Linköpings universitet | Institutionen för datavetenskap

Examensarbete på avancerad nivå, 30hp | Datateknik

2017 | LIU-IDA/LITH-EX-A--17/033--SE

Utveckling av tvådimensionellt online-spel i kategorin action-strategi för smartphones med fokus på spelupplevelse och prestanda

Development of two-dimensional online game in the category action-strategy for smartphones with focus on game experience and performance

Robin Andersson

Handledare : Kristian Sandahl
Examinator : Erik Berglund

LINKÖPINGS UNIVERSITET

Linköpings universitet
SE–581 83 Linköping
013-28 10 00 , www.liu.se

Abstract

Vid utveckling av spel till smartphones så krävs ofta flera olika implementationer för olika plattformar. Men det finns olika spelmotorer och ramverk till olika programspråk som kan kompilera samma kod till flera olika mobila plattformar. Kivy är ett sådant ramverk för programspråket Python.

I detta arbete användes prototypdriven utveckling i Kivy för att försöka höja spelupplevelsen av ett tvådimensionellt online-spel i kategorin action-strategi som tidigare fanns implementerat i GameMaker. Olika tekniker för att öka prestandan för spelet testades också. Det visade sig att med den bästa kombinationen av de tekniker som testades så blev prestandan med hög last cirka 29 bilder/s. Användarbarheten, spelbarheten och ljud-estetiken av spelet ökades medan nöjet och den personliga tillfredställelsen från spelet minskade. Totalt sett blev spelupplevelsen i stort sätt oförändrad.

Författarens tack

Jag vill tacka min handledare Kristian som gjorde detta arbete möjligt från början. Jag vill också tacka min fru för allt stöd jag fått från henne. Ett stort tack till Emil som utförde många speltester och Linus som snabbt ryckte in när det behövdes ny grafik. Tack till alla speltestare och alla andra som på ett eller annat sätt hjälpt mig med detta arbete.

Contents

List of Figures

List of Tables

1 Introduktion

Vid utveckling av spel till smartphones så behövs vanligtvis en implementation för varje operativsystem som ska stödjas. Detta eftersom Android vanligtvis kräver programmering i Java och IOS kräver programmering i Objective-C. För att undvika att behöva skriva om sin implementation flera gånger i olika språk, så finns det flera ramverk till olika programspråk som kan exportera ett projekt till flera olika mobila plattformar. *Kivy* är ett sådant ramverk till programspråket *Python*.

Spobot är ett tvådimensionellt onlinespel i spelkategorin action-strategi. Spobot är i skrivande stund i alphafasen och fungerar till *Android* och *IOS*. I Spobot spelar två spelare mot varandra, spelarna har varsin sida av skärmen där de styr en varsin robot. Robotarna kan skjuta projektiler mot varandra. Spelarna kan i spelet också köpa och placera ut olika sorts kanontorn, dessa kanontorn skjuter olika sorts projektiler mot motspelarens sida. Kanontornen kan förstöras av både spelarnas och andra kanontorns projektiler. Spelarna får i spelet pengar med tiden. För att vinna krävs det att kunna strategiskt välja vilka kanontorn som ska köpas vid vilket tillfälle och sätta ut dem på strategiska ställen. Det krävs samtidigt bra reflexer av spelarna för att lyckas undvika motståndarnas projektiler. Målet är att i spelet förstöra sin motståndare. I figur 1.1 visas en skärmdump från Spobot.

1.1 Motivering

En viktig aspekt vid spelutveckling är vilken spelupplevelse spelare upplever. Om de upplever en hög spelupplevelse av ett spel så kommer spelet mer sannolikt att lyckas. För att kunna erfara en hög spelupplevelse är det också viktigt att spelet har en bra prestanda.

I denna studie så ska det undersökas hur man kan höja spelupplevelsen av ett tvådimensionellt onlinespel i spelkategorin action-strategi. Detta ska göras genom att studera hur spelupplevelsen kan ökas för Spobot, som är ett spel i den kategorin. Under utvecklingen kommer det även undersökas vilka förändringar som kan göras för att öka spelets prestanda.

Figur 1.1: Skärmdump från Spobot

1.2 Syfte

Huvudsyftet med detta arbete är att undersöka hur Spobot ska omimplementeras i Kivy för att uppnå högre spelupplevelse än den redan existerande implementationen. Ett delmål av detta är att undersöka vilka åtgärder som kan genomföras vid utvecklingen av Spobot för att öka prestandan av spelet. Som en del av detta ska en kollisionsdetektionsalgoritm som ska fungera till liknande spel implementeras.

1.3 Frågeställning

1. Hur kan ett tvådimensionellt onlinespel i kategorin action-strategi implementeras till både Android och IOS med hjälp av Kivy så att det får högre spelupplevelse?
2. Hur kan ett sådant spel implementeras i Kivy för att uppnå så hög bildhastighet som möjligt?
3. Hur kan en generell pixelperfekt kollisionsdetektionsalgoritm för sådana spel implementeras som är så snabb som möjligt?

1.4 Avgränsningar

I detta avsnitt beskrivs de avgränsningar som görs i projektet.

1.4.1 Spelutveckling

Eftersom Spobot är i alphafasen så finns det mycket som behöver göras innan Spobot är färdigt. Detta projekt avgränsas till att endast om implementera de funktioner som redan finns i nuvarande implementation samt att omdesigna de funktioner som testpersonerna tycker ska göras om.

1.4.2 Spelplattformar

Även om Kivy kan exportera applikationer till både Android och IOS så kommer spelet av praktiska skäl endast att testas på Androidtelefoner.

1.4.3 Spelprestanda

Prestandan av ett spel kan mätas med flera olika mått, i detta arbete kommer endast bildhastigheten att användas som ett mått på spelets prestanda. För kollisionsdetektionsalgoritmen kommer både bildhastighet samt antal millisekunder som algoritmen kör att användas som prestandamått.

Av praktiska skäl kommer spelet endast prestandatestas på en HTC One M9 som har följande specifikationer:

Tabell 1.1: Specifikationer av HTC One M9 som används för prestandatester

Processorhastighet [Ghz]	2
Processorkärnor	8
RAM-Minne [GB]	3
Upplösning [px]	1920x1080
Tillverkningsår	2015

1.4.4 Kollisionsdetektionsalgoritm

Algoritmen begränsas till att endast hantera speglingar av objekt och inte rotationer.

2 Bakgrund

I detta kapitel beskrivs bakgrunden till varför projektet ska genomföras.

2.1 Spelbakgrund

Spobot är ett spel som i skrivande stund är i alpha-fasen och implementerat i spelmotorn *GameMaker*. Spelet är skapat av Robin Andersson (Programmerare), Linus Hägg (Grafik) och Emil Kammarbo (Ljud). Spelidén kom fram när skaparna bestämde sig för att de tillsammans ville utveckla ett spel. Skaparna hade då en braninstorming-session med olika spelidéer och idén om Spobot kom på det viset fram. Spobot började sedan att utvecklas i GameMaker då alla bakom spelet var insatta i det sedan tidigare. Målet med spelet är att skaparna ska starta ett företag och lägga ut spelet på *Google Play* och *App Store*.

2.2 Projektbakgrund

Detta projekt grundar sig i att personerna bakom spelet efter ett tag ville gå ifrån GameMaker. Detta för att både göra det enklare att vidareutveckla spelet och av ekonomiska skäl. Eftersom GameMaker inte har så mycket felmeddelanden medför detta att en bugg blir väldigt svår att upptäcka då den bara skapar odefinierat beteende och därmed blir det svårt att vidareutveckla spelet. GameMaker är ganska dyrt då det kostar cirka $400 per utvecklare.

Istället vill skaparna använda något som är gratis att använda. Eftersom programmeraren i projektet är en van Python-programmerare så valdes ramverket Kivy till projektet. Med hjälp av Kivy kan man programmera Android- och IOS-applikationer i Python. Eftersom spelet då behöver göras om från grunden så vill skaparna passa på att göra spelupplevelsetester på den nuvarande implementationen. Detta för att se om någon funktionalitet behöver göras om från grunden. Det skulle då vara väldigt passande att göra dessa förändringar när spelet ändå skrivs om från grunden.

3 Teori

I detta kapitel beskrivs den teori som är nödvändig för att förstå resten av detta arbete. Kapitlet tar upp de tekniker i Python och Kivy som använts i projektet på ett sådant sätt att en person som är insatt i programmering, men inte i Python eller Kivy ska kunna förstå.

Kapitlet beskriver också vad spelupplevelse är. Olika tekniker om hur man kan höja spelupplevelsen på ett spel samt hur man kan mäta den beskrivs också.

3.1 Python

Python är ett interpreterande, dynamiskt typat språk. [1] Python finns för närvarande i två versioner, *Python 2* och *Python 3*. Python 2 får numera endast säkerhetsuppdateringar [2] medan Python 3 förbättras hela tiden. [3]

3.1.1 Objektorientering

Python har stöd objektorientering med multipelt arv. En klass kan definieras enligt figur 3.1.

```
class Klass(Arv1, Arv2):
    egenskap2 = 42
    def __init__(self, egenskap1, egenskap2):
        self.egenskap1 = egenskap1
        self.egenskap2 = egenskap2

    def metod(self, argument1, argument2):
        self.argument1_2 = argument1+argument2
```

Figur 3.1: Python-kod för en klass. Inspirerad av olika kodexempel från Pythons dokumentation. [4]

Den första raden skapar en klass med namnet *Klass* som ärver från klasserna *Arv1* och *Arv2*. Den andra raden säger att *Klass* har en delad egenskap som heter *egenskap2*, när denna egendom ändras för en instans av *Klass* som kommer den även att ändras till samma värde för alla andra instanser av *Klass*. Den tredje raden definierar konstruktorn och den sjunde raden definierar en metod med namnet *metod*. [4]

Python har inget stöd för privata variabler, istället används som konvention ett understreck i början av variabelnamn som ska agera som en privat variabel. Ett användningsområde för privata variabler är dock att undvika namnkonflikter, detta kan göras genom att ha två understreck i början av ett variabelnamn och max ett understreck i slutet av det variabelnamnet. Variabelnamnet kommer då vid körning att bytas ut mot *_klassnamn_variabelnamn*. [4]

3.1.2 Moduler

I Python finns det olika sätt att importera moduler på. Ett sätt är att skriva *import modulnamn*, modulens innehåll kan då nås genom att skriva *modulnamn.innehåll*. Ett annat sätt är att först skriva *from modulnamn import innehåll* då kan *innehåll* användas direkt utan något prefix. [5]

3.1.3 Time modulen

Time är en modul i Python som kan användas för olika typer av tidsmätning. Metoden time.time() kan exempelvis användas för att ta reda på hur lång tid en viss koddel körs. Detta görs genom att sätta $t1 = time.time()$ innan koddelen som ska testas och vid slutet sätta $t2 = time.time() - t1$. $t2$ innehåller då exekveringstiden i sekunder. [6]

3.1.4 Dictionary

I Python finns en datatyp som kallas dictionary. I en dictionary kan man lagra godtyckligt många nyckel-värde par. Både nycklarna och värdena kan vara av godtycklig datatyp. En dictionary kan se ut på följande sätt: *dict = {'nyckel 1': 'värde 1', 'nyckel 2': 'värde 2'}*. Genom att då skriva *dict['nyckel 1']* så returneras strängen *värde 1*. [7]

3.2 Kivy

Kivy är ett ramverk till Python som gör det möjligt att skapa multiplattforms applikationer i Python. Det har stöd för många olika multitouch enheter.[8]

3.2.1 Widget

Kivy har en *Widget*-klass, alla klasser som ärver från denna klass får tillgång till många olika metoder. En av dessa är *collide_point()* som kontrollerar om en punkt finns i den nuvarande widgeten. Genom att ärva från *Widget* så kan även metoderna *on_touch_down()*, *on_touch_up()* och *on_touch_move()* definieras för att definiera vad som ska hända med nuvarande widget när olika touchevents händer. [9]

3.2.2 Design

För att separera logik från design så har Kivy ett eget desingspråk som heter *KV Language*. KV Language gör det möjligt att deklarativt skapa ett widget-träd. [10] Ett widget-träd kan deklareras enligt figur 3.2.

```
<MyWidget>:
  label_widget: label_widget
  Label:
    id: label_widget
    text: 'label'
  Button:
    text: 'button'
    on_press: root.remove_widget(label_widget)
```

Figur 3.2: Widget-träd i Kivy. Inspirerad av olika kodexempel från Kivys dokumentation. [11]

Den första raden definierar namnet på Widgeten. Den andra raden är en referens som säger att när vi skriver label_widget så vill vi ha det objekt som har id:t label_widget. Den tredje till femte raden säger att i *MyWidget* ska det finnas en *Label* med id label_widget och texten

label. Sjätte till åttonde raden säger att *MyWidget* också ska innehålla en knapp med texten *button* och när knappen trycks så ska funktionen remove_widget() köras med argumentet label_widget. *root* är ett nyckelord i Kivy som refererar till applikationens rotwidget. *remove_widget()* är en funktion som finns definierad i alla widgets, den tar bort barnet i nuvarande widget med det id som förses med argumentet.

3.2.3 Atlas

Atlasmodulen i Kivy sätter ihop flera texturer till en stor bild. Detta gör så att en bild kan laddas istället för flera vilket bland annat gör att spelet går snabbare att ladda. [12]

3.2.4 Clock

I Kivy finns en modul som heter *Clock*. Denna modul innehåller många olika tidsrelaterade metoder. De metoder som kommer användas i detta arbete är *schedule_interval*, *schedule_once*, *unschedule* och *get_rfps*. [13]

Schemaläggning

Metoderna schedule_interval och schedule_once kan användas för att schemalägga en funktion eller metod. Båda metoderna tar en funktion och en tid i sekunder som argument, funktionen kommer sedan köras efter den tid som angavs. Skillnaden mellan de två metoderna är att schedule_interval fortsätter anropa funktionen med den angivna tiden som intervall, medan schedule_once endast kommer anropa funktionen en gång.

När någon av ovanstående metoder anropas så returneras ett schema som kan sparas i en variabel. För att stoppa ett schema som skapats med hjälp av schedule_interval så kan schemat skickas in i metoden unschedule. [13]

Bildhastighetsmätning

Metoden get_rfps används för mätning av bildhastighet. Metoden returnerar en räknare som visar hur många bilder som visats den senaste sekunden. [13]

3.3 Spelupplevelse

I en artikel av E.H. Calvillo-Gámez et al. [14] så definieras huvudelementen av spelupplevelse. Detta görs genom att studera spelrecensioner och där identifiera återkommande element. Huvudelementen av spelupplevelsen innefattar både själva spelet samt dess interaktion med användaren.

Följande element och delelement identifierades för själva spelet:

- *Game-play* - Definierar vad spelet handlar om
  - *Regler* - De fysiska regler som alla objekt i spelet följer samt vad som går och inte går att göra i spelet
  - *Scenario* - Det scenario som spelaren sätts i
- *Miljö* - Hur spelet presenteras för spelaren
  - *Grafik* - Grafiken i spelet
  - *Ljud* - Ljudet i spelet

För interaktionen mellan spelet och spelaren identifierades följande element och delelement:

- *Kontroll* - Hur bra spelaren kan kontrollera spelet
  - *Mål* - Huvudsyftet med spelet
  - *Små handlingar* - Handlingar som kan göras på spelets objekt, exempelvis förflytting av position
  - *Kontroller* - Hur spelaren utför de små handlingarna, exempelvis genom att trycka på en tangent
  - *Minne* - Spelarens förmåga att komma ihåg mappningen från kontroller till de små handlingarna
  - *Något att göra* - Det ska alltid gå att göra något i spelet
  - *Synvinkel* - Hur spelaren ser miljön i spelet
- *Ägarskap* - Hur ansvarig spelaren känner sig för de handlingar som görs i spelet
  - *Stora handlingar* - De handlingar spelaren gör för att klara huvudmålet med spelet, detta görs genom att utföra flera små handlingar
  - *Personliga mål* - Spelaren har möjlighet att sätta egna mål och använda stora handlingar för att nå dem.
  - *Belöningar* - Spelaren får belöningar för att klara av olika mål
  - *Du men inte du* - Spelaren kan göra saker som denne inte skulle göra i verkligheten
- *Facilitatorer* -
  - *Tid* - Hur mycket tid spelaren är villig att spendera på spelet
  - *Tidigare erfarenheter* - Tidigare erfarenheter som spelaren fått från liknande eller helt andra spel
  - *Estetiska värden* - Hur estetiskt attraktivt spelaren tycker att spelet är

Enligt författarna är alla dessa element viktiga för spelupplevelsen, om något av dem skulle plockas bort från ett spel så skulle det påverka spelupplevelsen negativt. De säger också att ägarskap är det som slutligen leder till nöje, och ägarskap fås när spelaren har kontroll över spelet. Om spelaren har låg kontroll över spelet så behöver facilitatorerna vara bra för att spelaren ska kunna känna ägarskap.

I en annan studie av K. Poels et al. [15] så används fyra fokusgrupper samt fem spelforskare för att ta fram dimensionerna i spelupplevelse. Detta gjordes genom att först hålla diskussioner i fokusgrupperna. Dessa diskussioner var väldigt öppna men det fanns tre kärnfrågor som diskuterades:

- Vid vilka tillfällen börjar du vanligt vis spela?
- Vad upplever/känner du när du spelar?
- Vad upplever du eller hur känner du dig efter du har spelat?

