelen. Tar inputs från exempelvis uppgraderingsknapparna och knappen för att sälja torn.

- **MainScreenController**

Hanterar all input som kommer från startvyn. Här tar den inputs från exempelvis startknappen som tar spelaren till spelvyn.

- **GameScreenController**

Hanterar all input som kommer från spelvyn (exkluderat paneler). Inputs från pausknappen lyssnas efter här som ett exempel.

- **RightSidePanelController**

Hanterar all input som kommer från sidopanelen. Den lyssnar efter inputs från exempelvis tornknapparna, som används för att lägga till torn.

## 3.5 Utilities

Utilities är ett paket innehållande "assisterande" klasser. Att dem assisterar menas med att de ofta genomför en uppgift, som sedan en annan klass använder för att utföra en annan uppgift. Chalmers Defense har flera utilities paket beroende på var klasserna används men denna innehåller klasser som används i flera moduler.

De klasser Utilities-paketet innehåller är:

- **RangeCircle**

Används för att rita ut räckviddscirkeln runt det valda tornet. Klassen innehåller metoder om placering och storlek på cirkeln.

- **Preferences**

Lagar och gör inställningar tillgängligt i de olika delarna i programmet som behöver det.

- **ScreenOverlayEnum**

Ett enum för att beskriva vilket overlay som ska visas på skärmen.

- **"event" paket**

- **EventBus:** Möjliggör kommunikation genom en gemensam eventbuss där klasser kan lägga till eller hantera event.

- **IEvent:** Ett gränssnitt som alla olika event implementerar.

- **IEventListener:** Gemensamt gränssnitt för alla eventbuss lyssnare/publicerare.

- **"concreteEvents" paket**

- **ViewControllerEvent:** Ett even som används för att byta vyer i spelet

## 3.6 Designmönster

Chalmers Defense har använt sig av objektorienterade designmönster där de har varit applicerbara. Implementerade designmönster beskrivs nedan.

### 3.6.1 Model-View-Controller (MVC)

MVC är ett arkitekturmönster som går ut på att dela upp programmet i tre moduler. Dessa är Model, View och Controller. Model hanterar datan, View hanterar det presenterade innehållet och Controller hanterar användar-inputs. Vi använder MVC som vår övergripande programstruktur. Detta gör att programmet bättre följer designprincipen "High cohesion low coupling".

I Chalmers Defense uppdaterar Model all objektsdata, View hämtar kontinuerligt datan från Model och använder det för att representera det visuellt och till sist så lyssnar Controller efter användar-input och delegerar vidare till rätt metoder i Model.

### 3.6.2 Factory Method

*Factory method* är ett skapande mönster. Det är ett "interface" för att skapandet av ett objekt i en superklass. Subklasserna alternerar vilken typ som skall skapas och implementeras genom att göra en factory-klass med statiska metoder, som beroende på metod skapar olika typer av objekt. Den returnerade typen av objekten ska vara en abstraktion av objektet med nödvändig funktionalitet för klienten. I Chalmers Defense används factory method på flera ställen. Det används för skapandet av torn, projektiler, powerUps, virus, nya banor och fonter. Användningen gör att antalet sammanbindningar minskar för programmet.

### 3.6.3 Template method

*Template method* är ett "Beteende"-mönster som används när klasser har stora bitar kod gemensamt men med små saker som skiljer i dess sammanhang. Mönstret utnyttjar *implementation inheritance* då det flyttar gemensam kod till en abstrakt klass och det är abstrakta metoder som representerar variationerna. *Template method* används inom applikationen när ett torn ska skjuta en projektil. De flesta torn använder samma huvudmetod för hanteringen men ska skjuta olika projektiler vilket implementeras i de konkreta torn klasserna individuellt.

### 3.6.6 Singleton

Singleton är ett mönster som används när man bara vill ha en möjlig instansvariabel(en singleton) för ett objekt/modul. Det implementerar detta genom att skapa en privat static variabel av samma typ som klassen, gör konstruktorn private, och sen en statisk metod som returnerar den statiska variabeln, och skaparen ny om det inte finns någon. Singleton löser problemet när man måste skapa ett objekt flera gånger och är då användbart vid skapandet av databas-connections eller filhanterare. Chalmers Defense använder detta mönster för OverlayManager så att det lätt går att lägga till overlays.

