CHALMERS  GÖTEBORGS UNIVERSITET

Vengeful Vikings

Ett strategispel till Android

Kandidatarbete inom Data- och informationsteknik

Andreas Eklund
Anton Grönlund
Erik Termander
Johnny Väyrynen
Kalle Persson
Tomas Urdell

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg, Sverige 2012

Kandidatarbete nr 2012:20

Sammanfattning

Rapporten behandlar utvecklandet av en spelidé för ett turordningsbaserat strategispel med fokus på flerspelarläge över nätverk, samt hur idén implementeras till plattformen Android. Rapporten kommer fortsättningvis även att dokumentera och beskriva hur arbetet med att uppnå syftet att utveckla ett turordningsbaserat strategispel uppfylls. Utöver detta kommer rapporten dessutom att behandla studier av liknande spel för olika plattformar. Utvecklingen i detta projekt sker genom en agil arbetsprocess som kommer att beskrivas samt diskuteras. Resultatet av projektet är en fungerande betaversion av Vengeful Vikings, ett turordningsbaserat strategispel unikt anpassat för att användas på Android.

Nyckelord: android, strategispel, spelutveckling, agile

Abstract

This project aims to depict the process of developing an idea of a turn-based strategy game focused on multiplayer network play and implementing it for the Android platform. The report shows the progress over time, exhibiting the different development stages and their significance towards the main purpose. Part of the research consists of deriving inspiration from studies of similar games. We then proceed to develop the actual game using an agile development process, which is further elaborated in the report. The final result is a fully functional beta release of Vengeful Vikings, a strategy game specifically tailored for Android, although the concept is also functional for smartphones in general.

Keywords: android, strategy game, game development, agile

Innehåll

1

Introduktion

Kandidatgruppen som driver detta projekt består av sex studenter som tillsammans ska utveckla ett turordningsbaserat strategispel till mobilplattformen Android. Utvecklingen kommer att ha fokus på inlärning för att använda plattformen, samt presentera goda tekniska lösningar inom områden såsom AI, nätverk och spel- & grafikmotor. Utvecklingen kommer att ske iterationsbaserat och projektgruppen ämnar att försöka använda och utnyttja arbetssätt som används inom mjukvaruutveckling i industrin.

Rapporten beskriver projektets arbetsgång och fokuserar på de olika implementerings- och designval gruppen gör för implementering och design. Dessa val beskrivs samt motiveras. Vidare beskriver rapporten huruvida dessa val är, eller skulle ha varit, positiva för projektet.

Under inledningsfasen inser gruppen behovet av att kunna referera till projektet av olika anledningar. Därför antas tidigt projektnamnet Anstrat, som står för "**An**droid **Strat**egy game". Det är dock inte detsamma som namnet på spelet som under projektet utkristalliserar sig till Vengeful Vikings.

Projektgruppen vill prova på att utveckla till Android då medlemmarna anser att det är en intressant och spännande plattform. Dessutom är applikationer till smartphones en del av en expanderande marknad, med Android som den största smartphone-plattformen just nu. Detta innebär att om en plattform måste väljas når applikationen flest användare om den utvecklas till Android. Dessutom ses det som en fördel att Android inte är lika låst till ett företag på det sätt som Apples iOS är, där applikationerna för iOS på grund av sitt format är låsta att enbart kunna köras på Apples egna produkter.

Från gruppens perspektiv är Android ett betydligt bättre alternativ eftersom alla gruppmedlemmar redan äger en Android-telefon. Dessutom sker utveckling till Android i första hand i Java till skillnad mot Objective-C för iOS. Java är något som alla i projektgruppen sedan tidigare har erfarenhet av, och det språk som hittills använts mest i deras respektive utbildningar. Ingen i projektgruppen har någon erfarenhet av Objective-C, och ingen äger heller någon av de Apple-produkter som krävs för utveckling

till iOS (se 1.1.3).

De flesta som äger Androidtelefoner bär med sig dem mer eller mindre överallt vilket gör att tillgängligheten hos användaren är väldigt stor. Androidtelefonerna har nått en väldig popularitet och applikationer som idag släpps till dessa telefoner når ut till en stor grupp människor. Samtidigt anser projektgruppen att utbudet på applikationer inte följt övrig utvecklings framsteg och att det därför fortfarande finns en stor efterfrågan, av framförallt spel. Gruppen anser även att utbudet för just strategispel är särskilt undermåligt och i skrivande stund finns det inte ens en egen kategori för strategispel på Google Play.

Samtliga gruppmedlemmars genuina intresse för strategispel har också bidragit till valet av ämne. Tack vare detta intresse har gruppen vissa förkunskaper och idéer om vilka element som konstituerar grunden för ett lyckat strategispel.

1.1 Bakgrund

För att skapa förståelse för projektet så kommer bakgrunden inledningsvis redogöra för begreppet strategispel, därefter beskriva strategispel som redag idag existerar och detta projekt tar inspiration ifrån, för att slutligen jämföra smartphone-plattformarna Android och iOS med varandra.

1.1.1 Strategispel

Innebörden av ordet strategispel är inte allmänt definierat. I detta projekt definieras ett strategispel som ett spel där deltagarna använder strategiskt tänkande för att uppnå spelets mål. Ofta finns det olika sätt att uppnå detta mål och det är upp till deltagarna själva att avgöra vilken strategi som är den bästa.

Strategispel till datorer är antingen turordningsbaserade eller realtidsbaserade. Om spelet är turordningsbaserat innebär det, likt många brädspel, att spelarna alternerar om att göra sina drag. Om spelet istället är realtidsbaserat så sker alla handlingar som spelarna gör samtidigt. Av dessa två spelvarianterna introducerades turordningsbaserade spel först på marknaden, med spelet *Empire* som släpptes på 1970-talet [1].

Vanligt förekommande mål i turordningsbaserade strategispel är att förstöra motståndarens bas eller inta någon nyckelposition på spelets spelplan. Det existerar idag en rad olika strategispel med olika spelkoncept på marknaden.