Efter varje diskussionspass så fick deltagarna skriva på ett papper vilka upplevelser som de tyckte var relevanta för spel generellt. De mest relevanta skrevs i mitten på papperet och de mindre relevanta skrevs nära papperets kanter. När alla diskussionspass var avklarade så hölls ett expertmöte. På detta möte så fick de fem spelforskarna först skriva ner på post-it lappar vad de tyckte var viktiga dimensioner i spelupplevelsen. Sedan lades det även till post-it lappar från resultaten av fokusgruppernas diskussioner. Interaktivt och iterativt organiserades post-it lapparna efter relevans och likheter. Resultatet blev följande:

Dimension	Upplevelser i spelet	Upplevelser efter spelet
Nöje	Roligt, underhållande, ger njutning, avslappnande	Stimulerande, tillfredställande, avslappnande
Flöde	Konsentration, försjunkenhet, avskiljande	ger jetlag, tappar tidsuppfattning, överlåtelse
Imaginativ försjunkenhet	Absorberad av spelberättlesen, empati, karaktäridentifiering	Komma tillbaka till verkligheten
Sinnesförsjunkenhet	Närvaro	Komma tillbaka till verkligheten
Ovisshet	Utmaning, spänning, press, hopp, ångest, rysning	frigörelse, lättnad, utmattad, eufori
Kompetens	Stolthet, eufori, prestation	Stolthet, eufori, prestation, tillfredställande
Negativ effekt	Frustration, besvikelse, irritation, ilska	ånger, skuld, besvikelse, ilska, hämnd
Kontroll	Autonomi, makt, frihet	Makt, status
Social närvaro	Nöje med andra, uppkopplad med andra, empati, samarbete	Lagprestation, knyta kontakter

I en studie av Laura Ermi och Frans Mäyrä [16] så mäts spelupplevelsen genom att flera personer får testa ett spel och sedan svara på en enkät. Enkäten hade 30 påståenden som man fick välja från ett till fem hur mycket man höll med om påståendet. Slutsatser om spelupplevelsen drogs sedan utifrån svaren på denna enkät.

I en fallstudie av Pietro Guardini och Paolo Maninetti [17] så undersöks hur spelupplevelsen av ett rallyspel kan ökas. Detta görs genom att under utvecklingsfasen låta olika personer testa spelet och svara på frågor om hur de upplevde olika funktioner i spelet. Detta kombinerades med data om hur testpersonerna använde spelet i olika situationer som automatiskt samlades upp medans testpersonerna spelade spelet. Utifrån testpersonernas svar samt vanliga fel som spelarna gjorde så gjordes de testade funktionerna om, vilket slutligen ledde till högre spelupplevelse.

Enligt en studie av John P. Davis, Keith Steury, och Randy Pagulayan [18] så brukar en fokusgrupp på sex till tolv personer användas vid utveckling av spel. Denna fokusgrupp ska vara insatta i en viss spelgenre. Fokusgruppen ska diskutera kring vilka funktioner i spelet som är viktiga och vad de tycker om olika designkoncept.

3.3.1 GUESS

GUESS är en förkortning av *Game User Experience Satisfaction Scale*. GUESS innehåller 55 påståenden där användare får kryssa i från 1-7 hur mycket de håller med påståendet för det spel de testat. Alla påståenden är indelade i nio olika grupper: användarbarhet/spelbarhet, berättelser, spelupptagenhet, nöje, kreativitetsfrihet, ljud estetik, personlig tillfredsställelse, social uppkoppling och visuell estetik. Frågorna i social uppkoppling och berättelser går att ta bort. Detta kan behövas i vissa fall då det finns spel som inte har social uppkoppling eller berättelser. [19] Alla påståenden i GUESS finns angivna i appendix A.

Poängräkning

Rekommendationen för poängräkning är att ett medelvärde för varje grupp av påståenden som togs med beräknas. Detta gör att det går att se om vissa områden i spelet är sämre än andra. Dessa medelvärden kan sedan adderas för att få ett värde på den totala spelupplevelsen. [19]

Poängbetydelse

Eftersom GUESS är ett nytt mått på spelupplevelse så finns det ännu ingen siffra på vad som är gränsen för en hög spelupplevelse. Det som istället går att göra är att exempelvis använda GUESS på två olika spel inom samma genre, då kan slutsatser dras om vilket spel som har högre spelupplevelse än det andra. [19]

3.3.2 GEQ

GEQ är en förkortning av *Game Experience Questionare*. GEQ innehåller påståenden indelade i fyra olika kategorier: kärnmodul, i spelet, social närvaro modul och efter spelet modul. För varje påstående så får användaren ange hur mycket denne håller med om påståendet från 0-4, där 0 betyder att användaren inte alls håller med och 4 betyder att användaren håller med extremt mycket. Kärnmodulen innehåller 33 påståenden för att mäta spelupplevelsen på följande sju områden: immersion, flöde, kompetens, positiv och negativ påverkan, spänning samt utmaning. I spelet kategorin innehåller 14 frågor som mäter spelupplevelsen i samma områden som kärnmodulen. Påståendena i denna kategori ska svaras på flera gånger under en spelsession. Social närvaro modulen innehåller 17 frågor som mäter psykologiska och beteende aspekter av spelarens inblandning med andra sociala enheter. Denna modul innehåller tre områden: Empati, Negativa känslor och beteende. Efter spelet modulen innehåller 17 frågor som mäter hur spelaren känner sig efter denne har slutat spela. Påståendena i denna kategori är indelade i fyra områden: positiva erfarenheter, negativa erfarenheter, trötthet och tillbaka till verkligheten. [20]

Poängräkning

För varje kategori så räknas medelvärdet för varje område ut. All denna data är ett mått på spelupplevelsen. [20]

Poängbetydelse

Det finns ingen angivelse för vad som räknas som bra eller dålig spelupplevelse, i stället kan GEQ användas för att jämföra spelupplevelsen på olika spel. [20]

3.4 Fokusgrupp

Enligt en bok av R. Krueger och M.A Casey [21] så definieras fokusgrupper som: ”Noggrant planerade serier av diskussioner som är designade för att erhålla uppfattningar av ett område av intresse i en tillåtande ickehotande miljö.” Fokusgrupper kan enligt författarna användas för att få data om hur personer känner sig och vilken attityd de har till de diskuterade ämnena. De kan även hjälpa till att komma på nya idéer. Med fokusgrupper kan man få både individuella och interaktiva åsikter, de kan användas till både utveckling av nya program och till att utvärdera existerande. En fokusgrupp ska enligt författarna helst vara mellan sex till tio personer. En diskussionssession ska innehålla fyra till tolv ämnen som ska diskuteras. En session ska inte ta längre tid än en och en halv timme. Medlemmarna av fokusgruppen ska vara utvalda av en specifik anledning.

3.5 Prototypdriven utveckling

I en bok av M. Arvola [22] så beskrivs det att prototyper kan användas vid produktutveckling. En prototyp kan enligt författaren vara både på hela produkten eller på delar av produkten. Anledningen till att prototyper med fördel kan användas vid produktutveckling är att det ofta är svårt att förutse hur olika designval kommer påverka produkten. För att då slippa upptäcka vid slutet av tillverkningen av en produkt att en stor del av den inte blev bra så kan prototyper

av produktdelarna istället skapas för att snabbare få återkoppling kring hur bra de delarna av projektet var och om något behöver designas om. Författaren skriver att i början av ett projekt så bör skisser på konceptförslag tas fram i samarbete mellan designer, systemutvecklare och användare. Dessa skisser ska vara enkla och inte så genomarbetade så att det ska vara enkelt att förkasta dem. De skisser som inte blir förkastade kan sedan göras till prototyper.

I en studie av J. Manker och M. Arvola [23] så undersöktes hur 27 speldesigners använde sig av prototyparbete. Det konstaterades då att prototyper främst användes för att simulera och testa spelupplevelsen samt att testa mindre delar och specifika funktioner en i taget innan de sattes ihop.

I boken av M. Arvola [22] så skriver författaren att evolutionära prototyper är en bärande idé i agil programutveckling. Tanken med evolutionära prototyper är att de kan utvecklas i flera iterationer och byggas på efter återkoppling från användare.

3.6 Prestanda

I en bok av I. Molyneaux [24] så har en mjukvara bra prestanda om användaren kan utföra en uppgift utan att uppfatta en fördröjning eller irritation mellan användarens handlingar.

3.6.1 Lasttest

Enligt en bok av B. Wescott [25] så är lasttest en typ av prestandatest som oftast användas för system med många användare, som exempelvis webb-applikationer. Då är lasten antalet användare som använder webb-applikationen. Men lasttest kan även användas av andra system, exempelvis kan en ordbehandlare tvingas läsa ett stort dokument, då är lasten istället hur stort dokumentet är.

I en artikel av R. Khan och M. Amjad [26] så gjordes ett lasttest av en webb-applikation. Testet utfördes genom att mäta responstid, processoranvändning och minnesanvändning under tre timmar av vanlig användning.

3.6.2 Spelprestanda

Enligt en bok av R. Avisekhar [27] så är målet med prestandaoptimering i spel att öka bildhastigheten. Författaren skriver att en hög bildhastighet medför att spelet blir mjukare och förbättrar därmed spelupplevelsen. För ett tvådimensionellt mobilspel menar författaren att 60 bilder per sekund kan räknas som bra prestanda.

I en rapport av M. Claypool och K. Claypool [28] så undersöks bland annat hur bra spelarna presterar i förstapersonsskjutare-spel beroende på vilken bildhastighet som spelet har. Testet gjordes genom att testpersoner fick spela Quake III med en klient som efter 30 sekunder stängde av spelet och bytte bildhastighet. Klienten mätte vilken poäng spelarna fick efter varje 30 sekundersspel. Det visade sig att det är en liten skillnad i hur bra spelarna presterar när bildhastigheten är 30 bilder per sekund jämfört med 60 bilder per sekund. När spelarna spelade med 30 bilder per sekund fick de en medelpoäng på 3,3. När de istället spelade på 60 bilder per sekund fick de en medelpoäng på 3,6. Det blev alltså en skillnad på 0,3 poäng. Detta kan jämföras med när spelarna spelade med 15 bilder per sekund, då fick de en medelpoäng på 2,6 poäng. Det är alltså en skillnad på 0,7 poäng, det vill säga mer än dubbelt så stor skillnad som mellan 30 och 60 bilder per sekund.

I en annan rapport av M. Claypool och K. Claypool [29] så undersöks hur bildhastigheten påverkar spelarnas prestanda i tre olika varianter av ett spel de själva implementerat. Skillnaden

i de tre olika varianterna är vilken vy spelaren spelar i. De vyer som testas är första-persons vy, tredje-persons vy i tre dimensioner samt tredje-persons vy i två dimensioner. För dessa varianter så testades spelet för 7,5, 15 och 30 bildrutor per sekund. De mätvärden som testades var spelbarhet gällande skjutning, spelbarhet gällande navigering, poäng vid skjutning och poäng vid navigering. Studien visade att generellt så förbättrades spelarnas prestanda mycket om de spelade i 15 bildrutor per sekund istället för 7,5 bildrutor per sekund, medans förbättringen var relativt liten om spelarna istället spelade med 30 bildrutor per sekund istället för 15 bildrutor per sekund. Det var dock ingen större skillnad på hur viktig bildhastigheten var för de olika vyerna.

Förbättring av spelprestanda

I en bok av R. Avisekhar [27] så skriver författaren att kollisionsdetektion är en väldigt kostsam process. I en naiv implementation av kollisionshantering måste alla objekt testas mot alla andra om de kolliderar så fort ett spelobjekt har flyttat på sig. Författaren menar därför att snabba upp kollisionsdetekteringen kan ha en stor effekt på spelprestandan.

En annan förbättring som författaren nämner är att istället för att ha många olika bildfiler för varje bild så kan ett färre antal bildblad användas. Dessa bildblad innehåller ett flertal bilder. Istället för att ett draw call skickas för varje bild som ändras behövs då bara ett draw call för varje bildblad.

Enligt ett blogginlägg på en teknikblogg av M.K Lewandowski och J. Lewandowski [30] så kan spelprestandan ökas genom att använda sig av en objektpool. Istället för att skapa och ta bort objekt dynamiskt så uppskattas och skapas ett maximalt antal av de objekt som finns i början av det spelscenario som ska spelas. Författarna menar att skapande och borttagning av objekt är kostsamt för prestandan och därför skulle användandet av en objektpool öka prestandan.

3.7 Tekniker och algoritmer

Olika tekniker och algoritmer som kan användas för att förbättra prestandan i ett spel beskrivs i detta avsnitt.

3.7.1 Kollisionsdetektering

Enligt en online-artikel av N. Bobic [31] så finns det två huvudproblem med kollisionsdetektering, att behöva testa kollisioner mellan så få objekt som möjligt samt att utföra ett kollisionstest mellan två objekt så snabbt och precist som möjligt.

Minimering av kollisionstester

N. Bobic skriver i sin online-artikel [31] att i en naiv lösning så testas varje par av objekt om de kolliderar. Men författaren ger som exempel att två väggar aldrig kommer kunna krocka med varandra. Vidare skriver författaren att i många spel kommer heller inte kollisionsdetektering mellan två skott att behöva testas. Författaren föreslår därför att spelets objekt kan delas in i olika mängder där objekt i en viss mängd endast behöver testas med objekt i någon eller några andra mängder.

I ett blogginlägg på en teknikblogg av S. Lambert [32] så skriver författaren att kollisionsdetektionen kan snabbas upp genom att ge alla spelobjekt en metod *isCollidableWith* som har ett spelobjekt som parameter. Tanken med denna metod är att algoritmen inte ska behöva leta efter kollisioner mellan objekt som ändå inte kan kollidera med varandra. Denna metod är sedan det första som anropas i kollisionstestsloopen.

I ett blogginlägg på en teknikblogg av K. Schouviller [33] så skriver författaren att ett *quadtree* ofta används för att minimera antalet kollisionstester. Detta innebär att spelplanen fungerar som en rotnod, sedan delas den in i fyra lika stora rektanglar, detta blir rotnodens barn. Denna procedur fortsätter rekursivt till en nivå som bestäms beroende på hur stor spelplanen och spelobjekten är. Spelobjekten placeras sedan i denna trädstruktur genom att först titta om något objekt finns i rotnoden, i sådana fall ska kollision testas mellan de objekten och nuvarande objekt. Sedan ska de barn som nuvarande objekt kolliderar med kollas på samma sätt. Proceduren fortsätter rekursivt tills löven i trädet som det nuvarande objektet kolliderar med har kontrollerats. Sedan placeras nuvarande objekt in så djupt ner i trädet som är möjligt för att det endast ska existera i en ruta.

I en online artikel av N. Souto [34] så beskrivs att en algoritm som kallas *The Sort and Sweep Algorithm* kan användas för att minska antalet kollisionstester. Algoritmen går ut på att alla spelobjekts start och slutposition i en utvald axel läggs in i en lista. Listan sorteras sedan efter position. Sedan traverseras listan, för varje startposition som hittas så läggs det motsvarande intervallet in i en separat lista med aktiva intervall och för varje slutposition som hittas så tas det motsvarande intervallet bort från listan med aktiva intervall. Varje gång ett nytt intervall läggs till som aktivt så ska det motsvarande objektet kollisionstestas med alla motsvarande objekt till de intervall som redan finns i listan med aktiva intervall.

Kollisionstester

I online artikeln av N. Bobic [31] så beskrivs bland annat en teknik för kollisionstester som kallas *AABB* vilket står för *Axis-aligned bounding box*. Enligt författaren går tekniken ut på att varje spelobjekt representeras av en rektangel som är vinkelrät mot spelets koordinatsystem. Att testa om två objekt kolliderar kan då göras i konstant tid. Om spelobjektet inte är rektangulärt blir dock kollisionsdetektionen en grov uppskattning. Därför brukar denna teknik användas för att först se om två objekt möjligen kolliderar, om de gör det så kan en mer precis kollisionsdetekteringsalgoritm användas för att se om de faktiskt kolliderade.

Ett alternativ till AABB som författaren ger är att istället använda sig av *OBB* som står för *Oriented bounding box*. En OBB fungerar på samma sätt som en AABB med skillnaden att istället för att rektangeln är vinkelrät till koordinatsystemet, så är den roterad så att den har så liten area som möjligt för att objektet den hör till fortfarande får plats i den. På grund av detta så blir en OBB oftast mycket mer precis än en AABB, problemet enligt författaren är istället att det krävs många komplexa beräkningar för att ta fram en OBB.

I en online artikel skriven av D.A de Oliveira Neto [35] så beskriver författaren bland annat en teknik för att uppnå pixelperfekt kollisionsdetektering mellan två objekt, det vill säga en algoritm som signalerar kollision om och endast om de två objekten har minst en icke-transparent pixel på samma koordinat. Tekniken går ut på att när en bild som ska användas för ett spelobjekt läses in så skapas en matris med lika många element i höjd och bredd som bilden har pixlar i respektive riktning. Matrisen fylls sedan med falskt om den motsvarande pixeln i bilden är transparent, annars blir värdet sant. När kollision mellan två objekt sedan testas så jämförs dessa matriser med en förskjutning som beror på avståndet mellan objekten.

I online artikeln av N. Souto [34] så skriver författaren att ett teorem som kallas *SAT* som står för *The Separating Axis Therorem* brukar användas för att se om två konvexa polygoner kolliderar med varandra. Teoremet säger att två konvexa polygoner inte kolliderar med varandra om och endast om det finns minst en axel där de ortogonala projektionerna av polygonerna inte kolliderar med varandra. Problemet med detta teorem är att det finns oändligt många axlar att testa. Men enligt författaren behöver endast de axlar som är parallela till någon linje i de konvexa polygonerna att testas. Att polygonerna måste vara konvexa är dock ett problem. För att kunna testa konkava polygoner nämner författaren två tekniker, det ena är

att använda sig av det konvexa höljet av polygonen. Problemet med det är att man förlorar precision. Alternativet är att dela in den konkava polygonen i konvexa delar. Problemet med denna strategi är istället att vissa polygoner kan innehålla flera hundra konvexa polygoner.