### 3.6.7 Eventbus

Eventbus är ett arkitekturellt designmönster med många användningsområden. Idén är att skapa en typ av direktförbindelse där komponenter kan ansluta sig för att få någon typ av notifikation om ett event. Därmed påminner Eventbus om mönstret Observer och delvis Mediator då komponenter subscribar till förbindelsen och att informationen färdas genom en och samma punkt. Det finns ingen specifik implementationsstruktur för mönstret, utan det är snarare idén om dess flöde som är poängen. Chalmers Defense koden använder sig just nu av en egen Eventbus, för att exempelvis kommunicera mellan Map och Player när Map vill lägga till eller ta bort pengar från Player. Eventbus används också för kommunikation och uppdatering mellan GameTimer och Model, och till sist mellan View och Controller för att byta vyer.

## 4 Ihållande datahantering

Chalmers Defense datahantering består av att hämta data men skriver ingen data. Datat vi hämtar är då bilder, skins, musik och JSON-filer.

Bilder lagras som png filer i mappen "/core/assets" och är sedan uppdelade i fler mappar, så som towers, viruses och projectiles. Musiken lagras också i ".wav" format i mappen "/core/assets".

Alla knappar i applikationen är skapade med verktyget Skin Composer. Verktygen genererar tre olika filer som lagras i en mapp med ett passande namn för knappen. Dessa tre filer är vad som kallas skins och används sedan av libGDX biblioteket för att skapa en knapp med respons.

Uppgraderings data för varje torn sparas i en JSON fil. Filen är uppdelad efter torn namn och varje namn har sedan en JSON lista med uppgraderingar. Det gör det enkelt att lägga till fler uppgraderingar i filen i framtiden.

## 5 Kvalité

### 5.1 Tester

Kvalitén för programmet har mäts på ett flertal sätt. Den första kvalitéskontrollen görs redan i programmet via JUnit tester. Dessa kan hittas i ett test-paket, vilken ligger under huvudmappen "desktop". Testerna riktar sig mot Model och kollar ifall delar av kodem och metoder fungerar gentemot deras syfte. Chalmers Defense tester täcker över 99,7% & av programmet.

Ett annat sätt vi kontinuerligt testar programmet är via Travis. Detta program möjliggör kontinuerlig och säker utveckling. Testerna vi skapat via JUnit körs varje gång en utvecklare pushar upp till GitHub och dessa tester täcker hela programmet. Detta gör att eventuella fel upptäcks snabbt och kan korrigeras innan de orsakar någon större skada.

### 5.2 Kända problem

#### 5.2.1 Mac

Trots tester har ett antal problem identifierats i programmet. Det första kända problemet är att applikationen inte fungerar fullt ut till Mac operativsystem. Programmet startar och det går att köra det normalt men det uppstår ett problem med cirklarna runt tornen. Dessa blir aningen ovala istället för cirkulära, samt att positioneringen blir fel. Orsaken till detta är troligen Mac:arnas annorlunda skärmupplösning. Dessvärre har gruppen inte haft tillgång till en Mac under utvecklingsprocessen och därmed har orsaken inte fullständigt blivit identifierat, vilket gör att problemet inte har lösats.

Ett annat problem för Mac-användare gäller fullskärm och fönsterstorlek. När programmet startar befinner sig användaren i ett fullskärmsläge. Försöker Mac-användaren gå ur fullskärmsläget till fönsterläget kraschar programmet. Den bakomliggande orsaken till detta är inte känt och har därmed ingen lösning för problemet.

Linux operativsystem har inte kunnat testas eftersom gruppen inte heller har tillgång till detta operativsystem men ovan problem för Mac kan troligtvis förväntas.

### 5.2.2 Textkvalité i fönsterläge

Ett annat känt problem är skalning på text när man ändrar fönsterstorlek.

Programmet körs som sagt initialt i fullskärmsläge men ifall användaren vill spela i fönsterläge kan det ibland bli svårt att se vad det står. Orsaken bakom detta är hur libGDX skalar om och därmed kan problemet inte lösas, om nuvarande funktionalitet ska bibehållas.