1.1.2 Exempel på kända strategispel

Strategispel finns på en mängd olika plattformar, från PC till Android och *Game Boy*. Den största och vanligaste plattformen för strategispel är PC där det finns många olika strategispel som blivit storsäljare. Inom turordningsbaserad strategi återfinns bland annat den berömda *Civilization*-serien [2], med spel som sålts i miljontals exemplar. Nedan följer en lista med exempel på strategispel för PC.

Battle for Westnoth
Gratisspelet *Battle for Wesnoth* är ett turordningsbaserat strategispel till PC som släpptes 2005 [3]. Det utspelar sig i fantasy-miljö och har en spelplan bestående av hexagoner. Spelaren får inkomst av att ta över och hålla byar. Enheter rekryteras ifrån en borg som spelaren kontrollerar.

Civilization-serien
En serie turordningsbaserade strategispel till PC skapat av Sid Meier 1991 är *Civilization*-serien [4] som nu består av fem spel och blivit ett av världens mest kända strategispel till PC. Det finns flera olika mål för spelaren och vinnaren är den deltagare som först uppfyller kriterierna för ett av målen. Målen har olika fokus och utöver det militära målet där man tar över alla motståndarnas huvudstäder, kan man även vinna genom diplomati, kultur, eller teknologi.

PoxNora
PoxNora är ett turordningsbaserat strategispel som spelas i flerspelarläge över internet och som släpptes på marknaden 2006 [5] till PC. Spelet i sig är gratis, men användarna kan betala för att få tillgång till bättre enheter. Spelet utspelar sig i fantasymiljö.

Advance Wars
Advance Wars är en serie med ett flertal turordningsbaserade strategispel utvecklade till Nintendos bärbara konsoler. Det första spelet, *Famicom Wars*, kom redan 1988 [6] men spelen släpptes inte utanför Japan förrän *Advance Wars* år 2001 [7]. Likt *Battle for Wesnoth* får man sin inkomst genom att ta över byggnader på spelets karta.

Battle for Mars
Battle for Mars är ett strategispel till Android, som släpptes år 2009 [8] och är tydligt influerat av *Advance Wars*-spelen. Som namnet antyder utspelar det sig i en miljö på Mars, komplett med futuristiska robotar.

1.1.3 Android vs iOS

När det gäller utveckling till smartphones finns det två dominerande plattformar att utveckla till, nämligen Googles Android och Apples iOS. Under 2011 såldes 473 miljoner smartphones världen över [9, 10, 11, 12]. För dessa smartphones var iOS och Android de två vanligaste operativsystemen i det fjärde kvartalet år 2011, med 51% respektive 24% av marknaden [12].

För att köra utvecklingsmiljön till iOS krävs en Mac-dator, medan utvecklingsmiljön för Android går att få tag på helt gratis till Windows, Mac och Linux, utan extra kostnader. När det gäller iOS så krävs det att man går med i *iOS Developer Program* om man vill testa sin iPhone-app på en fysisk enhet eller kunna ladda ned deras iOS-simulator. Detta till en kostnad av $99 per år [13].

En applikation som är gjord för iOS kan, till skillnad från Android, enbart köras på Apples egna produkter. Det finns många olika telefontillverkare som använder Android som operativsystem på sina telefoner. Det gör att det är stor variation i prestanda på telefoner användandes Android. Eftersom det är ett betydligt mindre antal modeller som kör iOS, är variationen i begränsningar mindre när man utvecklar mot iOS, jämfört med när man utvecklar mot Android.

1.2 Syfte

Projektets syfte är att utreda hur man utvecklar ett strategispel för mobilmarknaden, ifrån att skapa en spelidé till att implementera en färdig produkt. Detta uppnås genom att utveckla ett turordningsbaserat strategispel till mobilplattformen Android.

1.3 Problem

Projektets primära problemställning är att skapa ett turordningsbaserat strategispel till mobilplattformen Android. Det kan sedan delas upp i två huvudsakliga delproblem; dels behöver det tas reda på hur man skapar ett lyckat spelkoncept som många människor uppskattar, dels behöver idén realiseras som en applikation till Android.

Det första huvudsakliga delproblemet är att ta reda på hur man skapar ett lyckat spelkoncept. Det kommer att vara svårt att basera spelkonceptet på objektiva och eniga källor, då det är väldigt subjektivt vad som gör att ett spelkoncept är bättre än något annat. Svaret kan variera ifrån person till person. Med största sannolikhet är inte heller alla människor ense om vilket spelkoncept som är det bästa. Därför kommer det att behöva identifieras vad målgruppen uppskattar för spel och finna vad det är i liknande spels koncept som gör att just de spelen uppskattas. Projektgruppen kommer sedan baserat på detta behöva göra antaganden om vad målgruppen faktiskt finner attraktivt.

Det tekniska utförandet kommer att behöva baseras på pålitliga, faktabaserade metoder så att spelet får en god teknisk grund. Därför utgörs det tekniska utförandet av att hitta de olika delar som behövs i applikationen och därefter finna metoder för implementering. I händelse av att flera metoder existerar så behöver det fattas beslut om vilka metoder som ska användas. Först därefter kan själva implementeringen av applikationen starta.

1.4 Avgränsningar

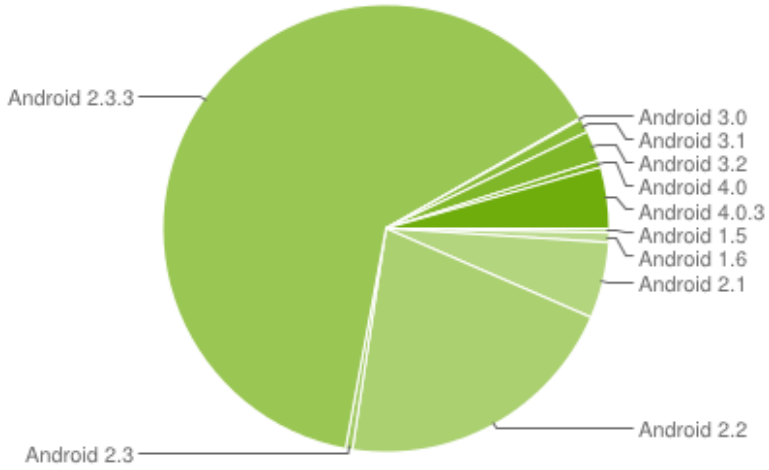
Detta projekt avgränsar sig till att endast implementera en fungerande applikation på de telefoner som funnits till förfogande. Dessa är:

- HTC Desire
- HTC Desire HD

- HTC Tattoo
- Samsung Galaxy S
- Samsung Galaxy S2

Anledningen till denna avgränsning är att gruppen ska kunna utföra tillförlitlig testning som försäkrar att applikationen fungerar som den ska på dessa modeller. Skulle applikationen fungera lika väl även för andra Android-telefoner så är det att betrakta som en bonus. Innan applikationen kan lanseras kommersiellt kommer det dock krävas stöd för fler modeller än de vi utför tester på under projektets gång.