3.7.2 Sorteringsalgoritmer

Insättningssortering

Enligt en bok av T.H. Cormen et al. [36] så går insättningssortering ut på att man traverserar en lista en gång. Under traverseringen så kollar man om det nuvarande elementet är mindre än det förra elementet, om så är fallet så byter man plats på de elementen och fortsätter sedan kolla på elementet bakom och byter plats på dem och på det sättet kommer nuvarande element till sin rätta position. När listan är traverserad är den också sorterad. Detta betyder att om algoritmen får in en sorterad lista så behöver den bara titta på alla element en gång det vill säga algoritmen får en bästafalls-komplexitet på $O(n)$. Algoritmen har dock en värstafalls-komplexitet på $O(n^2)$.

Urvalssortering

Enligt boken av T.H Cormen et al. [36] så är urvalssortering när man börjar med att gå igenom listan för att hitta det minsta elementet och så placeras det på den första positionen i listan, sedan fortsätter man på samma sätt med den resterande listan. Algoritmen har en komplexitet på $O(n^2)$.

Quicksort

Enligt boken av T.H Cormen et al. [36] så går quicksort ut på att man börjar med att välja ett pivotelement, alla element i listan som är mindre än eller lika med pivotelementet läggs innan det medans de övriga läggs efter det. Elementen med och innan pivotelementet är nu en del av listan och de efter pivotelementet är en annan del. Fortsätt på samma sätt rekursivt med de två delarna av listan. Enligt författarna är quicksort den sorteringsalgoritm som oftast brukar användas. Detta trots att värstafalls-komplexiteten är $O(n^2)$, dock är den förväntade komplexiteten $O(n \log(n))$.

Merge sort

I boken av T.H Cormen et al. [36] så skriver författarna att i merge sort så sorteras en lista av längd n genom att först dela in listan i n dellistor med ett element i varje dellista. Dellistorna slås sedan ihop två och två, detta sker genom att jämföra det första elementet i båda listorna och ta det minsta av dem. Denna procedur fortsätter rekursivt tills det endast finns en lista av längd n kvar. Den listan är då sorterad. Komplexiteten är $O(n \log(n))$.

Timsort

I ett blogginlägg på en teknikblogg av D. MacIver [37] så skriver författaren att Python använder sig av en sorteringsalgoritm som heter Timsort. Timsort är i grunden en optimerad merge sort. Timsort använder sig även delvis av insättningssortering. Värstafalls komplexiteten är $O(n \log(n))$ och bästafalls komplexiteten är $O(n)$.

4 Metod

Arbetet kommer ske i en förstudiefas, tre iterationer där varje iteration har en implementationsfas och en utvärderingsfas samt en utvärderingsfas för hela projektet.

4.1 Spelupplevelse

Spelupplevelsen kommer mätas genom att testpersoner får testa spelet och sedan svara på en online-enkät med GUESS frågorna. Detta kommer göras två gånger, en gång på det redan existerande spelet i början av projektet och en gång på den nya implementationen i slutet av projektet. Eftersom spelet inte har någon berättelse och ännu ingen social uppkoppling så kommer frågorna i de kategorierna vara borttagna i enkäten.

4.1.1 Testgrupper

Inför det första spelupplevelsetestet så ska två grupper på sex personer vardera väljas ut. Den ena gruppen kan liknas vid en fokusgrupp då personerna i den kommer vara insatta i spelkategorin action-strategi. Denna grupp kommer också vara med och både ge åsikter på de funktioner som finns i spelet samt ge förslag på förbättringar. Det som skiljer denna grupp från en fokusgrupp är att inga diskussionspass kommer att hållas, utan all data kommer samlas in genom online-enkäter. Därför är denna grupp enligt definition inte en fokusgrupp. Istället kommer förslagen från denna grupp användas för att ta fram och utvärdera prototyper. I detta arbete kommer denna grupp kallas för *prototypgruppen*. Den andra gruppen ska bestå av mer slumpmässigt utvalda personer. Denna grupp kommer kallas för *vardagsgrupp 1* och ska endast vara med och testa spelet samt svara på GUESS frågorna.

Inför det andra spelupplevelsetestet så ska ytterligare en vardagsgrupp att väljas ut. Denna grupp kommer kallas för *vardagsgrupp 2*. Alla tre testgrupper kommer testa den nya implementationen och svara på GUESS frågorna.

4.1.2 Speltester

Testpersonerna får själva välja en motståndare att spela mot. Huvudsaken är att motspelaren har samma förkunskaper om spelet som testpersonen. Spelarna får först instruktioner om att de ska läsa igenom en kort genomgång av spelet som finns i spelet. När de är klara med det kommer de få testa spela mot en enkel datorstyrd motståndare i max fem minuter för att kunna lära sig kontrollerna. Spelarna ska sedan spela i minst fem minuter mot varandra. Under denna tid ska testledaren inte ge några ytterligare instruktioner. Testledaren ska istället under tiden observera saker som testpersonerna har svårt att förstå med spelet och notera dem.

4.1.3 Förbättring av spelupplevelse

För att förbättra spelupplevelsen av spelet så ska prototypdriven utveckling användas. Efter det första spelupplevelsetestet så kommer testpersonerna i prototypgruppen få tillgång till en enkät där de ska föreslå förbättringar till spelet. Dessa förslag ska vara skisser till prototyper till någon funktion som testpersonen vill ska omdesignas. De förslag som fås kommer sedan diskuteras mellan utvecklarna av spelet. I mån av tid så ska prototyper för alla förslag som utvecklarna tyckte var genomförbara att implementeras. Utvecklarna kommer även ta fram skisser av prototyper för de observationer som gjordes under speltestningen. Eftersom det inte är säkert att tiden räcker till för att implementera prototyper för alla förbättringsförslag så ska de prioriteras. Hur prioriteringen går till beskrivs i avsnitt 4.1.4.

Alla prototyper ska sedan testas av prototypgruppen. Testpersonerna får sedan tillgång till ytterligare en online-enkät där de för varje prototyp får svara från -5 till 5 hur de tyckte att den påverkade spelupplevelsen. De får även ge förslag för varje prototyp om de tycker att de bör göras om eller kan förbättras på något sätt. Alla prototyper som får ett negativt medelvärde för hur testpersonerna tyckte att det påverkade spelupplevelsen och som inte fick några förbättringsförslag kommer slängas bort. Alla förbättringsförslag på prototyperna ska sedan diskuteras mellan utvecklarna och de som anses genomförbara ska sedan implementeras.

Slutligen kommer förbättringarna av prototyperna att utvärderas på samma sätt som tidigare. Om flera prototyper är olika sätt att lösa samma problem så kommer en fråga till prototypgruppen ställas, vilken av prototyperna de föredrar.

4.1.4 Prioritering av förbättringsförslag

Prioriteten p av ett förbättringsförslag ska bero på förslagets implementationstid t samt hur viktigt testpersonerna i prototypgruppen tycker att förbättringsförslaget är v. För att hinna med att implementera flera förbättringar så kommer förbättringar som uppskattas att ta mer än 50 timmar ej att genomföras. Tidsuppskattningen kommer att göras helt på intuition i hela timmar. För att få reda på hur viktiga testpersonerna i prototypgruppen tycker att förbättringsförslagen är så får de svara på enkäten med de förbättringsförslag som utvecklarna kom överens om. Denna enkät ska innehålla för varje förbättringsförslag en skala från minus fem till fem. Minus fem betyder att man tror att den påstådda förbättringen kommer påverka spelet starkt negativt och fem betyder att den kommer påverka spelet starkt positivt. För varje förbättringsförslag kommer medelvärdet att användas som värde för hur viktigt förslaget är. Ett förslag som får ett medelvärde på noll eller mindre kommer ej att genomföras. Utifrån detta kommer vi fram till ekvation 4.1 som kommer användas för beräkning av prioritet.

$$p = v\frac{51 - t}{10} \tag{4.1}$$

4.1.5 Testmiljö

För att kunna testa ett online-spel så krävs någon form av server som spelarna kan ansluta till. För att tester ska kunna ske så enkelt som möjligt så ska det sättas upp en server som väntar på att två klienter ansluter till den, när detta skett så hamnar dessa spelare direkt i ett spel mot varandra. Adressen till denna server hårdkodas i spelet så att spelaren inte behöver göra något annat än att trycka på en knapp för att ansluta till den.

4.2 Prestanda

Vid slutet av varje iteration ska spelets bildhastighet att undersökas. Spel-loopen kommer anropas 60 gånger per sekund vilket betyder att bildhastigheten för spelet inte kommer kunna överstiga 60 bilder per sekund, däremot så kan det hända att en spel-loop inte hinner köras

klart innan nästa påbörjas, vilket kommer leda till en mindre uppmätt bildhastighet. Varje testtillfälle innehåller två test av bildhastigheten. Först görs ett test på spelet som det är, sedan implementeras några av de tekniker som beskrivs nedan i detta avsnitt. Slutligen testas bildhastigheten igen för att kontrollera att åtgärderna faktiskt ökade prestandan.

För att testen ska vara jämförbara gäller det att de utförs på samma sätt. Därför ska testen genomföras genom att en av karaktärerna placerar ut två av varje torn. Detta kan då sägas vara en förbestämd last, därav blir detta en typ av lasttest. Under tiden så ska bildhastigheten mätas. Detta ska göras med hjälp av *Clock.get_rfps()* metoden som finns i Kivy. I varje uppdatering av objekten i spelet så ska denna metod anropas. Varje gång ett nytt minsta värde av bildhastigheten uppmäts så ska det visas. När testet är slutfört så kommer medelvärdet av bildhastigheten att räknas ut och visas tillsammans med det minsta värdet av bildhastigheten.

4.2.1 Kollisionsdetektionsalgoritm

För kollisionsdetektionsalgoritmen så ska även exekveringstiden för algoritmen att mätas. Detta ska göras med hjälp av Pythons *time* bibliotek.

Konstruktionen av kollisionsalgoritmen ska börjas genom att implementera en detektion för AABB kollisioner. Detta ska kompletteras med den pixelperfekta kollisionsdetektionen som beskrevs i avsnitt 3.7.1. Denna algoritm ska sedan prestandatestas.

Sedan ska ytterligare några av de kollisiondetektionstekniker som beskrivs i avsnitt 3.7.1 att implementeras. Varje teknik som implementeras ska prestandatestas tillsammans med ovanstående tekniker.

Kollisionsmängder

Fyra olika kollisionsmängder kommer implementeras: *moving_collider*, *moving_collidee*, *still_collider* och *still_collidee*. En *collider* är ett objekt som ska kollisionstestas mot alla objekt medans en *collidee* endast ska testas mot objekt som är en collider. Ett objekt i mängden moving_collider kommer då behöva kollisionstestas mot alla objekt som finns i någon av de ovanstående mängderna. Ett objekt i mängden still_collider kommer endast behöva testas mot objekt i moving_collidee eftersom den inte kan kollidera med stillastående objekt (förutsatt att objekt inte kan skapas i varandra) och objekt i mängden moving_collider redan har testats mot alla objekt i mängden still_collider.

Quadtree

Ett quadtree där djupet går att ställa in kommer att implementeras. Djupen tre och fyra kommer testas i Spobot.

The Sort and Sweep Algorithm

The Sort and Sweep Algorithm kommer implementeras så det går att ställa in vilken axel som objektens start- och slut-position ska tas ifrån. Både x- och y-axeln kommer att testas i Spobot. För sortering av intervallistan i algoritmen så ska insättningssortering, quicksort och timsort testas.

4.2.2 Bildblad

Det ska undersökas hur prestandan av spelet påverkas genom att använda sig av bildblad. Bildbladen kommer implementeras med hjälp av Atlas-modulen som finns i Kivy.

4.2.3 Objektpoler

Hur prestandan av spelet påverkas genom att använda sig av objektpoler ska undersökas. Det ska finnas en objektpol för varje objekt som kan skapas dynamiskt.

4.3 Motivering till metod

I detta avsnitt motiveras varför ovanstående metod valdes.

4.3.1 Mätning av spelupplevelse

För att mäta spelupplevelsen beskrevs två olika skalor i teorikapitlet: GUESS och GEQ. GEQ innehåller totalt sett fler påståenden och har fler kategorier som är en del av spelupplevelsen. Problemet med GEQ i denna studie är att vi enkelt vill kunna säga om spelupplevelsen på spelet blev bättre eller inte. Eftersom GEQ har flera värden som tillsammans säger något om spelupplevelsen så blir detta svårt. I GUESS får vi också flera värden som säger något om spelupplevelsen, men vi får också ett värde som står för den totala spelupplevelsen av spelet. Dessutom är GUESS nyare och baserad på senare forskning. Av dessa anledningar valdes GUESS som mätverktyg av spelupplevelsen i detta arbete.

4.3.2 Förbättring av spelupplevelse

I teorikapitlet togs tre metoder upp för förbättring av spelupplevelse.

Den första metoden var att låta flera testpersoner testa spelet i en övervakad testmiljö. Med denna metod kan dåliga saker i spelet upptäckas som spelarna själva inte tänker på, exempelvis kan det upptäckas att spelarna inte alls använder en viss funktion i spelet. Nackdelen med denna metod är att det är väldigt tidskrävande att skapa den övervakade testmiljön.

Den andra metoden var att ha en fokusgrupp på sex till tolv personer som är väl insatta i spelets genre. En fördel med denna metod är att det är väldigt tidseffektivt då ingen speciell testmiljö behöver sättas upp. En annan fördel är att genom att ha en mer utvald grupp som testar och kommer med förslag till spelet så blir antagligen förslagen mer riktade till den målgrupp som kommer spela spelet. En nackdel med denna metod är att vissa saker som exempelvis låg användning av funktioner lätt kan missas. En annan nackdel är att fokusgruppen kan se vissa funktioner i spelet som självklara, vilket gör att det kan bli svårt för personer som inte är så insatta i spelgenren att förstå spelet.

Den tredje metoden var att använda sig av prototypdriven utveckling. En fördel med denna metod är att testpersonerna själva kan ge idéer utan att bli påverkade av andra och falla för grupptryck vilket kan hända i diskussionspass. Flera olika idéer kan sedan testas och vidareutvecklas allt eftersom. En nackdel är precis som med att använda sig av fokusgrupper så kan det vara svårt att få fram om vissa funktioner används väldigt lite eller om någon funktion är svår att förstå.

Eftersom detta arbete ska göras på 20 veckor och det inkluderar en implementation av ett helt spel så är tid en väldigt begränsande resurs. Att sätta upp en övervakad testmiljö kommer därför inte vara görbart. Eftersom prototypgruppen kommer bestå av personer från olika städer med olika arbetstider kommer det inte heller vara möjligt att hålla diskussionspass. Därför valdes prototypdriven utveckling som arbetsmetod för att öka spelupplevelsen. För att även kunna fånga upp saker som är svåra att förstå eller om någon funktion inte används så observeras även testningen av en testledare som noterar vad testpersonerna verkar ha svårt att förstå och funktioner som inte används.

Nackdelarna med denna metod är att ett förslag kan missförstås när det inte finns någon diskussion. Dessutom så skulle i en diskussion ett förslag kunna förbättras tills dess att alla i gruppen tycker att förslaget är riktigt bra. Fördelarna med metoden är istället att ingen kommer kunna falla för något grupptryck och bara hålla med om något för att denne inte vågar säga emot vad de andra tycker. Observationerna kommer också kunna bidra till att funktioner som kanske inte är så triviala upptäcks, i en diskussion så skulle testpersonerna kanske inte vilja medge att de hade svårt för vissa saker för att de inte vill ses som dåliga.

4.3.3 Implementation av kollisionsdetektionsalgoritm

Eftersom OBB är väldigt komplicerat och det finns många olika algoritmer att skapa dessa på så valdes istället AABB som ett första kollisionstest. AABB valdes också att användas som grund till kollisionsalgoritmen, det vill säga AABB kollisionstest kommer alltid att användas någon gång oavsett vilka tekniker som testas. Detta då AABB tester är ett snabbt och enkelt test som alltid går att göra.

Kollisionsmängder

Kollisionsmängderna skulle kunna se väldigt olika ut i olika typer av spel. Men de fyra mängderna valdes utifrån det som N. Bobic [31] skrev i sin online artikel, att det är onödigt att testa kollisioner mellan objekt som inte kan förflytta sig. Men eftersom författaren också skrev att det ofta finns objekt som kan förflytta sig men som inte ska kollisionstestas så räcker det inte med två mängder där den ena är förflyttningsbara objekt och den andra stillastående objekt. Därför lades det även till en separation av objekt som ska kollidera med allt och objekt som kan kollidera med något. Resultatet blir väldigt generellt där inga antaganden görs om det spel eller program som ska använda kollisionsalgoritmen.

Kollisionsmöjligheter

Det valdes att inte implementera kollisionsmöjligheter då det alltid går att åstadkomma samma sak med kollisionsmängder istället. Kollisionsmängder blir dock mer generellt och man slipper definiera för varje spelobjekt vilka den kan kollidera med. I Spobot kommer kollisionsmängderna som ska implementeras att bara ge de test som faktiskt behöver kollas. Därav skulle ett test för kollisionsmöjligheter för Spobot bli ett mindre generellt sätt att åstadkomma samma sak.

Quadtree

Anledningen till att djupen tre och fyra ska testas i Spobot är att Spobot har en upplösning som är 854x480 pixlar, på ett djup av fyra kommer då rektanglarna vara cirka 107x60. Eftersom de flesta spelobjekt är 50x50 så kommer sällan fler än fyra djup kunna användas.

The Sweep and Sort Algorithm

Insättningssortering ska testas som sorteringsmetod i denna algoritm eftersom insättningssortering snabbt terminerar om den får en sorterad lista och den snabbt sorterar en nästan sorterad lista. Detta är en bra egenskap då listan med intervall i algoritmen kommer vara sorterad om inget objekt flyttat på sig. Quicksort ska testas eftersom det är den vanligaste sorteringsalgoritmen och den är snabb i medelfallet. Timsort ska testas eftersom även den ska kunna sortera nästan sorterade listor väldigt snabbt och dessutom har en snabb värsta falls komplexitet. Merge sort väljs att inte testas då timsort fungerar på nästan samma sätt men bör vara bättre på nästan sorterade listor vilket ofta kommer vara fallet i denna lista.