Ifall gruppen skulle välja att ändra programmets funktionalitet, skulle fasta upplösningar kunna implementeras och sedan rendera typsnitten och grafiken på nytt. Då hade det inte längre blivit problem med texten men funktionaliteten hade ändrats.

## 5.3 Analytiska testverktyg

STAN (Structure Analysis) är ett verktyg för att analysera program och dess kvalité. Se Chalmers Defense resultat bland bilagor.

## 5.4 Åtkomstkontroll och säkerhet

Inte applicerbart.

# 6 Förbättringar

## 6.1 Feedback från peer review som blev implementerade

Under arbetet gjorde en annan grupp en så kallad peer review på programmet där syftet var att förbättra designen av koden inom applikationen. Här kommer de förbättringarna som implementerades i koden.

*"Model exponerar många viktiga direktreferenser, vilket skapar en risk för alias problem. Detta kan förbättras genom att göra om getters i Model klassen att skicka tillbaka kopior av objekt istället för direkta referenser" .*

- Denna respons togs emot och infördes i koden. Det märktes att man kunde göra en till metod returnera en kopia av en lista, så det fixades.

*"Metoden 'createMapCollision' i Path är väldigt stor, bör använda funktionell dekomposition".*

- Kommentaren togs emot och hade i åtanke när vi gick igenom resterande kod för att dela upp större metoder.

*"Data för de olika tornen sätts i TowerFactory , vilket kan vara mer förvirrande än att sätta det i klassen för varje individuellt torn istället."*

- Detta infördes då vi höll med om att de borde bli flyttade för att göra det tydligare vilken data som tillhör vilket torn.

## **6.2 Feedback från peer review som ej implementerades**

Mycket av det som skrevs användes för att förbättrad koden i den objektorienterade aspekten, men det var några kommentarer som inte var helt korrekta. Därför handlar denna sektion om varför projektet inte använde sig av vissa åsikter som gruppen skrev.

*"Projektiler har många instansvariabler som inte är final, variablene borde sättas som final om de inte kommer ändrats efter att de har skapats"*

- Instansvariablerna som inte är final i projektiler kan inte vara det så som programmet är kodat. Exempelvis så har vi variabeln x som representerar projektilens x-position, denna variabel kommer uppdateras och därför ändras hela tiden när den skjuts av ett torn. Vi valde därför att inte göra något åt denna kommentar då vi redan har variablene till final om de kan vara det.
- Dock så kunde de vara final om vi från start designade koden med immutable objekts, där vi skapar nya objekt varje gång om exempelvis x variabeln skulle uppdateras. Men att skapa nya objekt varje gång är inte optimalt när det kommer till spel för då kommer det skapas allt för många objekt när allting ska röra på sig, vilket de gör hela tiden.

*“Projektiler har en variabeln ‘canRemove’ som endast används när projektiler ska bli borttagna från en lista som ligger i Map. Istället för att sätta variabeln till ‘true’, istället så bör metoden kedja objektet direkt till listan och skippa variabeln”*

- Det som nog missuppfattades här är att det inte är Map som sätter variabeln till true, det är projektilen själv. I de flesta fallen hade Map kunnat ta bort projektilen direkt när den vet att ett virus och projektil har träffats men eftersom vi har vissa projektiler som inte fungerar på det sättet kan man inte alltid göra det. Och för att undvika instanceOf som vi hade först så gjorde vi så att vi har att projektilen sätter variabeln och att Map bara kollar om den kan ta bort eller inte.

*“Metoden för att ändra ‘targetMode’ är inte utökbar. Den skulle kunna följa ett state mönster för att förenkla användningen av den”*

- De olika “targetModes” ligger just nu i en lista som helt enkelt skiftas igenom när man byter “targetMode”, detta anser vi redan är en typ av State pattern. Beroende på vilket index vi är i listan så kommer tornet att agera på ett annorlunda sätt. Utöver det så anses åsikten om att den inte är utökbar inte helt korrekt heller. Det är egentligen väldigt enkelt att lägga till ett nytt “targetMode”, man skapar en ny klass för den och sen lägger man den i listan i targetModeFactory och då är det klart.