Spelet kommer endast att stödja Android version 1.6 och uppåt. Detta på grund av att tidigare versioner kan sakna stöd för funktioner som kommer att implementeras i vår lösning; 1.6 är sedermera den äldsta tillgängliga versionen och därmed den äldsta versionen som det kan bedrivas testning till. Ytterligare argument för denna begränsning är att övriga versioner som inte inkluderats står för så pass låg marknadsandel, ca 0.3% i maj 2012 [14]. Om det senare visar sig att projektet kommer att kräva funktioner som bara är tillgängliga i en senare version av Android kan versionskravet komma att revideras. Beslutet kommer i så fall grunda sig på hur viktig funktionen är, vägt mot hur stor andel användare som utesluts.



Figur 1.1: Uppdelning av Androidversioner i maj 2012. Notera att versioner med samma version av Androids API har slagits samman, och att endast den äldsta versionen av dem sammanslagna visas. [14]

Ytterligare ett område med avgränsningar är spelets AI. Där är den huvudsakliga avgränsningen är sådan att det först och främst ses till att få fram en AI som kan ge ett lätt eller medelmåttigt motstånd, för att senare i mån av tid bygga ut och förbättra den. Spelets huvudmål kommer att vara flerspelarläget, både över nätverk och mellan flera spelare på samma telefon. AI:ns användningsområde kommer främst att vara att lära upp spelaren samt förbereda denne inför spel mot andra spelare.

Spelet kommer huvudsakligen anpassas för dess målgrupp som huvudsakligen består av de som gärna spelar andra strategispel på datorer och klassiska strategispel som schack. Det är även tänkt att nå alla som har en Android-telefon och som har tid över. Även de som endast har det när de till exempel åker buss.

1.5 Mål

Målet för projektet är i första hand att designa ett spel som är av tillräckligt hög kvalitet för att det ska ha ett signifikant underhållningsvärde för användaren. För att uppnå detta anser projektmedlemmarna att det krävs en bra och fungerande implementation av de tekniska delarna, så att spelet fungerar väl utan märkbara brister.

Ytterligare mål är att projektmedlemmarna ska lära sig så mycket som möjligt av projektet. Dels i form av professionella och aktuella metoder för att genomföra uppgifter på specifika delområden, men även i form av ökad förståelse av Android- och mjukvaruutveckling som helhet.

Några av de områden som utvecklandet av detta spel ställer krav på är speldesign, AI-programmering, nätverkskommunikation och utvecklandet av ett användargränssnitt för en mobilplattform. Förhoppningen är att i slutet av detta kandidatarbete kunna presentera en väl fungerande teknisk lösning som i möjligast mån följer de riktlinjer som förespråkas kring ämnena.

1.5.1 Google Play

En ambition som alla i kandidatgruppen delar är att utveckla spelet till en så pass högkvalitativ nivå att det går att släppa det på applikationsmarknaden Google Play. Således kommer en stor del av arbetet att omfattas av att åstadkomma ett för målgruppen tilltalande spelkoncept, utöver en tekniskt välstrukturerad och fungerande lösning.

Med stor sannolikhet kommer balansering och finjustering av spelmekaniken att kräva mer tid än vad projektets tidsram tillåter, vilket leder till att ett kommersiellt spelsläpp är mer av ett personligt mål för projektdeltagarna. Datum för publicering på Google Play är således inplanerat efter detta projekts avslutande.

1.5.2 Swedish Game Awards

Swedish Game Awards är en tävling för svenska icke-professionellt utvecklade spel, där minst hälften av utvecklarna i varje lag måste vara studerande i Sverige och delta aktivt i projektet. Tävlingen inkluderar utveckling för en mängd olika plattformar, varav Android är en av de tillåtna plattformarna [15].

Projektgruppen har enats om att anmäla projektet till Swedish Game Awards. Det innebär ytterligare ett starkt incitament till att producera ett genomarbetat spelkoncept som erbjuder användaren en trevlig spelupplevelse. Inlämnandet kommer sannolikt även ge mycket relevant och konstruktiv feedback som kan vara användbar att ta till sig och åtgärda innan applikationen släpps på marknaden.

2

Teori

I detta kapitel beskrivs den efterforskning som varit nödvändig för utvecklingen och implementeringen av detta projekt. Signifikanta delar är teori om artificiell intelligens, specifikationer för Android och återanvändbara algoritmer som använts i projektet.

2.1 Navigeringsalgoritmer

Ett återkommande problem i den här typen av strategispel är kortaste vägen-problemet. Det definieras som att givet en graf av noder och vägar mellan noderna går det att finna den kortaste vägen från nod A till nod B. För att lösa kortaste vägen problemet finns ett flertal utbredda algoritmer. Två av de vanligaste är följande:

Dijkstras Algoritm
Dijkstras algoritm utgår från startpositionen och beräknar avståndet till alla andra noder och väljer sedan vägen mellan startnoden och målet som har lägst kostnad [16].

Algorithm 1 Dijkstras algoritm

▷ Där s är startnod, V är mängden av alla noder och l_e är längden på kanten e.