SAT

Eftersom det är väldigt tidskrävande att få till en bra implementation av SAT som fungerar för konkava polygoner så kommer SAT inte implementeras i detta arbete.

4.3.4 Mätning av prestanda

I teorikapitlet beskrevs det att prestanda av mjukvara ofta mäts i responstid. Men för spel är inte responstid ett särskilt bra mått på prestanda. Detta eftersom det sker så mycket interaktion mellan användare och spel och då brukar det ofta förväntas av spelaren att denne inte upplever någon responstid alls. Det beskrevs också i teorikapitlet att prestandan för spel kan mätas i bildrutor per sekund. Eftersom dessa rapporter visade en koppling mellan bildhastighet och hur bra spelarna presterade i spelet så valdes bildhastighet som mått för spelets prestanda.

Anledningen till att två av varje torn kommer att placeras ut i lasttestet är att det är i överkant av hur många objekt som kommer kunna finnas på spelplanen i ett vanligt spel. Skulle en spelare lyckas få så många torn så skulle de väldigt snabbt döda motståndaren och vinna spelet. Skulle istället båda spelarna lyckas komma i närheten av hälften av så många torn var så kommer tornen ha sönder varandra snabbare än vad det går att bygga nya.

Kollisionsdetektionsalgoritmen ska prestandamätas med hjälp av både bildhastighet och antal millisekunder som algoritmen kör. Bildhastigheten ska mätas eftersom målet med alla prestanda optimeringar är att få bättre bildhastighet. Tiden som algoritmen kör ska mätas eftersom det ger ett direkt resultat för just algoritmen medan bildhastigheten kommer bero på många olika faktorer.

5 Resultat

5.1 Förstudie

Under förstudien gjordes GameMaker-implementationen redo för testning. Detta innebar bland annat att en testmiljö sattes upp, ikoner för de olika resurserna lades till samt små buggar som hade upptäckts fixades. Prototypgruppen och vardagsgrupp 1 valdes också ut. Sedan gjordes ett speltest med dessa testgrupper för att värdera spelupplevelsen för GameMaker-implementationen, resultatet för detta test finns i tabell 5.1.

Förstudien avslutades genom att prioritera förbättringsförslagen som skapades av prototypgruppen och från spelobservationer. Prioriteringarna finns i tabell 5.2.

5.2 Första iterationen

Under den första iterationen så återskapades Spobot till den grad att två spelare kan se varandra röra sig online och båda spelarna kan placera alla torn. Dessutom implementerades en kollisionsdetektionsalgoritm som går att använda till andra tvådimensionella spel som implementeras i Python.

Eftersom spelet efter den första iterationen ännu inte liknar ett spel särskilt mycket, exempelvis går det inte att ha sönder motspelarens torn, så gjordes inga prototyp- eller spelupplevelsetester efter denna iteration. Istället undersöktes hur bra olika tekniker för kollisionsdetektion fungerar i spelet. Hur denna algoritm skapades beskrivs i avsnitt 5.7.1.

Några prototyper började också utvecklas i den första iterationen. Hur de olika prototyperna utvecklades beskrivs i avsnitt 5.6

Tabell 5.1: Spelupplevelse av GameMaker-implementation

Kategori	Prototypgrupp	Vardagsgrupp 1	Sammanlagt
Användarbarhet/Spelbarhet	4,60	3,85	4,23
Spelupptagenhet	4,56	3,93	4,25
Nöje	5,80	5,11	5,46
Kreativitetsfrihet	4,57	4,00	4,29
Ljud estetik	5,55	4,29	4,92
Personlig tillfredsställelse	5,53	4,99	5,26
Visuell estetik	5,11	4,94	5,03
TOTALT	35,71	31,12	33,42

Tabell 5.2: Prioritet av genomförbara förbättringsförslag

Förslag	Betyg	Tidsestimering	Prioritet
Två tryck för placering av torn	3,2	6	14,4
Spelarna får välja vilken spelhalva de vill spela på	2,7	8	11,6
Karaktären går mot strax framför fingret istället för direkt på	2,2	1	10,4
Ta bort spelarens sköld från spelet	1,2	1	6
Gör så att egna torn inte kan skada varandra	1,2	8	5,2
När spelaren trycker på en tornknapp så placeras tornet ut framför karaktären	1	1	5
D-pad till styrning	2	20	4,2
Gör så att magier är gratis men har en nedkylningstid istället	0,7	6	3,2
Dra tornknappen åt det håll som tornet ska placeras åt relativt till karaktären	0,2	11	0,8
Ta bort två torn från spelet	-0,8	1	-
Gör så att torn kan placeras godtyckligt långt ifrån karaktären	-1	1	-

5.3 Andra iterationen

I den andra iterationen återskapades spelet helt. Ytterligare några prototyper implementerades. Iterationen avslutades genom att låta prototypgruppen testa de olika prototyperna för att sedan betygsätta dem. Resultatet finns i tabell 5.3. Påverkan av spelupplevelse i tabellen avser hur spelarna själva upplevde att spelupplevelsen förändrades från de tidigare versionerna av prototyperna. Datan samlades in med hjälp av de online-enkäter som finns i appendix B.3 och B.4.

Tabell 5.3: Prototypers påverkan av spelupplevelse enligt prototypgrupp

Prototyp	Version	Påverkan av spelupplevelse
Tvåtrycksplacering	-	2,2
Direktplacering	1	3,0
Styrkorsförflyttning	1	0,8
Magikostnad	2	3,8
Vådabeskjutning	2	2,2
Spelplanhalva	2	2,3
Fingerförflyttning	2	3,8

I slutet av den andra iterationen implementerades även bildblad, hur detta gjordes beskrivs i avsnitt 5.7.2.

5.4 Tredje iterationen

I den tredje iterationen så gjordes prototyperna styrkorsförflyttning och direktplacering om enligt förbättringsförslag från prototypgruppen, se avsnitt 5.6.7 och avsnitt 5.6.5. Vid slutet av iterationen fick prototypgruppen testa de nya versionerna av dessa prototyper. Resultatet av detta redovisas i tabell 5.4.

Testpersonerna fick även svara på vilken typ av tornplacering och vilken typ av karaktärsstyrning de föredrog. Samtliga föredrog direktplacering för tornplacering. Gällande karaktärsför-

Tabell 5.4: Prototypers påverkan av spelupplevelse enligt prototypgrupp

Prototyp	Version	Påverkan av spelupplevelse
Direktplacering	2	2
Styrkorsförflyttning	2	3,3

Tabell 5.5: Spelupplevelse av Kivy-implementation

Kategori	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp 2	Sammanlagt
Användarbarhet /Spelbarhet	5,32 (+0,72)	4,52 (+0,67)	4,40	4,75
Spelupptagenhet	4,23 (-0,33)	4,15 (+0,22)	3,22	3,87
Nöje	5,63 (-0,17)	4,5 (-0,61)	4,70	4.94
Kreativitetsfrihet	4,25 (-0,32)	4,29 (+0,29)	3,84	4.13
Ljud estetik	5,95 (+0,4)	5,08 (+0,79)	4,71	5,25
Personlig tillfredsställelse	5,14 (-0,39)	5,14 (+0,15)	4,75	5,01
Visuell estetik	5,39 (+0,28)	5,22 (+0,28)	5,22	5,28
TOTALT	35,91 (+0,2)	32,89 (+1,77)	30,85	33,22

Tabell 5.6: Spelupplevelse av Kivy-implementation

Kategori	Vardagsgrupp 1 (GameMaker)	Vardagsgrupp 2 (Kivy)
Användarbarhet /Spelbarhet	3,85	4,40
Spelupptagenhet	3,93	3,22
Nöje	5,11	4,70
Kreativitetsfrihet	4,00	3,84
Ljud estetik	4,29	4,71
Personlig tillfredsställelse	4,99	4,75
Visuell estetik	4,94	5,22
TOTALT	31,12	30,85

flyttning så tyckte fyra stycken att styrkorsförflyttning var bäst och två stycken att fingerförflyttning var bäst. Det beslutades därför att vid sluttestet så skulle styrkorsförflyttning och direktplacering användas.

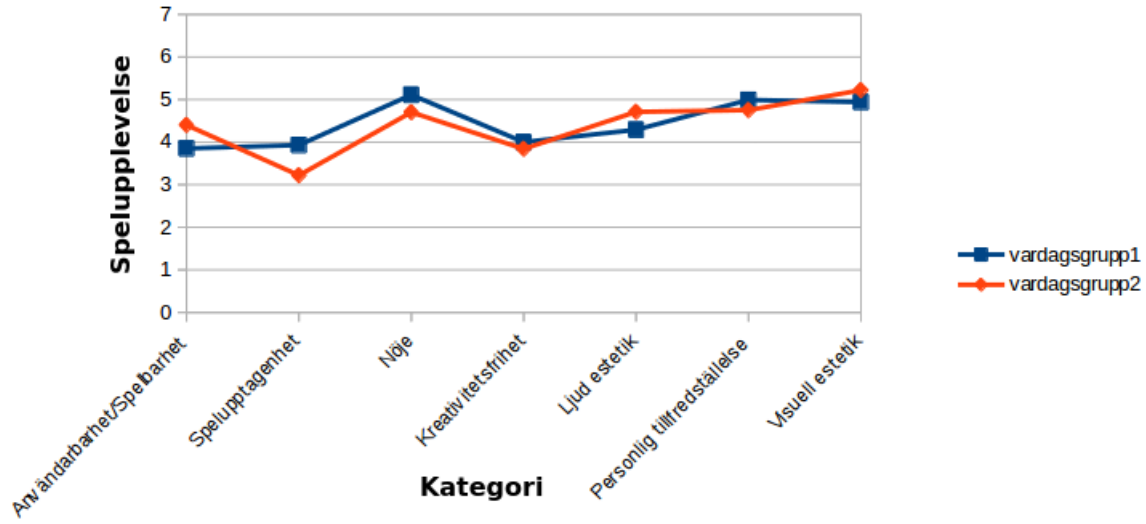
Dessutom implementerades objektpooler, detta beskrivs i avsnitt 5.7.3.

5.5 Utvärdering

Efter den tredje iterationen var slutförd så gjordes ett sluttest av Spobot. Resultatet av detta test finns i tabell 5.5 Parenteser i tabellen anger förändring från resultatet av speltestet på GameMaker versionen som gjordes i förstudien.

Svaren på Kivy-implementationen från vardagsgrupp 2 kan också jämföras med svaren på GameMaker versionen för vardagsgrupp 1 eftersom de två grupperna ska vara likvärdiga. Denna jämförelse visas i tabell 5.6.

För att se hur troligt det är att vardagsgrupp 1 och vardagsgrupp 2 är likvärdiga så gjordes ett χ^2-test, det visade att dessa grupper med 99.94% sannolikhet tillhör samma fördelning. I figur 5.1 visas också resultatet från de olika spelupplevelse-kategorierna från de två grupperna.



Figur 5.1: Spelupplevelse för vardagsgrupp 1 och vardagsgrupp 2

5.6 Prototyper

I detta avsnitt beskrivs de prototyper som utvecklades med hjälp av testpersonerna i prototypgruppen.

5.6.1 Dra-och-släpp-placering

I förstudien användes Dra-och-släpp-placering som enda sätt att placera torn på. När spelaren tryckte ner en tornknapp så visades ett rutnät och spelaren kunde då dra fingret till en ruta som spelaren ville placera tornet på. Rutan som spelaren hade fingret över blev grön om det var möjligt att placera på den rutan och röd annars. De krav som fanns för att det skulle vara möjligt att placera på en ruta var att inga objekt fick finnas i den rutan och den måste vara inom spelarens placeringsradie.

Vid speltester observerades att det både var svårt att förstå hur detta fungerade och att det var klumpigt. Fingret blockerade ofta hela rutan som spelaren ville placera på så det var då svårt att se om det gick att placera på den rutan eller inte. Därför bestämdes det att nya prototyper för tornplacering skulle tas fram.

5.6.2 Magikostnad

Version 1

Magier kostade i förstudien mana, som var en extra resurs som spelaren fick med tiden på samma sätt som pengar.

Det observerades vid speltester att det var svårt att hålla koll på alla resurser, mana förväxlades ofta med pengar och det var svårt att förstå vad som kostade pengar och vad som kostade mana. Prototypgruppen uppgav att de tyckte att magier skulle vara gratis att kasta men ha en nedkylningstid istället.

Version 2

I andra iterationen implementerades därför en nedkylningstid för magier. När en spelare använde en magi så täcktes den magi-knappen av en genomskinlig cirkel. För varje sekund som

sedan gick så togs ett segment av cirkeln bort, när hela cirkeln var borta så kunde spelaren använda den magin igen.

Prototypgruppen uppgav efter prototyptestningen efter den andra iterationen att detta fungerade bättre än att magierna kostade mana. Inga ytterligare förbättringsförslag för prototypen gavs.

Slutgiltlig version

Prototypen gjordes till färdig funktion genom att istället för att ha en cirkel över knapparna så gjordes en form som passar knapparna precis, så att det ser ut som att knappen blir lite gråare och sedan försvinner en grå del för varje sekund som passerar tills dess att knappen helt har sin originalfärg och då kan magin användas igen.

5.6.3 Spelarsköld

I förstudien hade spelaren en sköldegenskap, när spelaren tog skada så gick först skölden ner. När skölden var slut så började spelaren förlora hälsopoäng istället. När spelaren inte hade tagit skada på några sekunder så började sköldegenskapen att öka igen, men förlorade hälsopoäng var förlorat för alltid.

Vid speltester så observerades det att det var svårt att hålla isär spelarskölden och hälsopoäng. Prototypgruppen tyckte att det skulle vara bättre att ta bort sköldegenskapen helt och endast använda hälsopoäng istället.

När spelet i den första iterationen återskapades så implementerades därför inte skölden över huvud taget. Prototypgruppen tyckte också efter testning att spelet var bättre utan spelarsköld, denna prototyp lades därför ner.

5.6.4 Tvåtrycksplacering

Efter speltestet i förstudien så gav en testperson i prototypgruppen som förbättringsförslag att torn ska placeras med två tryck.

Detta implementerades i den andra iterationen genom att en spelare först trycker på det torn som den vill placera, då lyser de rutor som är möjliga att placera på grönt, om spelaren sedan trycker på någon av de gröna rutorna så placeras tornet där.

Prototypgruppen tyckte att detta fungerade bättre än dra-och-släpp placeringen, de hade inga fler förbättringsförslag på denna prototyp.

5.6.5 Direktplacering

Efter förstudien diskuterade spelutvecklarna hur tornplaceringen kan bli bättre. Utvecklarna kom då fram till att torn skulle kunna placeras direkt med endast ett tryck.

Version 1

Detta implementerades i den första iterationen. När en spelare trycker på en tornknapp så placeras det tornet i rutan framför karaktären om den är ledig. Skulle rutan vara upptagen så meddelas istället spelaren att det inte går att placera tornet där.

Efter prototyptestningen i andra iterationen så tyckte flera testpersoner att om rutan framför är upptagen så ska tornet kunna placeras på någon annan ruta i närheten av karaktären.

Figur 5.2: Placeringsordning för direktplacering

Version 2

I tredje iterationen gjordes därför prototypen om så att den kan placera tornet på alla rutor runt om karaktären förutom de tre rutor som är bakom karaktären. Det valdes att torn inte placeras på dessa rutor för att undvika att en spelare råkar bygga in sig själv. Det valda tornet placeras på den första rutan som är ledig i ordningen som visas i figur 5.2.

Prototypgruppen tyckte att detta fungerade bättre än version 1.

5.6.6 Fingerförflyttning

Version 1

I förstudien användes fingerförflyttning som enda sätt att förflytta spelarkaraktären på. När en spelare trycker ner fingret någonstans på spelplanen så går karaktären raka vägen mot den position som spelaren håller fingret.

Version 2

Efter förstudien kom några personer i prototypgruppen fram till idén att spelaren ska gå mot en position en bit framför fingret istället, detta då det var svårt att undvika motståndarens projektiler när spelaren försvann bakom fingret. Denna förändring implementerades i den andra iterationen.

Detta tyckte prototypgruppen fungerade bättre än version 1. Inga fler förbättringsförslag gavs.

5.6.7 Styrkorsförflyttning

Några testpersoner tyckte efter förstudien att det skulle kunna vara fördelaktigt att spelarkaraktären förflyttades med hjälp av ett styrkors.

Version 1

I den andra iterationen implementerades därför ett styrkors. Eftersom den inte fick plats på vektygsfältet så placerades den i nedre-bortre hörnet på motståndarsidan.

Efter prototyptestningen i den andra iterationen så kommenterades då att styrkorset måste göras genomskinligt eftersom motståndaren annars kan gömma sig bakom det. Det kommenterades även att styrkorset bör göras större.

Version 2

I den tredje iterationen gjordes därför prototypen om så att styrkorset täcker större delen av motståndarsidan och är genomskinlig.

Prototypgruppen tyckte efter prototyptestningen i den tredje iterationen att detta fungerade bättre än version 1.

5.6.8 Spelplanhalva

Version 1

I förstudien så hamnade spelaren som anslöt sig först på vänster planhavla och den andra spelaren på höger.

Vid speltester observerades dock att testpersonerna blev förvirrade när de spelade på ena sidan först och i nästa spel hamnade på den andra.

Version 2

Därför gjordes denna prototyp om i den andra iterationen till att spelarna själva får välja vilken planhalva de vill spela på. Om båda spelarna väljer exempelvis vänster sida så speglas alla motståndarens koordinater så att det ser ut som att den spelar på höger sida.