*“Informationen för att spawna virus är inte väldigt återanvändbar och bör vara någon typ av algoritm som räknar ut när och vilket virus den ska spawna”*

- Vi kan förstå varför det ser dåligt ut då all information är direkt inskrivet med data och använder sig inte av någon algoritm. Dock, eftersom vi vill ha ett bestämt antal virus och ett visst typ av virus per runda så måste det vara på detta vis. I vårt fall så ska inte virus spawna randomiserade och därför är all data redan inlagd och behöver inte någon algoritm.

*“Mestadels av koden är täckt med JavaDoc, men model klassen, som har många publika metoder, saknar dokumentation helt och hållet”*

- JavaDoc finns för alla dessa metoder de nämner, dock så ligger de i interfacet till model och behöver därför inte skrivas en gång till. De flesta publika metoder inom model klassen befinner sig i interfacen IViewModel och IControlModel med javaDoc, och då helt enkel har en override i klassen.

## 6.3 Förbättringsmöjligheter

Chalmers Defense har kommit långt kodmässigt och inkluderar de flesta funktionerna som var planerade att ha, dock så finns det förbättringar som kan göras, men har inte hunnits implementerats under tidsramen för projektet. Problemen i koden och tankar på förbättringar är som följande:

### 6.3.1 Utöka uppgraderingar för projektiler

Spelet har en funktion som tillåter spelaren att uppgradera tornen som placeras ut. Om man i framtiden skulle vilja lägga till flera uppgraderingar så är det inte så enkelt som det borde vara. Både för att ändra och lägga till uppgraderingar måste man gå till flera ställen i koden för att göra det. För torn är det relativt enkelt att lägga till uppgraderingar då all den datan ligger i en fil, så det är enkelt att utöka den. Problemet dock kommer till projektilerna, de har sin egna kod i de respektive klasserna och hämtar tornets uppgraderings nivå och bestämmer själv hur de ska uppgradera sig. Så om man exempelvis ville ha en ny uppgradering för "electroman" tornet som gör att tornet blir snabbare och projektilen studsar mellan virusen tio gånger så måste man gå till projektilklassen och lägga till kod för den specifika uppgraderingsnivån.

Lösningen till det är att uppgradera projektilerna på samma vis som vi uppgraderar tornen. Om man skapar en JSON fil som innehåller data för uppgraderingar för projektilerna kan man enkelt gå och lägga till/ändra uppgraderingar där istället för att gå till de enskilda projektilklasserna och lägga till helt ny kod.

### 6.3.2 Lägga till torn med liknande attacktyper

Ett annat aktuellt problem är om man skulle vilja lägga till flera torn som har samma "attacktyp" som ett befintligt unikt torn så kommer det bli kod duplikation. Om vi tar "electroman" tornet som ett exempel. Projektilen tillhörande det tornet studrar mellan olika virus, vilket ett vanligt torns projektil inte gör. För att åstadkomma detta så har dess projektil en unik kod som gör att det studrar ett specifikt antal gånger, vilket motsvarar attacken för endast "electroman". Om man nu skulle vilja ha samma projektilegenskap för ett nytt torn så måste man skriva exakt samma kod där men bara ändra vissa värden, vilket då leder till kodduplikation.

För att fixa detta så skulle koden kunna använda sig av delegering". Koden för en specifik egenskap hos en projektil kan flyttas ut till en abstrakt klass som kan

exempelvis anges "bouncingProjectile", denna klass kommer innehålla algoritmen som gör att en projektil studsar. En konkret klass som vår projektil för "electroman" tornet kan därefter delegera till denna klass och skicka in exempelvis antalet studsar som den vill göra och då tar "bouncingProjectile" hand om det. Nu kan ett nytt torn som vill ha studsande projektiller ha sin projektilklass delegera till denna klass. På så sätt har vi nu reducerat duplikation av kod och gjort det lättare att lägga till nya torn med olika attacktyper.