För varje nod $u \in S$ lagrar vi avståndet $d(u)$
Låt $S = s$ och $d(s) = 0$
while $S \neq V$ **do**
    Välj en nod $v \notin S$ med minst en kant från S för vilken $d'(v) = min_{e=(u,v):u \in S} d(u) + l_e$ är så liten som möjligt
    Lägg till v till S och definiera $d(v) = d'(v)$
end while

I sin grundläggande form har Dijkstras algoritm en komplexitet på $O(|V^2|)$, där V

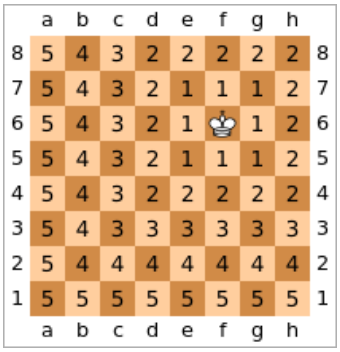
är antalet noder. Om den implementeras med en prioritetskö kommer komplexiteten ner till $O(|E| \log |V|)$, där V är antalet noder och E antalet kanter.

A*

A* är en vidarebyggnad av Dijkstras algoritm som med hjälp av heuristik optimerar valet av ordningen i vilken den går igenom noderna. Istället för att bara välja den nod som är närmast så tar den dessutom med i beaktning nodens avstånd till målet, som uppskattas av dess heuristiska funktion. Detta gör att algoritmen kan hitta den kortaste vägen till målet utan att behöva gå igenom alla noderna. Tidskomplexiteten är $O(\log h*(x))$, där $h*$ är den heuristiska funktionen.

Utav de två är A* snabbast men då spelets kartor kommer att vara relativt små, i storleksordningen 10^2 rutor, så kommer valet av algoritm inte nämnvärt att påverka spelets prestanda på en modern processor. Kortaste vägen problemet är definierat för noder med kanter mellan dem. Detta uppfylls även för hexagonala rutsystem eftersom hexagonerna är noder, och det finns alltid kanter till de angränsande hexagonerna. Denna strukturen behöver inte ändras för att tillämpa ovanstående algoritmer.

2.1.1 Avståndsberäkning



Figur 2.1: Chebyshevavstånd från en kung till alla andra rutor på ett schackbräde [17]

Avståndet i antal steg mellan två rutor i ett rutnät är ett Chebyshevavstånd [17]. Chebyshevavståndet mellan två vektorer p och q (med standardkoordinater p_i och q_i) beräknas med $d(p,q) := max_i(|p_i - q_i|)$[17]. Vilket för ett kvadratiskt rutnät blir $d(p,q) := max(|p_x - q_x|,|p_y - q_y|)$[17] och i ett hexagonalt rutnät med tre axlar blir $d(p,q) := max(|p_x - q_x|,|p_y - q_y|,|p_z - q_z|)$[18].

2.2 Begränsningar för smartphones

Smartphones har en del begränsningar som gör att de skiljer sig ifrån vanliga datorer. Nedan följer en beskrivning av ett antal begränsningar som utvecklare av applikationer till smartphones bör ta hänsyn till för att förbättra användarvänligheten.

Prestandabegränsningar
En smartphone är i allmänhet inte är lika stark prestandamässigt som en dator [19]. Därför lämpar sig heller inte prestandakrävande spel till en smartphone lika bra som de lämpar sig för skrivbordsdatorer. I de fall så det finns processer som är mer prestandakrävande än andra kan dessa placeras i en separat tråd när programmet körs, så att mottagligheten i användargränssnittet inte drabbas.

Batteritidsbegränsingar
Smartphones används i allmänhet utan att vara ansluten till elnätet och drivs då av sitt batteri. För att inte enervera användaren bör man ta hänsyn till detta och minimera energiförbrukningen så att användarens batteri inte behöver laddas lika omgående. Radiosändaren i mobilen drar mycket energi, framför allt när den startas [19]. Det är därför viktigt att minimera hur mycket och framförallt hur ofta man sänder data över Internet.

Internetanslutningsbegränsingar
På grund av en smartphones mobilitet så kommer internetmottagningen att ha en varierande täckning. Detta innebär att användaren kan tappa internettäckning helt under perioder. Dessutom är responstiden längre då mobilen är ansluten till Internet via 3G eller någon annan trådlös teknik [19].

Skärmbegränsningar
En normalstor smartphones skärmstorlek är i storleksordningen 2-6 tum över diagonalen. Detta är avsevärt mindre än en normalstor skrivbordsdators skärm, som brukar vara runt 16-22 tum över diagonalen. Skillnaden är markant och något som bör tas hänsyn till; det kan inte visas lika mycket information på en mobilskärm samtidigt som det kan på en datorskärm [19].

2.3 Plattformen Android

På Androids officiella webbplats defineras Android på följande sätt:

"Android is a software stack for mobile devices that includes an operating system, middleware and key applications." [20]

Det innebär att Android inte endast är ett operativsystem, utan inkluderar även viktiga program, samt ger stöd för hårdvarufunktionalitet såsom GPS och kamera. Själva kärnan i operativsystemet är baserad på Linux version 2.6 [20].

2.3.1 Programmeringsspråk

Programmering till Android sker nästan exklusivt i Java [21]. Det är i viss mån möjligt att använda sig av C och C++ med hjälp av Android NDK, som är en form av brygga till de underliggande C-strukturerna [22]. Det finns även ett flertal projekt som ämnar att möjliggöra programmering till Android i andra språk. Nämnvärt är det kommersiella projektet Xamarins Mono for Android [23], som med hjälp av *Mono*, en open-source

implementation Microsofts ramverk .NET [24] möjliggör utveckling till Android i C# och .NET.

Dalvik Virtual Machine
Java kan betyda två saker: programmeringsspråket Java, eller plattformen Java. Plattformen Java avser den teknologi runt språket och dess implementation via en Java Virtual Machine som ger möjlighet till att köra samma javakod överallt. Android använder sig inte av en traditionell Java Virtual Machine, utan har implementerat en helt egen virtuell maskin under namnet *Dalvik*. Android är därför rent strikt inte en del av plattformen Java, utan implementerar bara delar av Javas klassbibliotek [25]. *Dalvik* är speciellt designad för att köras på enheter där minneskapaciteten är begränsad. Minnesanvändningen har minimerats med hjälp av Dalviks eget bytecodeformat .dex [25].