Efter prototyptestningen i den andra iterationen så angav testpersonerna att det var en stor fördel att alltid kunna spela på samma sida. Inga fler förbättringsförslag angavs.

5.6.9 Vådabeskjutning

Version 1

Torn kunde i förstudien skada andra torn byggda av samma spelare. Flera testpersoner i prototypgruppen tyckte dock att det gav en för stor begränsning på var man kan placera torn.

Version 2

I andra iterationen implementerades därför en prototyp där torn inte skadar andra torn byggda av samma spelare.

Prototypgruppen tyckte efter prototyptestningen att detta fungerade bättre än version 1 och inga fler förbättrignsförslag gavs.

5.7 Prestanda

I detta avsnitt redovisas hur olika tekniker för att höja prestandan implementerades och vilken effekt de hade på prestandan.

I resultat-tabellerna så är endast datan i den vänstra kolumnen oberoende variabler medan datan i de andra kolumnerna är beroende variabler. Detta innebär att endast det som står i den vänstra kolumnen i dessa tabeller har bestämts innan testen, de övriga kolumnerna (Minsta bildhastighet, medelbildhastighet och medelexekveringstid) är resultat från det specifika testet.

5.7.1 Kollisionsalgoritm

Design

För att få algoritmen så generell som möjligt så krävs en design med så få antaganden av de program eller spel som kommer använda algoritmen som möjligt. Dock behövs några antaganden för att det ska gå att göra något över huvudtaget. Följande antaganden gjordes:

- Kollisionsobjekt innehåller en variabel *mask* som är en tvådimensionell lista där varje element motsvarar en pixel i kollisionsobjektets bild. Ett element har värdet *True* om motsvarande pixel ska kunna kollidera med andra objekt och *False* annars
- Kollisionsobjekt har en metod *on_collision(self, other)* som innehåller kod som ska köras om objektet kolliderar med ett annat
- Kollisionsobjekt innehåller variablerna *x*, *y*, *width*, *height* som innehåller objektets position och storlek. Positionen ska ange objektets nedre vänstra hörn och storleken ska vara den minsta rektangel som objektet får plats i
- Varje kollisionsobjekt har en variabel *id* som är en unik identifierare för objektet
- Om kollisionsmängder används så har alla kollisionsobjekt en variabel *set_strategy* som innehåller en mängdstrategi-klass

Algoritmen implementerades som en singleton-klass de publika metoderna som beskrivs i tabell 5.7

Tabell 5.7: Metoder i kollisionsdetektionsalgoritm

Metodnamn	Funktionalitet
handle_collisions()	Kör själva detektionsalgoritmen och anropar on_collision metoden för alla kolliderande objekt.
add_object(object)	Lägger till ett objekt som ska kollisionskontrolleras.
remove_object(object)	Tar bort ett objekt som inte längre ska kollisionskontrolleras.
collide_object(object1, object2)	Returnerar *True* om *object1* och *object2* kolliderar, annars returneras *False*.

Kollisionsdetektions-klassen har tre statiska variabler: *USE_COLLISION_SETS*, *USE_SORT_AND_SWEEP* och *USE_QUAD_TREE*. Genom att sätta en av dessa variabler till *True* så väljs den motsvarande tekniken som kollisionsdetektionsteknik.

Prestandatest

Ett första prestandatest av de olika kollisionsteknikerna gav resultatet som visas i tabell 5.8. Att använda sig av olika kollisionsmängder gjorde att exekveringstiden för kollisionsdetektionsalgoritmen blev cirka 3,5 gånger snabbare jämfört mot att endast använda sig av AABB. Sort and sweep gav istället en förbättringsfaktor på cirka 1.5 sett till exekveringstid. När ett quadtree användes så ökades istället exekveringstiden för kollisionsdetektionsalgoritmen. Detta är troligtvis för att Spobot inte innehåller tillräckligt många objekt samtidigt för att det ska löna sig med ett quadtree.

Tabell 5.8: Prestanda av kollisionsalgoritm med normallast för Spobot

Använda tekniker	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]	Medelexekveringstid [ms]
AABB	10	29,8	4,2
AABB och kollisionsmängder	7	34,6	1,2
AABB och quadtree (djup 3)	7	26,0	7,2
AABB och sort and sweep (insättningssortering)	7	32,8	2,8

För att kunna testa hur bra de olika teknikerna fungerar i spel där det kan finnas fler objekt så skapades två nya typer av objekt i Spobot, dessa kommer fortsättningsvis kallas för testobjekt. Båda typerna av testobjekt är fyrkanter som skapas på ett slumpmässigt ställe med en slumpmässig riktning som sedan studsar runt. Det ena typen studsar mot allt, medans den andra typen inte studsar mot andra testobjekt av samma typ. Detta för att kunna testa kollisionsmängder, den första typen blir då en *MovingCollider* medan den andra blir en *MovingCollidee*.

För att bestämma vilken sorteringsalgoritm i sort and sweep algoritmen som fungerar bäst samt vilket djup som Spobots quadtree fungerar bäst med så testades de olika variationerna med 200 testobjekt, 100 av båda typerna. Resultaten finns i tabell 5.9 och tabell 5.10

Tabell 5.9: Prestanda av quadtree med olika djup med 200 objekt som rör på sig

Djup	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]	Medelexekveringstid [ms]
3	3	4,0	69,5
4	3	4,0	83,4

Tabell 5.10: Prestanda av sort and sweep med olika sorteringsalgoritmer med en last på 200 objekt i rörelse

Sorteringsalgoritm / sorteringsriktning	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]	Medelexekveringstid [ms]
Insättningssortering / horisontellt	3	5,0	18,6
Insättningssortering / vertikalt	3	4,9	19,0
Quicksort / horisontellt	3	4,0	66,9
Quicksort / vertikalt	3	3,9	51,6
Timsort / horisontellt	3	4,0	18,5
Timsort / vertikalt	3	4,0	19,7

Tabell 5.11: Prestanda av kollisionsalgoritm med 200 objekt i rörelse

Använda tekniker	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]	Medelexekveringstid [ms]
AABB	3	2,9	174,2
AABB och kollisionsmängder	3	3,0	143,4
AABB och quadtree	3	4,0	69,5
AABB och sort and sweep	3	5,0	18,6

Även kollisionsmängder samt att endast använda AABB testades med 200 testobjekt. I tabell 5.11 redovisas dessa resultat tillsammans med de bästa resultaten från quadtree och sort and sweep algoritmerna.

För att ta reda på vilka tekniker som fungerar bäst för spel med olika många objekt aktiva samtidigt så testades de även med 100 och 50 testobjekt. Resultaten från dessa test finns i tabell 5.12 och tabell 5.13.

Från dessa resultat ser vi att det räcker med 50 objekt för att det ska vara lönsamt att använda sig av ett quadtree jämfört med att bara använda sig av AABB. Däremot så var sort and sweep den snabbaste tekniken i alla dessa tester.

Tabell 5.12: Prestanda av kollisionsalgoritm med 100 objekt i rörelse

Använda tekniker	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]	Medelexekveringstid [ms]
AABB	4	4,5	73,6
AABB och kollisionsmängder	2	4,9	64,8
AABB och quadtree	5	6,7	28,7
AABB och sort and sweep	3	7,8	9,3

Tabell 5.13: Prestanda av kollisionsalgoritm med 50 objekt i rörelse

Använda tekniker	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]	Medelexekveringstid [ms]
AABB	12	15,0	13,6
AABB och kollisionsmängder	11	13,4	13,1
AABB och quadtree	9	14,0	12,1
AABB och sort and sweep	9	19,3	3,6

5.7.2 Bildblad

Design

Det gjordes ett bildblad för varje animation. Varje bild i ett bildblad döptes till ett nummer som motsvarar bildens position i animationen. Numret formaterades så att det alltid blev två tecken långt. Exempelvis så fick den första bilden i en animation namnet *01.png*.

Tabell 5.14: Metoder i Animation

Metodnamn	Funktionalitet
___init___(self, atlas, obj, anim_delay=0.04, loop=False, anim_end=orig_source)	Konstruktor, skapar ett nytt animationsobjekt
play(self)	Spelar upp animationen

Varje animation som ska kunna visas är en instans av en klass som heter *Animation*. Klassen har de publika metoder som beskrivs i tabell 5.14. Konstruktorn kräver alltså att *atlas* och *obj* anges, atlas är en länk till det atlas som ska användas och obj är objektet som animationen hör till. Konstruktorn tar även tre frivilliga argument, *anim_delay*, *loop* och *anim_end*. Med anim_delay så bestäms tiden mellan varje bildruta, loop bestämmer om animationen ska börja om när den är färdig och anim_end är den funktion som körs när animationen är färdig om den inte ska börja om.

Tack vare att bildbladen strukturerades som de gjorde så kan animationerna enkelt hanteras genom att varje Animation-instans har en räknare som motsvarar en bildruta i animationen. Kivys metod *Clock.schedule_interval* används för att räkna upp räknaren. För varje gång som räknaren räknas upp så sätts objektet *obj*:s textur till bilden i bildbladet med samma namn som det nummer räknaren är på.

Prestandatest

Ett prestandatest med två torn av varje sort genomfördes när ovanstående implementation av bildblad användes och jämfördes med att använda en zip fil med PNG-bilder. Det visade sig att genom att använda bildblad så försämrades prestandan med cirka en tredjedel. Exakt resultat redovisas i tabell 5.15

Tabell 5.15: Prestandaskillnad med bildblad

Använda tekniker	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]
PNG bilder i zip filer	10	21,4
Bildblad med hjälp av kivys atlas modul	6	13,9

5.7.3 Objektpool

Design

Objektpooler implementerades genom att varje klass som det kan skapas och tas bort instanser av fick två klass-variabel *OBJECT_POOL* och *OBJECT_POOL_CAP*. OBJECT_POOL_CAP anger hur många objekt som ska finnas i objektpoolen och OBJECT_POOL är en kö som representerar själva objektpoolen.

En statisk klass *ObjectPool* skapades också, denna klass har en statisk metod *create_object_pools()*. När den metoden anropas så skapas OBJECT_POOL_CAP antal instanser av alla spelobjekt-klasser.

Istället för att skapa ett nytt spelobjekt när det behövs så tas det första elementet ur motsvarande klass OBJECT_POOL ut och aktiveras för att sedan användas som den nya instansen. När ett objekt sedan tas bort från spelet så inaktiveras det och läggs till sist i OBJECT_POOL kön.

När ett objekt aktiveras så läggs det till i kollisionsdetekteraren samt som ett *game_object* vilket gör att objektets *update* metod körs. När det inaktiveras så tas det istället bort från båda dessa ställen.

För att undvika att objekten ritas ut i onödan så tas även de inaktiverade objektens texturer bort genom att sätta deras egenskap *source* till *None*.

5.7.4 Prestandatest

Vid prestandatestet så visade det sig att genom att använda den ovanstående implementationen av en objektpool så försämrades prestandan med nästan en tredjedel. Exakta värden finns i tabell 5.16.

Tabell 5.16: Prestandaskillnad med objektpool

Använda tekniker	Minsta bildhastighet [FPS]	Medelbildhastighet [FPS]
Dynamiskt skapande av objekt	20	29.3
Objektpool	17	21.1

6 Diskussion

6.1 Resultat

I detta avsnitt diskuteras resultatet för både prestandan och spelupplevelsen.

6.1.1 Prestanda

Både bildbladen och objektpooler är tekniker som brukar användas för att höja prestandan för spel, men i denna studie visade sig att båda teknikerna faktiskt försämrade prestandan.

När det gäller bildblad så skulle det kunna bero på att Kivys hantering av zip-filer är väl optimerad och använder sig av mer lågnivå funktioner och att det därför är väldigt svårt att implementera en egen animationshanterare med hjälp av atlas-modulen som är snabbare.

Gällande objektpool så verkar det som att Kivy hanterar många objekt på ett långsamt sätt, ett annat resultat som tyder på detta är resultatet för kollisionsalgoritmen med 100 objekt i rörelse som redovisas i tabell 5.12. Där ser vi att när AABB och sort and sweep användes som kollisionsdetektion så tog själva kollisionsdetektionen bara 9,3 ms vilket i sig skulle kunna ge en bildhastighet över 60 bilder/s, men det blev endast 7,8 bilder/s. När antalet objekt halverades till 50 st så mer än fördubblades bildhastigheten till 19,3 bilder/s vilket visas i tabell 5.13.

När det gäller kollisionsdetektionen så kan vi se att bildhastigheten inte varierar särskilt mycket för olika tekniker, detta beror troligtvis på att det vid testtillfället inte var kollisionsdetektionen som var flaskhalsen för bildhastigheten. Däremot så varierade exekveringstiden för kollisionsdetektionen ganska mycket beroende på vilken teknik som användes. Så om vi därför endast tittar på exekveringstiden för de olika algoritmerna så ser vi i tabellerna 5.8 - 5.13 att när väldigt få objekt ska kollisionstestas så går med ganska stor marginal snabbast att använda AABB och kollisionsmängder medan att använda sig av AABB och ett quadtree försämrar prestandan jämfört med att endast använda AABB. När det däremot kommer upp till över 200 objekt som ska kollisionstestas ser vi att AABB och kollisionsmängder är bättre än endast AABB men sämre än de andra testade teknikerna. Detta tyder på att AABB och kollisionsmängder har minst ursprungskostnad men skalar sämre när det blir fler objekt som ska kollisionstestas. Quadtree verkar istället ha dyrast ursprungskostnad men skala betydligt bättre än kollisionsmängder. Sort and sweep verkar dock ha en väldigt låg ursprungskostnad och dessutom skala bättre än quadtree i de tester som gjordes i denna studie. Något som inte testades i denna studie är att ha olika storlekar på spelområdet, detta är något som också skulle kunna påverka prestandan för de olika teknikerna.

6.1.2 Spelupplevelse

Många intressanta prototyper skapades med hjälp av prototypgruppen, vissa prototyper som exempelvis tvåtrycksplacering kom som förslag från en enskild person och tycktes om av samt-

liga andra i prototypgruppen medan andra förslag kom från utvecklarna men att prototypgruppen utvärderade och gav förbättringsförslag som sedan implementerades.

I tabell 5.5 kan vi se att förändringarna mellan de två spelupplevelse testen i allmänhet var ganska liten för prototypgruppen och vardagsgrupp 1. Det som sticker ut mest är att för båda grupperna så ökade Användarbarhet/Spelbarhet med mer än en halv poäng. Detta beror antagligen på att de flesta prototyper som utvecklades hade detta område som fokus. En annan ganska tydlig trend för dessa grupper var att nöjet minskade, inte särskilt signifikant för prototypgruppen men desto mer för vardagsgrupp 1. En möjlig orsak till detta är att prestandan blev klart sämre för Kivy versionen än GameMaker versionen. Den sista stora förändringen är ljud-estetiken, vilket är väldigt intressant eftersom exakt samma ljudfiler används i båda implementationerna. Det som möjligen skulle kunna vara orsak till denna förändring är att GameMaker och Kivy spelar upp ljuden på olika sätt. Startar man den ena versionen först och sedan den andra så hör man att ljudet låter annorlunda. Detta i sin tur beror antagligen på att Kivy och GameMaker använder olika kompressionsalgoritmer för ljudet.

En annan intressant observation är att vardagsgrupp 2 i allmänhet har gett lägre betyg än vardagsgrupp 1. Detta skulle kunna bero på att personer ger spelet högre betyg när de redan testat det en gång eftersom de då känner igen spelet och vet vad det går ut på.

I tabell 5.6 ser vi en jämförelse mellan hur vardagsgrupp 1 svarade på GameMaker versionen och hur vardagsgrupp 2 svarade på Kivy versionen. Denna jämförelse stöttar det vi tidigare såg, att Användarbarheten/Spelbarheten av spelet har ökat, nöjet har minskat samt att ljud-estetiken har ökat. Det som sticker ut i denna jämförelse dock är att spelupptagenheteten har minskat kraftigt. Vi kunde se en liten försämring på detta även för prototypgruppen, däremot så ökade detta en aning för vardagsgrupp 1. Totalt sett blir det dock en ganska tydlig minskning, denna minskning skulle precis som nöjet kunna bero på att spelets prestanda försämrats och att det därför är svårare att leva in i spelet.

När det gäller hur trovärdigt det är att vardagsgrupp 1 och vardagsgrupp 2 kommer från samma fördelning så kan vi i figur 5.1 se att skillnaderna på svaren i allmänhet är väldigt små för dessa grupper. χ^2-testet stöttar även detta antagande, även om chi^2-test är säkrare med flera datapunkter så är det svårt att tänka sig att det skulle vara någon större skillnad på dessa två grupper när testet gav 99,94% sannolikhet att de tillhör samma mängd.

När det gäller den totala spelupplevelsen så är den totala förändringen extremt liten, från 33,42 till 33,22. Eftersom förändringen är såpass liten så kan det tolkas som att spelupplevelsen blev oförändrad.

6.2 Metod

I detta avsnitt diskuteras metoden som användes både för att höja spelupplevelsen och prestandan.

6.2.1 Spelupplevelse

Mätning av spelupplevelse

Mätning av spelupplevelsen gjordes i detta arbete med hjälp av GUESS. GUESS valdes främst för att det är väldigt nytt och baserat på ny forskning kring spelupplevelse. Detta arbete handlade mest om att förbättra spelupplevelsen genom att omdesigna olika funktioner, men i GUESS så ingår mycket mer än bara hur spelare upplever olika funktioner. Exempelvis ingår ljud estetik och visuell estetik som faktorer i GUESS och detta är områden som detta arbete inte har haft som avsikt att ändra på.