### 6.3.3 Utöka antalet banor(paths)

Ännu ett utökbarhetsproblem med koden är att det går inte att använda sig av flera olika paths vilket betyder olika banor. Just nu så har Chalmers Defense bara en bana som representeras av en path, och sätter den som valda banan direkt i Map klassen. Koden har sätt att lägga till fler paths då vi har ett pathFactory där man skulle kunna lägga in fler. Det enda som krävs för att lägga till fler banor är att skapa en ny klass för dem och sedan lägga in dem i pathFactory. Problemet dock är att det inte finns något sätt att välja bana, varken i GUI:t eller i koden. Map sätter den aktiva banan direkt till den enda vi har, som är *ClassicPath()*.

För att försäkra att man skulle kunna lägga till fler banor i framtiden skulle programmet behöva implementera ett sätt att välja mellan olika banor. GUI:t måste därför utökas med ännu en vy så att användaren kan välja bana, och därefter måste koden hämta vilken bana spelaren klickade på och sätta den som aktiv bana. Denna ändring är inte svår men det tar tid att få upp en ny vy så därför har inte detta projekt hunnit med att implementera det.

### 6.3.4 Ej fullständig användning av EventBus

Som tidigare nämnt i rapporten använder sig Chalmers Defense utav en EventBus. På grund av brist på tid så kunde vi dessvärre inte införa användning av EventBus i hela projektet. Exempelvis så hade vi velat att torn, virus, projektiler, och övriga objekt läggs till i listan i Model/Map genom en EventBus så skulle vi inte behöva skicka med listan till respektive objekt.

Problemet dock med EventBus implementationen som koden har nu är att det finns ett tomt gränssnitt vid man *IEvent* som de konkreta eventklasserna implementerar. Gränssnittet behövs för att EventBus klassen ska veta att det som skickas in i "emit" metoden faktiskt är en typ av event. Dock så skulle gränssnittet fyllas på med

metoder om projektet hade implementerat flera events, men som det är just nu finns det ingenting att lägga in där så det fick vara tomt.

### 6.3.5 Hårdkodade värden

Att ha hårdkodade värden i koden när man egentligen inte behöver det är inte det bästa. För att kontrollera när projektiler och torn går utanför spelskärmen så används just nu en metod som har hårdkodade värden för storleken på skärmen. För att förbättra detta så skulle vi kunna lägga in värdena i vår *Preferences* fil och sedan kan Model hämta värdena därifrån när den behöver dem.

### 6.3.6 Stor variabel fyllt med ren data

Sedan så har vi ett litet mer av ett stilproblem i koden. Vår applikation använder sig av hårdkodad data för att spawna rätt virus varje runda, detta data sparar just nu som en enda stor matris i *SpawnViruses*. Denna data bör istället sparas i någon separat fil som koden sedan hämtar så slipper vi ha så stora variabler och ren data i koden.

### 6.3.7 Har ej flera ljudfiler och ljudeffekter

Det finns saker vi hade även gärna velat implementera. Just nu har vi bara en ljudfil för bakgrundsmusik i applikationen, egentligen hade vi velat ha fler. En förbättring som hade gjort hela spelet roligare är att lägga till flera ljudfiler och fixa ett system som kan byta mellan dem. Sedan så skulle vi också vilja implementera ljudeffekter, som just nu inte existerar. Här skulle vi kunna använda en EventBuss för att spela upp ljudeffekter vi rätt tidpunkt.

# 7 Referenser

## 7.1 Verktyg

- Github
  - Webbaserad och centraliserad lagring av versionshistorik för programvaruutvecklingsprojekt.
- Slack
  - En kommunikativ plattform för att diskutera intern information gällande projektet.
- Figma
  - Webbaserad plattform för att designa gränssnitt.
- Zoom
  - Verktyg för digitala möten.
- Google Drive
  - Gemensam molntjänst för att skapa och dela filer
- Google Docs
  - Textredigeringsprogram, som tillåter delning och parallellt arbete med live-uppdateringar.
- Trello
  - Projektplaneringsverktyg
- Diagrams.net
  - Webbsida för att skapa och redigera UML-diagram
- JUnit
  - Java bibliotek samt verktyg för att testa skriven kod.
- Travis CI
  - Verktyg för att automatisera byggandet och testandet av JUnit tester.
- Javadoc
  - Ett system som autogenererar kommentarer från koden på ett standardiserat sätt till en sammanställning.
- Skin Composer
  - Ett verktyg för att skapa knappar med visuell respons.