Just-In-Time-kompilator
De Androidversioner som släpptes tidigare än version 2.2 saknar helt en JIT-kompilator [26]. Viss kodstruktur missgynnas av denna avsaknad. Framförallt missgynnas användningen av metoder vars syfte enbart är att hämta variabelvärden. Om koden använder publika variabler istället för metoder kan de bitarna av koden bli upp till 7 gånger snabbare [27]. Detta blir särskilt tydligt i exempelvis en renderingsloop, som körs upp till 60 gånger eller mer per sekund.

2.3.2 OpenGL

Android stödjer två huvudkategorier av renderingstekniker, nämligen Canvas API och OpenGL ES. Canvas API användas för att rita upp enklare geometriska figurer i 2D och är främst avsett för uppritande av gränssnitt, men kan också användas för enklare animationer och spel. OpenGL ES renderar 2D- och 3D-scener med hjälp av grafikprocessorn på telefonen. Eftersom många av de nyare telefonerna har en dedikerad grafikprocessor ger detta en betydande prestandaökning. Den utökande funktionalitet som erbjuds av OpenGL ES gör det till ett bättre val för alla spelprojekt, möjligtvis med undantag av de allra mest simpla spelprojekt [28].

Även Canvas API har stöd för hårdvaruacceleration via grafikprocessorn, men bara från Android version 3.0 eller senare. I skrivande stund utgör dock modeller med version 3.0 eller nyare en minoritet [14].

OpenGL är en öppen standard för att skicka instruktioner till en grafikprocessor, och har stöd på många plattformar [29] utöver Android. OpenGL ES, där ES står för Embedded Systems, är en mindre omfattande standard som endast implementerar en del av OpenGLs specifikation. Standarden är skapad för användning i inbyggda system, som till exempel telefoner [30].

OpenGL ES finns i två huvudversioner, nämligen OpenGL ES 1.X och OpenGL ES 2.X, där 1.1 och 2.0 för närvarande är de senaste. I OpenGL ES 2.X styrs många av funktionerna genom egenprogrammerade shaders, vilket bryter bakåtkompabiliteten. OpenGL ES 1.0 och 1.1 stöds av Android version 1.0 eller senare, medan Android version

2.2 eller senare har stöd för OpenGL ES 2.0. Detta projekt är avgränsat till Android 1.6 och senare, vilket gör att OpenGL ES 1.X behöver användas [31].

2.3.3 Utvecklingsramverk

För utveckling av ett spel till Android finns det en rad olika hjälpmedel. Utveckling i Java är det vanligaste och mest direkta tillvägagångssättet men det finns även andra alternativ.

Ett ramverk är en samling färdig kod som tillhandahåller funktionalitet och är till att underlätta utvecklandet av en applikation. Alla spelutvecklingsramverk till Android som undersökts erbjuder som ett första steg ett mellanlager mellan det underliggande OpenGL och den högre delen av applikationen. Vissa ramverk går längre än så och sköter allt från hantering av externa resurser, som texturer och filer, till att erbjuda en helt färdig grafik- och fysikmotor. De spelutvecklingsramverk som granskades inför projektet beskrivs nedan.

PlayN
PlayN är ett relativt nytt projekt från Google. Det har som mål att från en kodbas i Java nå en rad olika plattformar. I dagsläget stöds Java Desktop, Android, iOS och HTML5 [32]. PlayN är i ett tidigt skede och alla plattformar stöds inte fullt ut. Dokumentation är i dagsläget bristfällig. Som en del av marknadsföringen nämns bland annat att HTML5 versionen av spelet Angry Birds utvecklades med hjälp av PlayN [33]. Fokus ligger på HTML5 via WebGL vilket i sig är en mycket ny och oprövad teknik, med varierande stöd i moderna webbläsare.

LibGdx
LibGDX är ett open-source bibliotek skapat av Mario Zechner [34], författaren till Beginning Android Games [35]. LibGDX har numera likt PlayN stöd för HTML5 [36] via GWT. GWT står för Google Web Toolkit, och är ett mjukvaruprojekt som översätter Java-applikationer till Javascript så att applikationerna ska kunna köras i en webbläsare [37].

En stor fördel med LibGDX är att det stödjer direkt körning av applikationen på en skrivbordsdator. Det underlättar testning, då testningen annars måste göras på en Android-telefon eller i emulatorn, som är mycket långsam. Detta sker med hjälp av klassbiblioteken Jogl [38] eller Lwjgl [39], som båda ger möjlighet att komma åt funktioner i OpenGL med Java-kod .

Unity
Unity är mycket mer omfattande än de övriga ramverken, då Unity är en helt färdig spelmotor med tillhörande utvecklingsmiljö. I denna spelmotor ingår bland annat fysikmotor, partikelsystem samt färdiga ljud- och nätverkslösningar [40]. Utvecklingen sker mot .NET via Mono i C#, Javascript eller Boo, som är dialekt av programmeringsspråket Python [41].

Unity stödjer utveckling av 2D-spel men dess fokus ligger främst på 3D-applikationer. Unity är en kommersiell produkt, med en begränsad gratisversion där plattformen Android inte ingår. Distribution och testning av ett spel i Android kräver en extra kostnad på $400 [42].

AndEngine och Cocos2D
Två ramverk som bör nämnas men som inte undersökts närmare är AndEngine [43] och *Cocos2D for Android* [44]. Till skillnad från de övriga tre ramverken stödjer dessa två bara utveckling av 2D-spel. AndEngine är, likt LibGDX, ett open-source ramverk för spelutveckling till Android via OpenGL ES. Den största skillnaden mellan LibGDX och AndEngine är att spel som är utvecklade med AndEngine bara går att köra och testa direkt på en Android-telefon. Cocos2D eller *Cocos2D for Android* är en port av det populära ramverket Cocos2D för utveckling av 2D-spel. Det finns numera till en rad plattformar, däribland annat iPhone, HTML5 och XNA.