Förbättring av spelupplevelse

För att förbättra spelupplevelsen av spelet så användes prototypdriven utveckling. Detta gjorde att spelare i spelets målgrupp fick vara med och ta fram, testa och värdera olika prototyper för olika funktioner. Skulle mer tid funnits för denna studie så skulle fler testpersoner finnas i prototypgruppen, detta skulle troligtvis då leda till fler och mer varierade prototyper som skulle kunna testas. Om det dessutom skulle hållas diskussionspass så skulle prototypgruppen delas in i flera fokusgrupper vilket enligt R. Krueger och M.A Casey [21] brukar leda till väl genomtänkta idéer.

6.2.2 Prestanda

Mätning av prestanda

Bildhastighet var det som valdes som huvudsakligt prestandamått, problemet med detta var dock att väldigt många faktorer påverkar bildhastigheten och som vi såg i kollisionsdetektionsalgoritmen så gav bildhastigheten nästan ingen data när många testobjekt användes eftersom kollisionsdetektionsalgoritmen då inte var flaskhalsen för bildhastigheten. Vid testning av specifika funktioner skulle därför exekveringstid varit ett bättre mått att använda. Men eftersom bildhastigheten enligt M. Claypool och K. Claypool [29] påverkar hur spelarna presterar i spel så bör bildhastigheten användas som ett mått för den generella prestandan och möjligtvis användas på ett liknande sätt som GUESS användes för spelupplevelsen i detta arbete, för att se om det blev någon prestandaökning i slutändan.

Prestandaökning

Några akademiska artiklar för prestandaökning av spel kunde inte hittas, följande sökord användes på unisearch: *Video game performance*, *Video game frame rate*, *Computer game performance*, *Computer game frame rate*, *Smartphone game performance* och *Smartphone game frame rate*. Istället användes olika tekniker och metoder som hittades på olika online-källor. Skulle mer tid funnits hade flera tekniker kunnat testats. Om projektet skulle göras om nu så skulle spelet även utvecklas i Kivent som är en spelmotor till Kivy som enligt deras hemsida ska klara av att köra i 60 bilder/s när 5000 animerade objekt som förflyttar sig. [38] Det skulle antagligen vara mycket bättre att testa olika prestandalösningar i detta eftersom Kivy verkar vara den största flaskhalsen för prestandan i detta projekt.

6.3 Arbetet i ett vidare sammanhang

När det gäller datorspel, tvspel och mobilspel så pratas det ofta om negativa effekter som exempelvis spelberoende. Men i en studie av R. Sevin och W. DeCamp [39] så kommer författarna fram till att datorspel ökar både självförtroende i hantering av datorer samt intresse för datorvetenskap. Detta spel är visserligen inget datorspel, utan ett spel till smartphones. Men då spelet lika gärna skulle kunna vara ett datorspel så ser jag ingen anledning att spelande av detta spel också skulle öka intresse av datorvetenskap. Men istället för att öka självförtroende i hantering av datorer så ökas antagligen självförtroendet i hantering av smartphones istället.

Det finns också belägg för vissa negativa effekter, i en studie av T. Greitemeyer och D.O. Mügge [40] så visar författarna att våldsamma spel ökar aggression hos spelarna. Dock visar också samma studie att mer sociala spel också ökar social kompetens. Om detta spel är ett våldsamt spel går att diskuteras. Visserligen går spelet ut på att förstöra sin motståndarspelare genom att skjuta mot den samt placera ut kanontorn som skjuter mot den. Men det finns inget blod, och det är en robot man ska förstöra. Ofta brukar våldsamma spel förknippas med just dödande och mycket blod. Även om spelet efter detta arbete inte har någon social interaktion med andra spelare så kommer spelet ha det innan det släpps. Så när personer kan spela detta spel så kommer det alltså vara ett socialt spel och därmed kunna öka social kompetens hos spelarna.

7 Sammanfattning

Den första frågan i detta arbetets frågeställning var hur ett tvådimensionellt onlinespel i kategorin action-strategi kan implementeras till både Android och IOS med hjälp av Kivy för att det ska få högre spelupplevelse. Detta arbete kan inte helt svara på denna fråga då spelupplevelsen blev i princip oförändrad. Däremot så ökades användarbarheten och spelbarheten av spelet genom att använda prototypdriven utveckling där speltestare fick vara med och ta fram och utvärdera olika prototyper. Även ljud-estetiken ökades direkt bara genom att använda Kivy istället för GameMaker. Ett stort problem var dock prestandan i Kivy, och troligtvis var det den försämrade prestandan som drog ner spelupplevelse-kategorierna *nöje* och delvis *personlig tillfredställelse*, vilket slutligen gjorde att den totala spelupplevelsen i stort sätt blev oförändrad.

Den andra frågan var hur ett sådant spel kan implementeras i Kivy för att uppnå maximal bildhastighet. I detta arbete testades prestandan för när Kivys atlas modul användes som bildblad för animationer och jämfördes med att istället använda zip-filer med .png bilder. Det visade sig att det gav bättre prestanda att använda zip-filer. Skillnaden mellan att skapa objekt dynamiskt samt att använda en objektpool undersöktes också, det visade sig att det gav bättre prestanda att skapa objekt dynamiskt.

Den tredje och sista frågan var hur en generell pixelperfekt kollisionsdetektions algoritm för sådana spel kan implementeras som är så snabb som möjligt. Pixelperfekt kollision implementerades genom att alla objekt har en sanningstabell som beskriver om motsvarande pixel ska vara en kollisionspixel eller inte. AABB kollision kontrollerades innan sanningstabellen itererades igenom. Flera olika tekniker för att minska antalet AABB kollisioner som behöver kontrolleras testades också. Det visade sig att när färre än 50 objekt användes så var att använda sig av kollisionsmängder den snabbaste av de provade teknikerna. När fler objekt användes så var istället *sort and sweep* den snabbaste tekniken.

Från detta kan vi dra följande slutsatser:

- Kivy i sig självt är inte lämpligt för att implementera action-strategispel på grund av prestandaproblem
- Prototypdriven utveckling fungerar bra för att förbättra Användarbarhet/Spelbarhet, men inte för andra delar av spelupplevelsen
- Om ett spel innehåller 200 objekt eller färre så fungerar sort and sweep väldigt bra som kollisionsdetektions-algoritm

7.1 Framtida arbete

I ett framtida arbete skulle en liknande prestandaundersökning med hjälp av spelmotorn Kivent till Kivy kunna göras. Där finns dock redan kollisionsdetektion färdigt, men bildblad, objektpooler och andra tekniker skulle kunna prestandatestas i Kivent.

Litteratur

[1] Python Software Foundation. *The Python Tutorial*. 2016. URL: https://docs.python.org/2.7/tutorial/index.html (hämtad 2016-11-29).

[2] Python Software Foundation. *Python 2.7 release schedule*. 2008. URL: http://legacy.python.org/dev/peps/pep-0373/ (hämtad 2016-12-01).

[3] Python Software Foundation. *Python 3.4 release schedule*. 2015. URL: https://www.python.org/dev/peps/pep-0494/ (hämtad 2016-12-01).

[4] Python Software Foundation. *Classes*. 2016. URL: https://docs.python.org/2.7/tutorial/classes.html (hämtad 2016-11-29).

[5] Python Software Foundation. *Modules*. 2016. URL: https://docs.python.org/2/tutorial/modules.html (hämtad 2016-12-01).

[6] Python Software Foundation. *Time*. 2016. URL: https://docs.python.org/2/library/time.html (hämtad 2017-07-30).

[7] Python Software Foundation. *Datastructures*. 2016. URL: https://docs.python.org/2.7/tutorial/datastructures.html (hämtad 2016-12-01).

[8] Kivy authors. *Kivy Framework*. 2016. URL: https://kivy.org/docs/api-kivy.html (hämtad 2016-11-29).

[9] Kivy authors. *Widget class*. 2016. URL: https://kivy.org/docs/api-kivy.uix.widget.html (hämtad 2016-12-01).

[10] Kivy authors. *Kivy Language*. 2016. URL: https://kivy.org/docs/api-kivy.lang.html (hämtad 2016-12-01).

[11] Kivy authors. *Kivy Language*. 2016. URL: https://kivy.org/docs/guide/lang.html (hämtad 2017-01-07).

[12] Kivy authors. *Kivy Language*. 2016. URL: https://kivy.org/docs/api-kivy.atlas.html (hämtad 2017-03-09).

[13] Kivy authors. *Kivy Language*. 2017. URL: https://kivy.org/docs/api-kivy.clock.html (hämtad 2017-05-24).

[14] Eduardo H Calvillo-Gámez, Paul Cairns och Anna L Cox. "Assessing the core elements of the gaming experience". I: *Game User Experience Evaluation*. Springer, 2015, s. 37–62.

[15] Karolien Poels, Yvonne De Kort och Wijnand Ijsselsteijn. "It is always a lot of fun!: exploring dimensions of digital game experience using focus group methodology". I: *Proceedings of the 2007 conference on Future Play*. ACM. 2007, s. 83–89.

[16] Laura Ermi och Frans Mäyrä. "Fundamental components of the gameplay experience: Analysing immersion". I: *In DIGRA. DIGRA*. 2005.

[17] Pietro Guardini och Paolo Maninetti. "Better Game Experience Through Game Metrics: A Rally Videogame Case Study." I: *Game Analytics* (2013), s. 325. ISSN: 9781447147688.

[18] John P Davis, Keith Steury och Randy Pagulayan. “A survey method for assessing perceptions of a game: The consumer playtest in game design”. I: *Game Studies* 5.1 (2005).

[19] Mikki H Phan, Joseph R Keebler och Barbara S Chaparro. “The Development and Validation of the Game User Experience Satisfaction Scale (GUESS)”. I: *Human Factors: The Journal of the Human Factors and Ergonomics Society* (2016), s. 0018720816669646.

[20] WA IJsselsteijn, YAW de Kort och K Poels. “The game experience questionnaire”. I: *Manuscript in preparation* (2008).

[21] Richard A Krueger och Mary Anne Casey. *Focus groups: A practical guide for applied research*. Sage publications, 2014.

[22] Mattias Arvola. *Interaktionsdesign och UX: om att skapa en god användarupplevelse*. Studentlitteratur, 2014.

[23] Jon Manker och Mattias Arvola. “Prototyping in game design: Externalization and internalization of game ideas”. I: *Proceedings of the 25th BCS Conference on Human-Computer Interaction*. British Computer Society. 2011, s. 279–288.

[24] Ian Molyneaux. *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*. Ö'Reilly Media, Inc.", 2009.

[25] Bob Wescott. *Every Computer Performance Book: How to Avoid and Solve Performance Problems on The Computers You Work With*. 1st. USA: CreateSpace Independent Publishing Platform, 2013. ISBN: 1482657759, 9781482657753.

[26] Rijwan Khan och Mohd Amjad. “Performance testing (load) of web applications based on test case management”. I: *Perspectives in Science* 8 (2016), s. 355–357.

[27] Avisekhar Roy. *The Android Game Developer's Handbook*. Packt Publishing Ltd, 2016.

[28] Mark Claypool, Kajal Claypool och Feissal Damaa. “The effects of frame rate and resolution on users playing first person shooter games”. I: *Electronic Imaging 2006*. International Society for Optics och Photonics. 2006, s. 607101–607101.

[29] Mark Claypool och Kajal Claypool. “Perspectives, frame rates and resolutions: it's all in the game”. I: *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM. 2009, s. 42–49.

[30] Matt Kain Lewandowski och Jake Lewandowski. *2D Mobile Game Performance in Unity Part 1*. 2015. URL: `http://team2bit.com/wordpress/2015/04/12/mobile-frame-rate-and-memory-issues-in-unity/` (hämtad 2017-02-28).

[31] Nick Bobic. *Advanced Collision Detection Techniques*. 2000. URL: `http://www.gamasutra.com/view/feature/131598/advanced_collision_detection_.php?page=1` (hämtad 2017-02-28).

[32] Steven Lambert. *HTML5 Canvas Game: 2D Collision Detection*. 2013. URL: `http://blog.sklambert.com/html5-canvas-game-2d-collision-detection/` (hämtad 2017-03-09).

[33] Kyle Schouviller. *QuadTree*. 2007. URL: `http://www.kyleschouviller.com/wsuxna/quadtree-source-included/` (hämtad 2017-03-01).

[34] Nilson Souto. *Video Game Physics Tutorial - Part II: Collision Detection for Solid Objects*. 2014. URL: `https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects` (hämtad 2017-03-03).

[35] Dejaime Antônio de Oliveira Neto. *Intelligent 2D Collision and Pixel Perfect Precision*. 2013. URL: `https://www.gamedev.net/resources/_/technical/game-programming/intelligent-2d-collision-and-pixel-perfect-precision-r3311` (hämtad 2017-03-02).

[36] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[37] David MacIver. *Understanding timsort, Part 1: Adaptive Mergesort*. 2010. URL: `http://www.drmaciver.com/2010/01/understanding-timsort-1adaptive-mergesort/` (hämtad 2017-03-09).

[38] Kovak. *Kivent*. 2017. URL: `http://www.kivent.org/` (hämtad 2017-06-13).

[39] Rebecca Sevin och Whitney DeCamp. "From Playing to Programming: The Effect of Video Game Play on Confidence with Computers and an Interest in Computer Science". I: *Sociological Research Online* 21.3 (2016), s. 16.

[40] Tobias Greitemeyer och Dirk O Mügge. "Video games do affect social outcomes a meta-analytic review of the effects of violent and prosocial video game play". I: *Personality and Social Psychology Bulletin* (2014), s. 0146167213520459.

A Enkätsvar GUESS

Observera att det finns sex personer i varje grupp, svaren är sorterade i stigande ordning, svar som saknas betyder att personen som inte svarade tyckte att påståendet inte var tillämpningsbart på Spobot.

A.1 Svar för GameMaker version

Tabell A.1: GUESS svar del 1/3

Påstående	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp2
I think it is easy to learn how to play	4, 4, 5, 5, 7, 7	3, 4, 5, 5, 5, 6	-
I cannot tell that i am getting tired while playing the game	4, 5, 5, 7, 7	2, 4, 5, 5, 6	-
I am in suspense about whether I will succeed in the game	4, 5, 5, 6, 7	3, 3, 6, 6, 6, 7	-
I feel the game allows me to be imaginative	3, 5, 5, 6, 6	2, 3, 4, 5, 5, 6	-
I think the game is fun	5, 6, 6, 6, 6, 7	3, 5, 5, 6, 6, 6	-
I enjoy the sound effects in the game	4, 5, 5, 7, 7	1, 3, 4, 4, 6, 6	-
I find the controls of the game to be straightforward	3, 4, 4, 5, 6, 6	3, 4, 4, 4, 5, 5	-
I tend to spend more time playing the game than I have planned	6	1, 1, 4, 5, 7	-
I feel creative while playing the game	4, 4, 5, 5, 5, 6	2, 2, 4, 4, 5, 6	-
I feel successful when I overcome the obstacles in the game	5, 5, 6, 6, 6	4, 4, 5, 6, 6	-
I always know how to achieve my goals/objectives in the game	3, 3, 4, 5, 5, 7	1, 2, 4, 6, 7	-
I feel bored while playing the game	1, 1, 1, 1, 2, 2	2, 2, 3, 3, 5	-
I think the game's audio fits the mood or style of the game	4, 6, 6, 6, 7	3, 4, 5, 5, 5, 6	
I find the game's interface to be easy to navigate	2, 4, 4, 6, 6, 7	2, 3, 3, 5, 5, 5	-

Tabell A.2: GUESS svar del 2/3

Påstående	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp2
I feel the game constantly motivates me to proceed further to the next stage or level	4, 4, 5	1, 2, 4, 4	-
Whenever I stopped playing the game I cannot wait to start playing it again	1, 2, 3, 6	2, 4, 5, 5, 5	-
I enjoy the game's graphics	3, 5, 5, 5, 6, 7	4, 4, 5, 5, 6, 6	-
I feel detached from the outside world while playing the game	2, 2, 3, 4, 6	1, 2, 3, 3, 5	
I feel I can explore things in the game	3, 3, 5, 5, 7	2, 2, 3, 4, 4, 5	-
I find my skills gradually improve through the course of overcoming the challenges in the game	5, 5, 6, 6, 7, 7	1, 5, 5, 7, 7	-
I do not need to go through a lengthy tutorial or read a manual to play the game	2, 3, 5, 6, 6, 7	1, 3, 3, 4, 5, 6	-
I can block out most other distractions when playing the game	4, 5, 5, 6, 7	1, 5, 5, 5, 5, 6	-
If given the chance, I want to play this game again	5, 5, 5, 6, 7, 7	2, 3, 7, 7, 7	-
I feel the game's audio (e.g., sound effects, music) enhances my gaming experience	4, 5, 5, 7, 7	2, 4, 4, 5, 5	-
I find the game's menus to be user friendly	4, 5, 6, 6, 6, 6	3, 3, 4, 5, 6, 6	-
I feel the game allows me to express myself	1, 2, 2, 6	1, 2, 3, 4, 6	-
I feel the game trains me well in all of the controls	4, 4, 4, 6, 6	3, 4, 4, 5, 5	-
I am very focused on my own performance while playing the game	4, 6, 6, 6, 6, 7	3, 4, 6, 6, 7, 7	-