## 7.2 Bibliotek

- JUnit
  - Är ett ramverk som programmerare i Java använder för enhetstestning.
- libGDX
  - Är ett gratis plattformsoberoende Java- spelutvecklingsramverk. Det är ett javabibliotek som ger stöd för bland annat grafik, inmatningar och ljud.

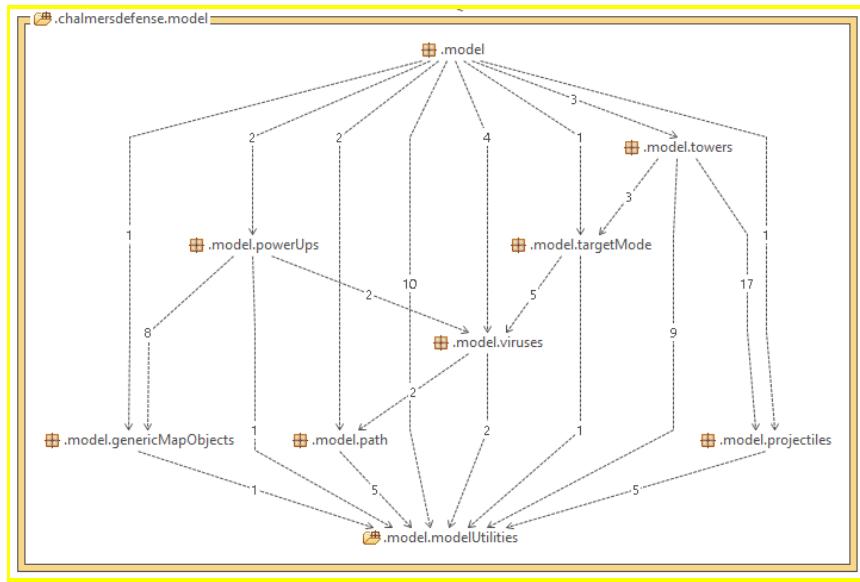
# A - GitHub

Länk till GitHub: <https://github.com/Danill01/TDA367-OO-Projekt>

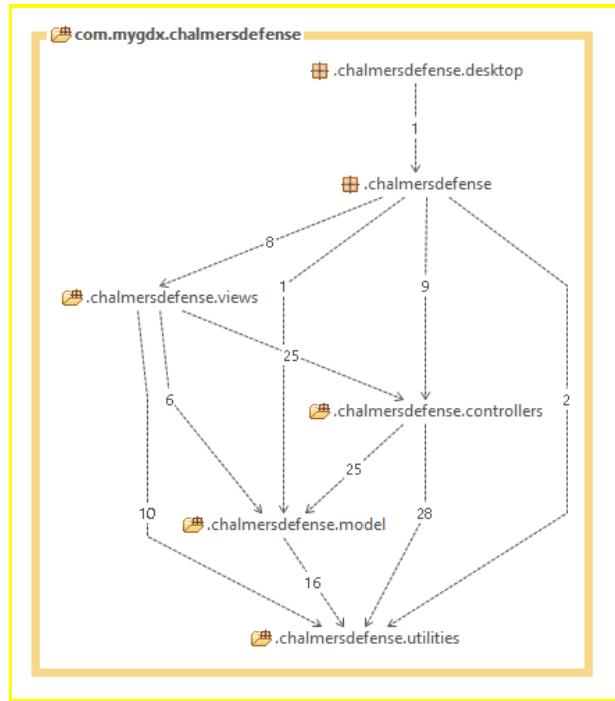
## B - Travis

Länk till Travis: <https://app.travis-ci.com/github/Danilll01/TDA367-OO-Projekt>

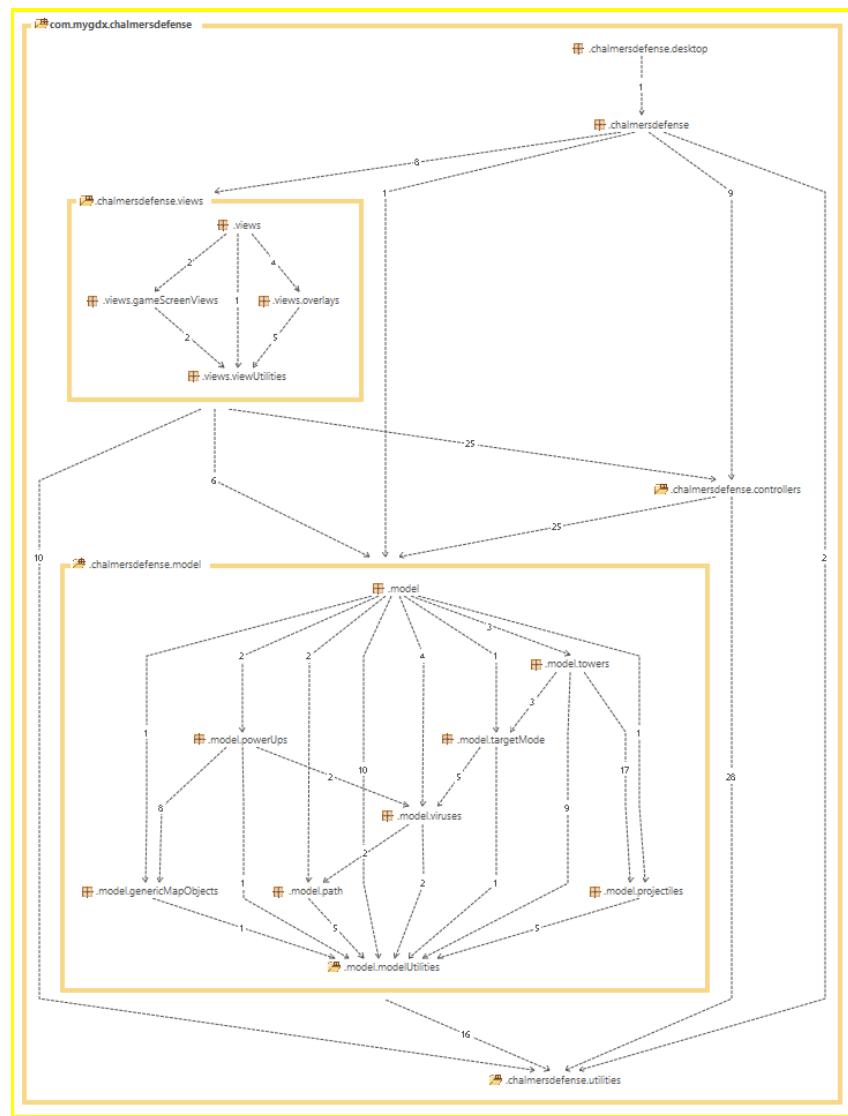
## C - Resultat från analytiskt test STAN



Figur B.1: Struktur för Model



Figur B.2: Övergripande struktur



Figur B.3: Struktur över hela programmet

# Peer Review av MailMe

Skriven av:



MailMe är ett trevligt och användarvänligt program med mycket potential. Det är imponerande att ta sig an utmaningen att försöka göra en mailklient och MailMe's prototyp är lovande.

## Kodstil och dokumentation

Vad det gäller kodstil så är de flesta namnen på metoder, klasser och variabler bra, men det finns undantag. Till exempel skulle "getEmailServiceProvider" i EmailServiceProviderfactory kanske kunna heta "createEmailServiceProvider" eftersom det är en Factory-klass som skapar objekt.

Användningen av JavaDoc är bra i allmänhet. Ibland skiljer sig dock strukturen på kommentarerna gällande *author*, *params* och *descriptions*, vilket skulle kunna göras mer enhetligt. Ibland saknas det också förklarande kommentarer över instansvariablerna i klasserna.

När det gäller undantag är det väldigt få klasser som hanterar undantagen direkt. Istället kastar många klasser upp undantagen genom metod kedjan. Undantag bör hanteras så nära källan som möjligt där de kastas. När ni använder många undantag kan det också vara bra att skapa egna undantags klasser som är mer beskrivande än "Exception".