2.4 Artificiell Intelligens

Artificiell intelligens i spel, vanligen förkortat AI, innefattar hur datorn via logik kan utföra handlingar likt en mänsklig spelare. En begränsning i undersökningen är att denna rapport fokuserar på den information inom området som är applicerbar på ett turordningsbaserat strategispel.

Den mest välutvecklade AI:n är inte som har den mest komplicerade eller komplexa lösningen utan vars lösning är anpassad efter situation och algoritm [45].

För att kunna implementera en artificiell intelligens krävs en modell av verkligheten, innehållande information på ett sådant sätt att en "dator" kan tolka det. Flera olika tillvägagångsätt existerar för hur man kan omsätta verkligheten till en sådan modell och skapa de regler och förhållningsorder som en AI kan använda för att utföra sina drag [46].

AI kan implementeras genom två grundläggande varianter. Den enklaste varianten är en statisk implementation, som innebär att den artificiella spelaren inte använder information om dess resultat för att förändra eller förfina sitt beteende. Detta gör en statisk implementations skicklighet fullständigt beroende av programmerarens domänkunskap.

Man kan också implementera en artificiell spelare med ett dynamiskt tillvägagångsätt. En sådan implementering innebär att AI:n utnyttjar information om dess resultat för att försöka anpassa och förbättra sig själv. De dynamiska lösningar som har undersökts i detta projekt har också en statisk implementation som en komponent i lösningen. Skillnaden är dock att den statiska komponenten i en dynamisk lösning utvärderas och uppdateras mellan spel samt i vissa fall kontinuerligt genom spelet [46].

Denna rapport undersöker ett skriptbaserat tillvägagångsätt samt ett regelbaserat tillvägagångsätt.

2.4.1 Skriptbaserad AI

Skriptbaserad artificiell intelligens utgår från en lista med potentiella drag den kan tänkas utföra. För varje drag finns det sedan en rad fördefinierade krav som måste vara upprätthållna för att det draget ska kunna utföras. Den skriptbaserade implementationen kommer gå igenom alla drag sekventiellt och så fort den stöter på ett drag som möter de fördefinerade krav draget har så kommer det draget att genomföras. Det kräver således att alla tänkbara drag är sorterade i ordning efter hur fördelaktiga de anses vara för den artificiella spelaren så att de i första hand blir valda [47].

De flesta av den artificiella spelarens rundor kommer dock bestå av mer än ett drag. Så när det första draget gjorts kommer den skriptbaserade implementationen att börja överst på listan igen och gå igenom alla drag för att se om den kan hitta fler drag att utföra. Inte förrän inget drags krav längre kan uppfyllas kommer den att avsluta sina runda. Det innebär att drag som inte tillför något värde inte bör finnas med på listan över tänkbara drag från första början.

Givet tillräcklig implementeringstid så skulle AI:n kunna modelleras så att alla olika scenarion får ett drag tilldelat. Det skulle kunna åstadkommas att kraven designas på ett sådant sätt att minsta förändring på spelplanen ger upphov till ett unikt scenario. Enda begränsningen för hur väl en AI då kan spela skulle vara implementerarens domänkunskap samt dennes förmåga att göra beräkningar för vilka framtida drag olika scenarion leder till. Problemet är dock att tillräcklig implementeringstid för detta tillvägagångsätt tidigt skulle bli längre än vad något projekt kan anse som rimlig utvecklingstid, främst på grund av antalet tänkbara drag ökar exponentiellt för varje ytterligare framtida drag man överväger. En skriptbaserad implementation kräver därför att man modellerar den genom att selektivt välja ut mer generella scenarion som man anser vara relevanta att specifiera drag för.

Detta tillför dock en begränsning i den artificiella spelarens förmåga att spela spelet på bästa sätt. Dessutom gör det att den skriptbaserade implementationen i högsta grad är beroende av hur väl programmeraren klarar att utföra valet av relevatanta scenarion som ska erbjuda ett unikt anpassat drag.

Den skriptbaserade implementationen ger den artificiella spelaren ett relativt bra handlingsmönster redan vid en första simpel implementationen. Det finns dock nackdelar som att en erfaren spelare på förhand kan räkna ut hur AI:n kommer att agera. Dessutom erbjuder inte AI:n den avancerade användaren en större utmaning. Detta eftersom den är förutsägbar och dessutom ofta tvingas agera efter generaliserat handlingsmönster som lämnar brister för specifika situationer.

En erfaren användare kan med lätthet utnyttja dessa brister och ger densamme ett visst övertag. Det räcker oftast med detta övertag för att användaren relativt enkelt ska kunna vinna. Strategispel designas ofta så att de premierar den som kan skaffa sig ett taktiskt bättre position eller som kan lägga beslag på nyckelpositioner på spelplanen. Detta faktum gör att ett tidigt övertag ofta leder till att försprånget utökas successivt under spelet och gör att användaren tycker att den artificiella spelaren i slutet av spelet inte längre erbjuder tillräcklig utmaning [48].

2.4.2 Regelbaserad AI

Regelbaserade system är sätt att omvandla mänsklig expertis inom begränsade kunskapsområden till automatiserade system [49].

En regelbaserad AI lämpar sig bäst för ett problem som går att modellera genom ett antal satser, och om detta är sant utföra eller modifiera modellen på detta sätt. Problemets omfattning får inte vara alltför stort eftersom det då blir ohållbart att skapa tillräckligt med regler för att fånga hela omfattningen av problemet [49].

Den vanligaste implementationen i undersökningen är att alla tänkbara drag har ett basvärde, samt ett antal regler tilldelade. Uppfyllandet av dessa regler kan sedan modifiera basvärdet både negativt och positivt. Vilket drag som blir valt bestäms sedan slumpmässigt, där drag med ett högt värde har större sannolikhet att bli valda [45].