Tabell A.3: GUESS svar del 3/3

Påstående	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp2
I do not care to check events that are happening in the real world during the game	1, 4, 6, 6, 6	2, 5, 6, 6	-
I always know my next goal when I finish an event in the game	2, 3, 4, 5, 5	1, 1, 1, 4, 6	
I am likely to recommend this game to others	1, 4, 5, 5, 6, 7	1, 5, 5, 6, 6, 7	-
I think the game is visually appealing	2, 3, 5, 5, 6, 6	3, 4, 5, 5, 6, 6	-
I feel my curiosity is stimulated as the result of playing the game	3, 4, 4, 6	1, 3, 4, 5, 5, 6	-
Sometimes I lose track of time while playing the game	3, 3	1, 2, 3, 4, 4	-
I feel the game provides me the necessary information to accomplish a goal within the game	2, 3, 3, 4, 4, 7	1, 2, 3, 4, 5	-
I think the graphics of the game fits the mood or style of the game	4, 5, 5, 6, 7, 7	4, 5, 5, 5, 5, 6	-
I think the game is unique or original	5, 5, 5, 5, 5, 7	4, 4, 5, 5, 6, 7	-
I enjoy the music in the game	4, 4, 4, 7, 7	3, 4, 4, 5, 5, 6	-
I feel very confident while playing the game	2, 2, 5, 5, 7	1, 2, 4, 5, 5	-
I enjoy playing the game	5, 5, 6, 6, 6, 7	3, 4, 5, 5, 7, 7	-
I temporarily forget about my everyday worries while playing the game	5, 5, 6	3, 4, 5, 5, 5	-
I think the information provided in the game (e.g, onscreen messages, help) is clear	2, 3, 3, 5, 6, 7	3, 3, 4, 5, 5, 6	-
I feel the game gives me enough freedom to act how I want	4, 5, 5, 6, 6	3, 3, 4, 4, 6, 6	-
I want to do as well as possible during the game	4, 6, 6, 6, 7, 7	5, 6, 7, 7, 7, 7	-

A.2 Svar för Kivy version

Tabell A.4: GUESS svar del 1/3

Påstående	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp2
I think it is easy to learn how to play	5, 5, 5, 6, 6, 7	2, 4, 5, 6, 7, 7	3, 5, 5, 6, 6, 6
I cannot tell that i am getting tired while playing the game	4, 4, 4, 5, 6, 6	2, 3, 4, 4, 5, 6	1, 4, 5, 5, 6
I am in suspense about whether I will succeed in the game	1, 5, 5, 5, 6, 6	4, 4, 5, 5, 6, 6	5, 5, 5, 6, 7, 7
I feel the game allows me to be imaginative	3, 4, 5, 5, 6, 6	1, 4, 4, 5, 6, 6	4, 5, 5, 5, 6, 7
I think the game is fun	5, 5, 6, 6, 6, 7	2, 3, 5, 5, 5, 6	3, 4, 5, 5, 6, 7
I enjoy the sound effects in the game	4, 5, 6, 7, 7	3, 3, 5, 5, 6, 7	2, 4, 4, 5, 6, 7
I find the controls of the game to be straightforward	4, 5, 6, 6, 6, 7	4, 4, 5, 5, 5, 7	3, 4, 5, 6, 6, 6
I tend to spend more time playing the game than I have planned	1, 3, 4, 5, 6	2, 2, 5, 5, 5, 7	1, 1, 1, 2, 3, 4
I feel creative while playing the game	2, 4, 4, 5, 5, 6	3, 3, 4, 4, 4, 5	3, 3, 3, 4, 4, 5
I feel successful when I overcome the obstacles in the game	5, 5, 5, 5, 5, 6	4, 5, 5, 5, 6, 6	3, 4, 4, 6, 7
I always know how to achieve my goals/objectives in the game	4, 4, 5, 6, 6, 7	2, 3, 5, 5, 5, 6	2, 2, 4, 5, 6, 7
I feel bored while playing the game	1, 1, 1, 2, 2, 4	3, 3, 3, 3, 5, 5	1, 2, 3, 3, 5, 5
I think the game's audio fits the mood or style of the game	5, 6, 6, 7, 7	4, 5, 5, 6, 7, 7	4, 4, 5, 5, 6, 7
I find the game's interface to be easy to navigate	3, 5, 6, 6, 7, 7	1, 3, 4, 4, 5, 6	2, 3, 3, 6, 7

Tabell A.5: GUESS svar del 2/3

Påstående	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp2
I feel the game constantly motivates me to proceed further to the next stage or level	4, 4, 4, 4, 5	3, 4, 4, 5, 5, 6	1, 1, 4, 4, 6
Whenever I stopped playing the game I cannot wait to start playing it again	1, 4, 4, 4, 4, 5	2, 3, 3, 3, 4, 7	1, 1, 3, 4, 4
I enjoy the game's graphics	5, 5, 5, 5, 6, 7	4, 4, 5, 6, 6, 6	4, 4, 6, 6, 6, 7
I feel detached from the outside world while playing the game	1, 5, 5, 5, 5, 5	2, 3, 4, 5, 5, 7	2, 4, 4, 4, 4, 6
I feel I can explore things in the game	2, 3, 4, 4, 5	2, 3, 4, 4, 4, 4	1, 2, 2, 3, 5, 6
I find my skills gradually improve through the course of overcoming the challenges in the game	4, 5, 6, 6, 6, 6	3, 5, 5, 6, 6, 6	2, 4, 5, 6, 6, 6
I do not need to go through a lengthy tutorial or read a manual to play the game	3, 4, 4, 6, 6, 7	2, 4, 5, 5, 6, 6	2, 3, 5, 6, 6, 6
I can block out most other distractions when playing the game	5, 5, 5, 5, 6, 6	1, 3, 3, 4, 5, 5	1, 3, 5, 5, 5
If given the chance, I want to play this game again	3, 5, 5, 6, 6, 7	2, 4, 4, 5, 6, 6	2, 3, 5, 6, 7, 7
I feel the game's audio (e.g., sound effects, music) enhances my gaming experience	4, 6, 6, 7, 7	4, 4, 5, 5, 6, 7	3, 4, 4, 5, 5, 7
I find the game's menus to be user friendly	5, 5, 5, 6, 7, 7	3, 5, 5, 5, 6, 7	4, 5, 5, 6, 7, 7
I feel the game allows me to express myself	2, 3, 4, 4, 4, 5	2, 4, 4, 4, 4, 5	2, 2, 3, 3, 3, 4
I feel the game trains me well in all of the controls	4, 4, 5, 5, 5, 6	2, 3, 4, 5, 5, 5	2, 2, 3, 3, 4, 4
I am very focused on my own performance while playing the game	4, 5, 6, 6, 6, 6	3, 5, 5, 6, 6, 7	2, 4, 5, 5, 5, 6

Tabell A.6: GUESS svar del 3/3

Påstående	Prototypgrupp	Vardagsgrupp 1	Vardagsgrupp2
I do not care to check events that are happening in the real world during the game	1, 1, 3, 5, 6, 6	2, 2, 2, 6, 6, 7	1, 2, 2, 3, 6
I always know my next goal when I finish an event in the game	3, 4, 5, 5, 6, 7	2, 4, 4, 5, 5, 6	3, 4, 4, 6, 6
I am likely to recommend this game to others	1, 4, 5, 6, 7, 7	2, 4, 5, 5, 6, 7	1, 2, 5, 5, 5, 7
I think the game is visually appealing	3, 4, 5, 6, 6, 7	4, 4, 5, 5, 6, 6	3, 3, 5, 5, 6, 6
I feel my curiosity is stimulated as the result of playing the game	2, 4, 4, 5, 5, 5	4, 4, 4, 5, 5, 5	1, 3, 3, 4, 5
Sometimes I lose track of time while playing the game	1, 4, 4, 5, 6	2, 3, 4, 4, 5, 6	1, 1, 3, 4, 4
I feel the game provides me the necessary information to accomplish a goal within the game	4, 6, 6, 6, 7, 7	2, 2, 5, 6, 6, 7	2, 2, 3, 6, 6
I think the graphics of the game fits the mood or style of the game	4, 5, 5, 6, 6, 7	5, 5, 5, 6, 6, 6	4, 5, 5, 5, 7, 7
I think the game is unique or original	4, 4, 5, 5, 5, 6	5, 5, 6, 6, 6, 7	2, 4, 4, 5, 6, 7
I enjoy the music in the game	4, 5, 6, 7, 7	3, 3, 5, 5, 5, 7	2, 3, 4, 5, 5, 7
I feel very confident while playing the game	4, 5, 5, 5, 5, 7	1, 3, 3, 5, 5, 7	2, 3, 4, 4, 4, 7
I enjoy playing the game	5, 5, 6, 6, 6, 7	2, 4, 5, 5, 5, 6	2, 2, 5, 5, 6, 7
I temporarily forget about my everyday worries while playing the game	2, 4, 4, 5, 5, 5	2, 2, 4, 6, 6, 7	1, 3, 4, 5, 6
I think the information provided in the game (e.g, onscreen messages, help) is clear	2, 3, 4, 5, 6, 6	2, 3, 4, 6, 6, 6	1, 3, 5, 5, 5, 5
I feel the game gives me enough freedom to act how I want	3, 4, 4, 4, 5, 6	3, 3, 4, 5, 5, 5	2, 3, 4, 5, 5, 5
I want to do as well as possible during the game	5, 5, 6, 6, 6, 7	4, 5, 5, 6, 7, 7	1, 5, 6, 6, 7, 7

B Enkätsvar vid prototyputveckling

B.1 Förbättringsförslag efter förstudie

Har du något förslag på hur rörelsen av spelarkaraktären kan förbättras?

- D-pad verkar vettigt. Fingret hamnade ofta över karaktären för mig och jag hade lite svårt med precision i rörelser.
- D-pad skulle vara bättre. Det var krångligt att gå samtidigt som man vill placera ut turrets
- Problemet nu är att tummen hamnar över skeppet, så det blir svårare att parera fiendens skott samt att placera turrets. En virituell D-pad skulle fungera, alternativt att skeppet inte hamnar precis där tummen trycker, utan lite mer åt sidan
- D-pad eventuellt, men svårt att säga om det blir bättre. Problemet nu är att tummen döljer en stor del av skärmen beroende på hur man behöver röra sig

Har du något förslag på hur placeringen av byggnader skulle kunna bli bättre?

- Markera alla placerbara rutor direkt, inte bara den man försöker placera på. Antagligen två tryck istället för att dra. Någon tooltip skulle kanske behövas
- Ett alternativ är ju att låta spelaren placera byggnader var som helst, oavsett var man befinner sig
- Visa det gröna någon annanstans än vid fingret
- Det gick inte att placera turrets på samma ruta som spelaren stod på, fastän spelaren bara knappt nuddade rutan. Det borde vara en mindre marginal för var man kan placera ut turrets nära spelaren
- Just nu är det en cirkel som visar området där man kan placera turrets, detta blir inte så exakt eftersom placeringarna utgår ifrån ett rutnät, skulle möjligtvis vara bättre om det var en rektangel som följde rutnätet istället
- Det är svårt att se var man placerar turrets för tummen döljer det. Kanske att en turret placeras där man står när man trycker på den

Har du något förslag på nya byggnader eller magier? Tycker du att någon/några bör bytas ut?

- Det verkade oklart för mig vad som sköt på andra turrets och vad som gick igenom. Såg aldrig om det gick att se turrets hp. Vet inte heller vad som gör att reflektorerna förstörs. En melee-gubbe kan finnas, som går från där den placeras rakt över till andra sidan och slår på saker
- Inget förslag på nya turrets, de som finns nu känns bra
- Jag förstod inte riktigt skillnaden mellan de blåa och de röda. Annars var de röda en bra kombo tycker jag
- Var en ganska bra fördelning mellan olika typer av turrets/spells, vill man göra det lite mer avancerat senare så skulle man kunna ha olika typer av decks, där spelaren själv får välja vilka typer av turrets som ska kunna användas under matchen
- Nej inte direkt. Kanske en turret som skjuter mot motspelarens riktning lagom långsamt så att denna måste förflytta sig.

Vad tycker du om antalet byggnader/magier?

- Ok
- Känns lagom. För mycket hade bara gjort att man inte vet vad man ska bygga
- Lite för många kanske
- Lagom
- Lagom, ska inte vara för mycket i ett sånt här typ av fast 1v1 battle-spel

Tycker du spelet borde gå snabbare eller långsammare?

- Verkar vara vettigt tempo
- Beror väl på hur pass duktig man är på spelet. Som nybörjare känns spelets hastighet lagom
- Jag tycker det var ett tempo som kändes bra
- Verkade vara ganska lagom hastighet, men man vill kanske ha snabbare när man väl blivit bättre, så möjligtvis om man själv kan ställa in hastigheten
- Jag tycker nog hastigheten var bra. Eventuellt att man skulle tycka att det ska gå snabbare om man har kört det mycket och är duktig på spelet

Tycker du spelet borde vara indelat i olika faser, exempelvis en byggfas och en spelfas, i så fall hur tycker du det skulle fungera?

- Det tycker jag inte
- Att bygga och samtidigt ha koll på allt annat, gör att man ibland har svårt att bygga. En möjlighet skulle vara att spelaren/spelarna får 15 sekunder i början att bygga innan någon kan skjuta. Alternativt ett nytt spelläge, där man får tid att bygga byggnader för en startsumma och kan inte bygga fler när spelet startar
- Kan vara ett battle mode. Men det vanliga bör vara som nu

- Jag tror det är bättre att allt sker på en gång. Det blir lite svårare då
- Tyckte det var bra som det är, kanske att man inte ska kunna skjuta något de första 10 sekunderna, så får spelarna en större chans att börja bygga
- Nej, jag ser det mer som ett snabbt 1v1 battle spel där man snarare spelar flera battles än att spela långa omgångar

Har du några övriga förbättringsförslag?

- Interaktion mellan saker eller spelaren, knappar som lyser till när man klickar på dom, animationer när saker träffar saker. Option-menyn bör sticka ut mer från resten av knapparna. Telefonens bakåtknapp fungerade inte, fick klicka runt tills nån knappt synlig B knapp dök upp
- Skulle gärna se att skotten är mer tydliga för att lättare se dem. Byta färg kanske. Sedan känns placeringen av där man väljer byggnader mindre bra. Om man använder vänster tumme för att styra gubben vill man inte dra ner den för att ta fram en byggnad och göra sig sårbar. Vore bättre om de placerades till höger. Detta blir väl tvärtom för spelare 2
- Det var svårt att hänga med i poängsystemet - mana, liv, pengar osv. Det kanske skulle räcka med en eller två resurstyper? Särskilt på en så liten skärm är det svårt att hålla koll på hur det ser ut med pengar när man redan håller fingrarna överallt
- När jag spelade multiplayer så hamnade jag på höger sida, fast i tutorialen hade jag tränat på vänster, så det blev tvärtom för tummarna. Skulle föredra om man alltid är på en viss sida, eller att man kan välja själv. En annan sak var friendly fire, skulle hellre ha det avstängt
- Tydligare varningar när man börjar få dåligt med liv eller blir träffad. T.ex. att skärmen eller liv-siffrorna blir rött när man blir träffad och att skärmen fadar mot rött när livet nästan är slut. En nedräkning innan en runda börjar. Lite schysst end-game grafik eller animationer

B.2 Bedömning av förbättringsförslag

Denna enkät gjordes från de förbättringsförslag som prototypgruppen gav, även några förslag från utvecklarna finns med. De svarande har inte testat någon av prototyperna, svaren användes endast för att prioritera förbättringsförslagen.

Tabell B.1: Bedömning av förbättringsförslag

Förbättringsförslag	Betyg
Användning av D-pad för styrning av karaktär	-3, 0, 2, 3, 5, 5
Karaktären går mot strax framför fingret istället för direkt på	0, 1, 2, 2, 3, 5
För att placera byggnader så trycker man först en gång på den byggnadsknapp man vill placera, då markeras alla rutor som det går att placera på, sedan trycker man på en av de markerade rutorna och då placeras byggnaden där	2, 3, 3, 3, 4, 4
För att placera byggnader så drar man byggnadsknappen åt det håll man vill placera (exempelvis om man drar rakt uppåt så placeras byggnaden framför karaktären om den rutan är ledig)	-3, -2, -1, 0, 2, 5
Spelaren kan placera byggnader var som helst på sin planhalva oavsett hur långt ifrån man står	-5, -2, -2, -1, 0, 4
När man trycker på en byggnadsknapp så placeras den byggnaden direkt framför karaktären om den rutan är ledig	-2, 0, 1, 2, 2, 3
Innan varje spel så får spelaren själv välja ut de (9) byggnader/magier bland många fler som ska gå att använda i det spelet	0, 1, 2, 3, 4, 5
Det finns olika sorts karaktärer (klasser) som alla har olika sorts uppsättningar av byggnader och magier	-2, 0, 1, 2, 3, 4
Byggnader byggda av samma spelare kan inte skada varandra	-2, -2, 0, 2, 4, 5
Minska antalet byggnader/magier till sju stycken (från nio)	-2, -1, -1, 0, 0, 3
Ta bort sköld-egenskapen, det räcker med liv	-2, -1, 0, 3, 3, 4
Ta bort mana-egenskapen, gör så att magier kostar pengar istället	-3, -2, -1, 1, 2, 3
Ta bort mana-egenskapen, gör så att magier är gratis men med en nedkylningstid istället	-2, -2, 0, 2, 3, 3
Spelarna får själva välja vilken sida de vill spela på	0, 0, 1, 5, 5, 5

B.3 Första prototyputvärderingen

B.3.1 Påverkan av spelupplevelse

Tabell B.2: Prototyputvärdering 1 - Påverkan av spelupplevelse

Prototyp	Påverkan av spelupplevelse
Tvåtrycksplacering	-3, 2, 3, 3, 3, 5
Direktplacering	1, 2, 3, 3, 4, 5
Fingerförflyttning version 2 (karaktären går till strax framför fingret istället för direkt under)	3, 3, 3, 4, 5, 5
Styrkorsförflyttning	-3, -2, 0, 0, 5, 5
Magikostnad version 2 (Magier är gratis men har en nedkylningstid istället)	2, 2, 3, 3, 3, 4
Byggnader skjuter igenom andra byggnader skapade av samma spelare	0, 1, 2, 2, 3, 5
Spelarna får själva välja vilken spelhalva de vill spela på	0, 1, 2, 3, 3, 5

B.3.2 Förbättringsförslag på prototyperna

Tvåtrycksplacering

- Kanske att fler gröna rutor blir tillgängliga att placera på
- Större knappar för byggnader etc
- Möjligtvis att vilken ruta även väljs där man väljer torn, exempelvis att rutorna hamnar runt tornets ikon istället för skeppet. Kan vara lite svårt att placera ibland som det är nu

Direktplacering

- Om det redan finns ett objekt framför spelaren placeras turret bredvid i clockwise ordning runt spelaren
- Om rutan precis framför redan är upptagen så skulle tornet kunna placeras på någon annan närliggande ruta istället

Styrkorsförflyttning

- Att den hela tiden syns påverkar mycket. Då ser man inte motståndaren ibland. Den borde döljas på något sätt
- Större D-pad vore att föredra för mig
- Borde få välja dpad eller ej
- Gör d-paden lite genomskinlig så man ser saker bakom den

B.4 Andra prototyputvärderingen

Tabell B.3: Prototyputvärdering 2 - Påverkan av spelupplevelse

Prototyp	Påverkan av spelupplevelse
Styrkorsförflyttning version 2 (Större och genomskinlig)	1, 3, 3, 4, 4, 5
Direktplacering version 2 (Om rutan framför är upptagen så placeras tornet på någon annan ledig ruta runt karaktären)	-2, 0, 3, 3, 3, 5

Vilken typ av byggnadsplacering föredrar du?