## MVC

Under punkt 2 i MailMe's SDD står det om användningen av MVC och varianten *passive View*. Det är bra att ni har anpassat systemarkitekturen så att det passar syftet, ändå verkar det som att ni inte har följt eran modell/klassdiagram fullt ut. Enligt diagrammet skall det fortfarande finnas en View-modul/paket med innehåll men i det verkliga programmet existerar inte denna. View är för tillfället integrerad i Controller. Därmed kan det vara bra om man implementerar en "riktig" View-modul så att det följer systemarkitektur ni angett i erat diagram.

## Modularitet

Modulariteten i programmet är generellt rätt bra då packages är gjorda väl då eftersom de är uppdelade i logiska grupper. Däremot borde de kanske delas upp ännu mer (läs längre ned om *Separation of concern*).

Det är få abstrakta klasser och interfaces i programmet. Det resulterar i att klasser är beroende på delar av samma abstraktionsnivå, vilket bryter *Dependency Inversion Principle* som säger att klasser skall vara beroende av abstraktioner och inte konkreta implementationer. Därför borde fler abstraktioner och interfaces implementeras, där det kan finnas för att förbättra modulariteten.

## Designmönster

Bra användning av *Factory* mönstret, men en tanke är att kanske göra *Factory* klasserna abstrakta då metoderna inom dem är statiska. Vi ser i SDD att ni skrivit om *Aggregate* mönstret, men vi tror att förklaringen av användningen faktiskt passar bättre in på mönstret *Facade*. Er "TextFinder" skulle kunna ses som hanteraren och blir ingångspunkten för systemet. Användningen skulle göra att programmet bättre följer principen *high cohesion low coupling*, samt möjliggöra att göra de flesta klasser package private. Om ni inte redan följer det helt så kan det vara värt att implementera fullt ut.

## Designprinciper

### High cohesion, low coupling

När det gäller *coupling* har programmet en del beroenden mellan paketen model, controller och services. Det kan göra att programmet upplevs som mindre modulärt eftersom det är svårt att byta ut paket utan att ändra i alla andra paket.

Programmet har bra *cohesion* överlag. Men data paketet har få beroenden mellan klasserna. Det kan upplevas som låg *cohesion* och kanske bör flyttas från paketet Model till exempelvis *utilities* där alla delar av koden kan använda det.

### Open Closed Principle

Gällande *closed for modification* delen av principen följer applikationen inte det helt. Anledningen till detta är för att alla klasser är publika vilket gör man kan komma åt alla klasser oavsett var man befinner sig i koden. Det finns även publika instansvariabler som man väldigt lätt kan komma åt genom punktnotation vilket man vill undvika.

### Single Responsibility Principle / Separation of concern

Metoder gör ibland flera saker. Som exempelvis i *Model* klassen, *refresh()* metoden gör tre olika saker, som man ser i javaDocen som tillhör den. Metoder bör delas upp till flera små metoder som har hand om endast en sak.

När det kommer till separation of concern så bryts den av kontroller modulen då den hanterar både inputs och saker som en view ska göra. Kontroller just nu är som en sammanslagning av kontroller och view, vilket inte följer denna designprincip.

## Inkapsling och muterbarhet

Man kan kapsla in ett flertal av instansvariablene så de blir privata. Sedan finns det några instansvariabler som ligger i Model klassen som används endast av en kontroller. Alternativt så kan man flytta dessa variabler från Model klassen eller sätta dem som privata variabler med getters. Det är exempelvis inte jättebra att accounts ligger i en publik lista, vilket möjliggör extern åtkomst för dessa (inkl. lösenord).

Generellt sett är det viktigt att göra objekt icke-muterbara men det är nog extra viktigt när man gör en mejlklient, pga säkerhetsskäl. Just nu är det inte så stort fokus på detta, vilket kanske borde prioriteras inför slutinlämningen.

## Övriga anmärkningar

- Användandet av generiska- och parametriserade typer är bra.
- Det finns JavaFx i Model , vilket inte får vara där.
- Just nu så täcker testerna inte mycket i model modulen, vilket de bör göra mer.
- Det finns en extra måsvinge i Master.css som orsakar error.
- Istället för att bara serialisera och lagra lösenordet kan man kolla på att använda något krypteringsbibliotek som tex bCrypt.