Olika implementationer skiljer sig lite på området hur urvalsprocessen ska gå till. En del implementationer överväger bara toppkandidaterna med högst värden, i ett försök att försöka sålla bort mindre bra alternativ från att kunna bli valda överhuvudtaget. Vissa andra implementationer överväger alla alternativ och anser att även de som anses som mindre bra alternativ kan vara fördelaktiga i rätt situation, samt att det redan tagits hänsyn till genom deras låga sannolikhet. Det är framförallt det selektiva i att bara överväga toppkandidaterna som har fått kritik, vilket gör implementationen förutsägbar. Just oförutsägbarheten och egenskapen att motspelaren inte kan förutspå vilket drag som den artificiella spelaren kommer svara med anses ofta som en avgörande fördel med en regelbaserad AI.

2.4.3 Dynamisk implementation

En dynamisk implementation av artificiell intelligens går ut på att den utnyttjar resultaten av dess val för att förfina och förbättra senare val [50]. Figur 2.2 visar hur en dynamisk implementation av den skriptbaserade AI:n kan se ut [46].

Figur 2.2: Modell av en dynamiskt scriptad AI. [46]

Utifrån regelbasen så väljs ett skript som sedan den artificiella spelaren använder för att utföra sitt drag. När draget utförts så utvärderas det och betygsätts baserat på om AI:n vunnit eller förlorat på draget och informationen används för att uppdatera regelbasen så att den premierar de drag som leder till vinst och försöker undvika de som leder till förlust. Det är på detta sätt som AI:n kan lära sig av sina misstag och faktiskt

bli bättre. Det krävs dock att AI:n får omfattande träning för att informationen ska vara användbar och därmed förbättra den artificiella intelligensen.

Är grunden för AI:n regelbaserad istället för skriptbaserad så kan i stort samma modell appliceras med skillnaden att istället för ett skript, så väljs ett antal regler som kan utföras var för sig. Endast de regler som utförts i en spelomgång går vidare till steget att utvärdera dem. När de väl utvärderats så används informationen på liknande sätt för att uppdatera regelbasen.

Det finns också svårigheter vad det gäller att utvärdera om ett drag är bra eller dåligt. Ett pålitligt och robust sätt att gå till väga är att associera alla drag som leder till vinst i ett spel som bra drag, samt alla som leder till förlust som dåliga drag. En taktik som leder till vinst är uppenbarligen framgångsrik och eftersom den artificiella spelaren har som mål att vinna spelet är det just den här typen av drag man vill premiera. En nackdel med det här tillvägagångssättet är att det kräver en omfattande inlärningsprocess för att den artificiella spelaren ska lära sig tillräckligt mycket för att vara en utmaning även för avancerade spelare. Om den utför flera bra drag och ett dåligt, varav de dåliga draget leder till att AI:n förlorar spelet så kommer alla drag som utfördes att associeras med en förlust och således anses dåliga.

Effektivare implementering för att få en snabbare inlärningsprocess vore att associera varje drag med hur framgångsrikt det är. Det innebär istället svårigheter med avvägningar för vad som ska anses som bra och gör det istället utvärderingen av dragen väldigt komplext.

2.5 Nätverk

Utveckling av välfungerande nätverkskommunikation är ingen trivial uppgift. Ett flertal olika protokoll och modeller finns att välja mellan på olika abstraktionsnivåer, och nedan följer några av dessa.

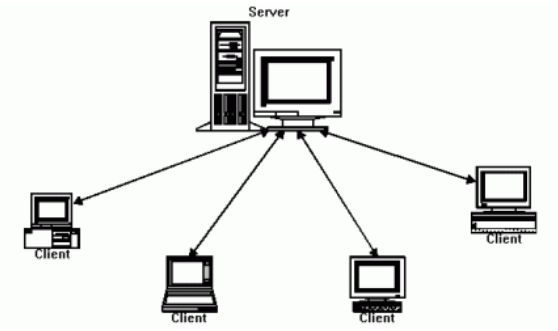
2.5.1 Nätverksarkitektur

Det finns i det stora hela två möjligheter att välja mellan när man skall bygga upp en nätverksarkitektur, klient-server- samt *peer-to-peer*-modellerna.

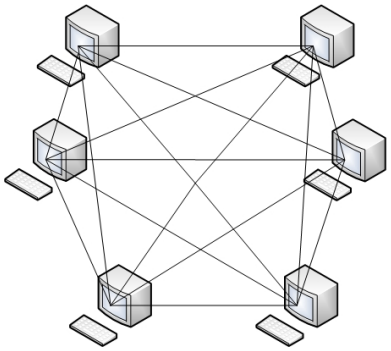
Klient-server
Klient-server-modellen [51] är baserad på en hierarkisk modell. I denna har man en central enhet, en server, som klienterna kopplar upp sig emot för att få den betjäning de behöver (se figur 2.3a). Strukturen är väldigt användbar vid tillfällen då man har behov av att spara centraliserad data såsom statistik om användarna. I vissa lösningar kan klienterna även spara undan data på servern, för att sedan kunna komma åt den från en annan enhet vid ett senare tillfälle, eller för att kunna dela data med en annan klient utan att behöva vara direkt kopplad till den andra klienten vid tillfället.

Det är även värt att notera att i vissa sammanhang så är det av säkerhetsskäl nödvändigt att ha en server för att hantera kommunikation och data. I många fall så vill

man bara ge spelarna tillgång till en begränsad mängd funktioner som de kan anropa på servern, så att de inte skall få tillgång till orättvisa fördelar genom att begå otillåtna kommandon eller få tillgång till information som endast bör vara tillgängligt för andra spelare eller servern i sig. Det är en god idé att ha en central server när det tillhandahålls betaltjänster, för att inte fel personer skall få tag på känslig information såsom kreditkortsnummer.



(a) Client-server [52]



(b) Peer-to-peer [53]

Figur 2.3: Modeller av de två nätverksarkitekturerna.

Peer-to-peer

Peer-to-peer [54] har till skillnad från klient-server-modellen ingen central enhet i sin arkitektur, utan alla enheter i systemet kan agera både som klienter och servrar (se figur 2.3b). Denna icke-hierarkiska modell ger ett mer feltolerant system överlag eftersom försvinnandet av en nod inte har någon större effekt på strukturen som helhet. Däremot är det så gott som omöjligt att spara centraliserad data.