- Direktplacering - 6 st
- Tvåtrycksplacering - 0 st

Vilken typ av karaktärsförflyttning föredrar du?

- Styrkorsförflyttning - 4 st
- Fingerförflyttning - 2 st

C Prestandakod

C.1 Kollisionsdetektion

C.1.1 Kollisionshanterare

```
from time import time
from util.singleton import Singleton

from set_strategy_handler import SetStrategyHandler
from sort_and_sweep import SortAndSweep
from quad_tree import QuadTree

from game_settings import GameSettings


@Singleton
class CollisionDetecter(object):
    USE_COLLISION_SETS = True
    USE_SORT_AND_SWEEP = False
    USE_QUAD_TREE = False

    def __init__(self):
        self.collision_objects = {}
        self.set_strategy_handler = SetStrategyHandler.get_instance()
        self.sort_and_sweep = SortAndSweep({}, horizontal=True,
                                           sort_algorithm='insertion_sort')
        self.quad_tree = QuadTree(1024, 512, 2)

    def handle_collisions(self):
        def _handle_collisions(iterable):
            for obj1, obj2 in iterable:
                if self.collide_object(obj1, obj2):
                    obj1.on_collision(obj2)
                    obj2.on_collision(obj1)

        start_time = time()
        if self.USE_COLLISION_SETS:
            _handle_collisions(self.set_strategy_handler.tests_generator())

        if self.USE_SORT_AND_SWEEP:
            _handle_collisions(self.sort_and_sweep.sort_and_sweep())
```

```
        if self.USE_QUAD_TREE:
            self.quad_tree.update()
            _handle_collisions(self.quad_tree.tests_generator())

        if not self.USE_COLLISION_SETS and not self.USE_SORT_AND_SWEEP \
                and not self.USE_QUAD_TREE:
            objects = self.collision_objects.values()
            for i in range(len(objects)-1):
                for j in range(i+1, len(objects)):
                    obj1 = objects[i]
                    obj2 = objects[j]
                    if self.collide_object(obj1, obj2):
                        obj1.on_collision(obj2)
                        obj2.on_collision(obj1)
        self.collision_time = time() - start_time

    def add_object(self, obj):
        if self.USE_COLLISION_SETS:
            obj.set_strategy.add_object(obj)
        elif self.USE_SORT_AND_SWEEP:
            self.sort_and_sweep.add_object(obj)
        elif self.USE_QUAD_TREE:
            self.quad_tree.add_object(obj)
        else:
            self.collision_objects[obj.id] = obj

    def remove_object(self, obj):
        if self.USE_COLLISION_SETS:
            obj.set_strategy.remove_object(obj.id)
        elif self.USE_SORT_AND_SWEEP:
            self.sort_and_sweep.remove_object(obj)
        elif self.USE_QUAD_TREE:
            self.quad_tree.remove_object(obj)
        else:
            del self.collision_objects[obj.id]

    def collide_object(self, obj1, obj2):
        obj1.collision_x = obj1.x + (obj1.width - obj1.collision_width)/2
        obj1.collision_y = obj1.y + (obj1.height - obj1.collision_height)/2
        obj2.collision_x = obj2.x + (obj2.width - obj2.collision_width)/2
        obj2.collision_y = obj2.y + (obj2.height - obj2.collision_height)/2
        if self._is_bounding_box_colliding(obj1, obj2):
            return self._is_colliding(obj1, obj2)
        return False

    def _is_bounding_box_colliding(self, obj1, obj2):
        if obj1.collision_x + obj1.collision_width < obj2.collision_x:
            return False
        if obj1.collision_x > obj2.collision_x + obj2.collision_width:
            return False
        if obj1.collision_y + obj1.collision_height < obj2.collision_y:
            return False
```

```
        if obj1.collision_y > obj2.collision_y + obj2.collision_height:
            return False
        return True

    def _is_colliding(self, obj1, obj2):
        up, down, left, right = None, None, None, None
        if obj1.collision_x < obj2.collision_x:
            left = obj1
            right = obj2
        else:
            left = obj2
            right = obj1
        if obj1.collision_y < obj2.collision_y:
            up = obj2
            down = obj1
        else:
            up = obj1
            down = obj2

        for y in range(int(up.collision_y),
                       min(int(down.collision_y + down.collision_height),
                           int(up.collision_y + up.collision_height))):
            for x in range(int(right.collision_x),
                           min(int(left.collision_x + left.collision_width),
                               int(right.collision_x + right.collision_width))):
                if obj1.MASK[y-int(obj1.collision_y)][x-int(obj1.collision_x)] \
                        and obj2.MASK[y-int(obj2.collision_y)]\
                        [x-int(obj2.collision_x)]:
                    return True
        return False
```

C.1.2 Sort and sweep

```
class SortObject(object):
    def __init__(self, obj, start, horizontal):
        self.obj = obj
        self.start = start
        self.horizontal = horizontal
        self.update_positions()
        if start:
            obj.sort_start = self
        else:
            obj.sort_stop = self

    def update_positions(self):
        if self.start:
            if self.horizontal:
                self.pos = self.obj.x
            else:
                self.pos = self.obj.y
        else:
            if self.horizontal:
                self.pos = self.obj.x + self.obj.width
```

```
            else:
                self.pos = self.obj.y + self.obj.height


class SortAndSweep(object):
    def __init__(self, objects, horizontal=True, sort_algorithm='timsort'):
        self.sort_objects = []
        self.sort_algorithm = sort_algorithm
        for obj in objects.values():
            self.add_object(obj)
        self.horizontal = horizontal

    def sort_and_sweep(self):
        self.update_positions()
        self.sort_positions()
        self.active_objects = set([])
        for sort_obj in self.sort_objects:
            if sort_obj.start:
                for obj2 in self.active_objects:
                    yield sort_obj.obj, obj2
                self.active_objects.add(sort_obj.obj)
            else:
               self.active_objects.remove(sort_obj.obj)

    def update_positions(self):
        for sort_obj in self.sort_objects:
            sort_obj.update_positions()

    def sort_positions(self):
        if self.sort_algorithm == 'timsort':
            self.sort_objects.sort(key=self.sort_function)
        elif self.sort_algorithm == 'insertion_sort':
            self.insertion_sort(self.sort_objects)
        elif self.sort_algorithm == 'quicksort':
            self.quicksort(self.sort_objects)

    def insertion_sort(self, lst):
        for index in range(1, len(lst)):
            current_obj = lst[index]
            position = index
            while position > 0 and lst[position-1].pos > current_obj.pos:
                lst[position] = lst[position-1]
                position = position-1

            lst[position] = current_obj

    def add_object(self, obj):
        self.sort_objects.append(SortObject(obj, True, self.horizontal))
        self.sort_objects.append(SortObject(obj, False, self.horizontal))

    def remove_object(self, obj):
        self.sort_objects.remove(obj.sort_stop)
        self.sort_objects.remove(obj.sort_start)
```

```
    def sort_function(self, sort_object):
        return sort_object.pos

    def quicksort(self, array, begin=0, end=None):
        def partition(array, begin, end):
            pivot = begin
            for i in xrange(begin+1, end+1):
                if array[i].pos <= array[begin].pos:
                    pivot += 1
                    array[i], array[pivot] = array[pivot], array[i]
            array[pivot], array[begin] = array[begin], array[pivot]
            return pivot

        if end is None:
            end = len(array) - 1
        def _quicksort(array, begin, end):
            if begin >= end:
                return
            pivot = partition(array, begin, end)
            _quicksort(array, begin, pivot-1)
            _quicksort(array, pivot+1, end)
        return _quicksort(array, begin, end)
```

C.1.3 Quadtree

```
class QuadTree(object):
    def __init__(self, width, height, depth, x=-10, y=-10, parent=None):
        self.objects = []
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.parent = parent
        if depth > 0:
            children_size = width//2, height//2
            children_positions = [(0, 0), (width//2, 0), (0, height//2),
                                  (width//2, height//2)]
            self.children = [QuadTree(children_size[0], children_size[1],
                                      depth-1, children_positions[i][0],
                                      children_positions[i][1], self)
                             for i in range(4)]
        else:
            self.children = []

    def tests_generator(self):
        for i in range(len(self.objects)):
            for j in range(i+1, len(self.objects)):
                obj1, obj2 = self.objects[i], self.objects[j]
                yield obj1, obj2

        for obj in self.objects:
            for child in self.children:
```

```
                for obj2 in child.get_all_objects():
                    yield obj, obj2

        for child in self.children:
            for collision_pair in child.tests_generator():
                yield collision_pair

    def get_all_objects(self):
        for obj in self.objects:
            yield obj
        for child in self.children:
            for obj in child.get_all_objects():
                yield obj

    def fits(self, obj):
        return not(obj.x < self.x or obj.y < self.y or
                   self.x + self.width < obj.x + obj.width or
                   self.y + self.height < obj.y + obj.height)

    def push_down(self, obj):
        for child in self.children:
            if child.add_object(obj):
                return True
        return False

    def push_up(self, obj):
        node = self.parent
        while not node.add_object(obj):
            node = node.parent

    def add_object(self, obj):
        if not self.fits(obj):
            return False

        if not self.push_down(obj):
            self.objects.append(obj)
        return True

    def remove_object(self, obj):
        if not self.fits(obj):
            return False

        for child in self.children:
            if child.remove_object(obj):
                return True
        try:
            self.objects.remove(obj)
        except:
            print obj
            print self.objects + self.get_all_objects()
        return True

    def update(self):
```

```
        for child in self.children:
            child.update()
        old_objects = self.objects
        self.objects = []
        for obj in old_objects:
            if not self.fits(obj):
                self.push_up(obj)
            elif not self.push_down(obj):
                self.objects.append(obj)
```

C.1.4 Set strategy

```
from set_strategy_handler import SetStrategyHandler


class SetStrategy(object):
    def __init__(self):
        self.sh = SetStrategyHandler.get_instance()

    def add_object(self, obj):
        pass

    def remove_object(self, obj):
        pass


class MovingCollider(SetStrategy):
    def add_object(self, obj):
        self.sh.moving_colliders[obj.id] = obj

    def remove_object(self, id):
        del self.sh.moving_colliders[id]


class MovingCollidee(SetStrategy):
    def add_object(self, obj):
        self.sh.moving_collidees[obj.id] = obj

    def remove_object(self, id):
        del self.sh.moving_collidees[id]


class StillCollider(SetStrategy):
    def add_object(self, obj):
        self.sh.still_colliders[obj.id] = obj

    def remove_object(self, id):
        del self.sh.still_colliders[id]


class StillCollidee(SetStrategy):
    def add_object(self, obj):
        self.sh.still_collidees[obj.id] = obj
```

```
    def remove_object(self, id):
        del self.sh.still_collidees[id]
```

Set strategy handler

```
from util.singleton import Singleton


@Singleton
class SetStrategyHandler(object):
    def __init__(self):
        self.moving_colliders = {}
        self.moving_collidees = {}
        self.still_colliders = {}
        self.still_collidees = {}

    def tests_generator(self):
        moving_collider_list = self.moving_colliders.values()
        moving_collidee_list = self.moving_collidees.values()
        still_collider_list = self.still_colliders.values()
        still_collidee_list = self.still_collidees.values()

        for i in range(len(moving_collider_list)):
            for j in range(i+1, len(moving_collider_list)):
                yield moving_collider_list[i], moving_collider_list[j]
            for j in range(len(moving_collidee_list)):
                yield moving_collider_list[i], moving_collidee_list[j]
            for j in range(len(still_collider_list)):
                yield moving_collider_list[i], still_collider_list[j]
            for j in range(len(still_collidee_list)):
                yield moving_collider_list[i], still_collidee_list[j]

        for i in range(len(still_collider_list)):
            for j in range(len(moving_collidee_list)):
                yield still_collider_list[i], moving_collidee_list[j]

    def get_all_objects(self):
        return self.moving_colliders.values() + self.moving_collidees.values() \
                + self.still_colliders.values() + self.still_collidees.values()
```

C.2 Bildblad

C.2.1 Animation creator

```
from kivy.atlas import Atlas

from util.singleton import Singleton
from animation import Animation


@Singleton
class AnimationCreator(object):
```

```
    def __init__(self, obj):
        obj_cls = obj.__class__
        if obj_cls.ATLASES is None:
            atlases = {}
            if obj.orig_animation is not None:
                atlases['orig_animation'] = Atlas(obj.orig_animation)
            if obj.shoot_animation is not None:
                atlases['shoot_animation'] = Atlas(obj.shoot_animation)
            if obj.init_animation is not None:
                atlases['init_animation'] = Atlas(obj.init_animation)
            if obj.destroy_animation is not None:
                atlases['destroy_animation'] = Atlas(obj.destroy_animation)
            obj.__class__.ATLASES = atlases

        if obj.orig_animation is not None:
            obj.animation_orig = Animation(obj_cls.ATLASES['orig_animation'],
                                           obj, loop=True)
            if obj.init_animation is None:
                obj.animation_orig.play()

        if obj.shoot_animation is not None:
            obj.animation_shoot = Animation(obj_cls.ATLASES['shoot_animation'],
                                            obj)

        if obj.init_animation is not None:
            obj.animation_init = Animation(obj_cls.ATLASES['init_animation'],
                                           obj)
            obj.animation_init.play()
```

C.2.2 Animation

```
from kivy.clock import Clock
from kivy.atlas import Atlas


class Animation(object):
    def orig_source(self):
        if self.orig_animation is not None:
            self.animation_orig.play()
        else:
            self.source = self.orig_source

    def __init__(self, atlas, obj, anim_delay=0.04, loop=False,
                 anim_end=orig_source):
        self.atlas = atlas
        self.obj = obj
        self.anim_delay = anim_delay
        self.anim_end = anim_end
        self.loop = loop

        self.img_count = 0
        self.nr_imgs = len(self.atlas.textures.keys())
```

```
    def play(self):
        self.next_img()
        func = (lambda x: self.next_img()) if self.loop else self._play_once
        self.schedule = Clock.schedule_interval(func, self.anim_delay)

    def _play_once(self, dt):
        if self.img_count == self.nr_imgs:
            Clock.unschedule(self.schedule)
            self.anim_end(self.obj)
        else:
            self.next_img()

    def next_img(self):
        if self.img_count == self.nr_imgs:
            self.img_count = 0
        self.img_count += 1
        self.update_source()

    def update_source(self):
        self.obj.texture = self.atlas[str(self.img_count).zfill(2)]
```

C.3 Objektpool

Användningen av objektpooler implementerades i flera olika filer, *object_pool.py* som innehåller en statisk metod för skapande av själva poolerna. Fabriksmetoden användes för skapande av objekt i projektet, därav hamnade aktiveringen av objekt i objektpooler i dessa filer och inaktiveringen av objekt hamnade i den abstrakta klassen *GameObject*. De relevanta delarna från de olika filerna finns med i detta avsnitt.

C.3.1 object_pool.py

```
from collections import deque


class ObjectPool:
    OBJECTS_QUEUE = {'SmallBullet': deque([])}
    INACTIVE_POS = [1000, 400]

    @staticmethod
    def create_object_pools():
        import model.game_objects.damage_objects.projectiles as projectiles
        import model.game_objects.damage_objects.special_damage as special_dmg
        import model.game_objects.spells.shield as shield
        import model.game_objects.turrets.turrets as turrets
        import model.ui.placement_square as placement_square
        classes = [projectiles.SmallBullet, projectiles.FreezeBullet,
                   projectiles.ExplosionProjectile, special_dmg.Flame,
                   special_dmg.Laser, shield.Shield, turrets.RotatingTurret,
                   turrets.RotatingPart, turrets.FlameTurret,
                   turrets.MortarTurret, turrets.FreezeTurret,
                   turrets.BumperTurret, turrets.MoneyTurret,
                   placement_square.PlacementSquare, turrets.Block]
```

```
        for class_ in classes:
            class_.create_object_pool(class_)
```

C.3.2 Fabriker

```
@staticmethod
def activate(class_, pos, owner, speed, direction, damage):
    obj = class_.OBJECT_POOL.popleft()
    obj.owner = owner
    obj.pos = pos
    obj.direction = direction
    obj.speed = speed
    obj.damage = damage
    obj.active = True
    return obj
```

C.3.3 GameObject

```
def deactivate(self):
        self.active = False
        self.pos = self.INACTIVE_POS
        self.speed = 0
        self.active = False
        self.__class__.OBJECT_POOL.append(self)
        Globals.deleted_game_objects.append(self.id)

    @staticmethod
    def create_object_pool(class_):
        class_.OBJECT_POOL = deque([class_(class_.INACTIVE_POS, speed=0)
                                    for x in range(class_.OBJECT_POOL_CAP)])
```