I spelsammanhang används peer-to-peer tämligen ofta för att ge stöd åt mer lokaliserade spelomgångar med flera spelare. Spel kan då antingen ha stöd för både klient- och serverdelen i samma program, eller skilja dessa åt så att en användare kan välja att använda sig av serverdelen eller klientdelen eller bådadera.

2.5.2 Nätverksprotokoll

För reguljära spel och applikationer finns det två protokoll som kan användas för nätverkskommunikation, TCP och UDP. Bland de viktigaste delarna att ta upp i deras karaktäristik är begreppen “connection-oriented” och “connectionless”.

Connection-oriented, eller förbindelseorienterat, innebär att protokollet upprätthåller en aktiv anslutning mellan de två deltagarna under hela sessionens tid. En ny session måste startas varje gång det skall skickas information och det inte finns en session sedan innan mellan parterna, vare sig det handlar om stora eller närmast obefintliga mängder data.

Connectionless, eller förbindelselöst, medför att protokollet inte tillhandahåller sessioner för parterna som kommunicerar, utan kommunikationen är bara åt ett håll och alla paket skickas enskilt. Respons på UDP-kommunikation måste därför implementeras på applikationsnivå.

TCP
TCP [55] är ett förbindelseorienterat protokoll som genom att ge paketen serienummer ser till att tappade paket skickas igen, och allting som är ämnat att skickas kommer fram i slutändan. Protokollet använder även sig av kontrollräkning, *checksums*, för att se till så att korrupterade paket skickas om och inte används på mottagarsidan.

TCP används typiskt i applikationer där det är viktigt att all data är korrekt och anländer så som det ska, men där det inte är lika kritiskt att kommunikationen sker så snabbt som möjligt. Exempel på sådana är chattapplikationer, turbaserade spel och filöverföring med hjälp av FTP [56].

UDP
Det förbindelselösa protokollet UDP [57] är främst användbart för spel och applikationer där hastighet är kritiskt. UDP utför inga kontroller om huruvida innehållet kommer fram, och upprätthåller inte anslutningar. Däremot utför UDP i likhet med TCP kontrollräkningar för att se till att paketen som anländer inte är korrupta.

UDP används därmed ofta i sammanhang där paketförluster emellanåt är acceptabelt men där det är av högsta vikt att paket anländer till mottagaren så snabbt som möjligt. Exempel på typiska användningsområden för UDP är action- och realtidskrigsspel, streamande av musik och video samt IP-telefoni.

2.6 Speldesign

Speldesign är en kreativ process då man bestämmer regler och innehåll i ett spel. Detta bör göras innan de huvudsakliga processerna inom spelutveckling, som till exempel programmering och grafikskapande, tar fart. Ofta modifierar man då ett ett redan färdigt koncept. Resultatet av speldesignen är bland annat dokument som beskriver vad spelet ska kunna göra.

Anledningen till att speldesignen bör göras innan andra delar av utvecklingen är att om speldesignen ändras, så kommer också andra delar av spelutvecklingen, till exempel programmeringen, med stor sannolikhet behöva göras annorlunda. Det är alltså tidsbesparande om enbart speldesignen behöver ändras.

2.6.1 Speldesign för smartphones

En smartphone skiljer sig mycket ifrån en skrivbordsdator. Därför skiljer sig också sättet som man använder en smartphone på, både när och hur man använder den. Detta ställer andra krav på speldesignen. En smartphone har inte lika hög prestanda och inte lika stor

skärm. För att smartphonen ska kunna hantera spelet bör det vara mindre omfattande än vad ett motsvarande pc-spel är.

Användarna använder smartphones annorlunda än vad de använder en dator. Det kan antas att användaren vill använda mobilen när de är ute och reser snarare än när de är hemma. När de är hemma är det troligare att de använder en skrivbordsdator eller en spelkonsoll för att spela. Därför bör spelen till smartphones designas så att de kan spelas även när användaren inte är hemma.

Vidare kan man anta att användare använder smartphones i kortare intervall, till exempel under tiden de åker spårvagn. Detta till skillnad ifrån pc-användare där de ofta sitter i längre perioder. Därför bör spel vara designade så att det inte stör spelupplevelsen om man bara använder spelet under korta stunder.

Som spelutvecklare vet man inte när användaren är tvungen att avbryta spelandet. Det kan vara när användaren ska gå av spårvagnen eller något annat som plötsligt kräver ett avbrott i spelandet. Helst ska man därför kunna avbryta spelet och återuppta det närsomhelst utan att det för med sig negativa påföljder för användaren.

2.6.2 Spelanalysramverket MDA

För att kunna utvärdera hur väl en speldesign fungerar samt erbjuder för spelupplevelse för användaren, så krävs det att man har någon formell modell att utgå ifrån, som kan definiera hur väl spelupplevelsen och spelmekaniken fungerar. MDA är ett spelanalysramverk för att kunna göra just detta [58]. Ramverket utgår från 3 begrepp, dessa är mekanik, dynamik och estetik. Mekanik är den kod som utvecklaren skriver. Dynamik kan liknas vid de regler och förhållanden som råder. Estetik är de känslor som användaren upplever från spelet. Estetiken har sedan MDA-ramverket försökt att sammanfatta i 8 punkter, för att kunna ge utvecklaren en kvantifierbar lista att använda för utvärdering av vilka positiva känslor som spelet väcker.

- Sensation - En känsla som skapar välbefinnande.
- Fantasy - En känsla som stimulerar användarens fantasi.
- Narrative - En känsla av att man är en del av en berättelse.
- Challenge - En känsla där utmaningsmoment står i fokus.
- Fellowship - En känsla av social samhörighet.
- Discovery - En känsla där spelarens nyfikenhet att upptäcka nya saker tilltalas.
- Expression - En känsla som skapar självkännedom.
- Submission - En känsla som fungerar bra för tidsfördriv.

Beroende på vilket spelet är kommer det sedan att framkalla en antal av dessa känslor, som alla försöker beskriva de olika typer av känslor som spelaren kan erfara under sin spelupplevelse. Det är inte nödvändigtvis så att så många känslor som möjligt är bättre.

Ej heller är det så att någon speciell kombination av känslor eller någon enskild känsla är starkare eller bättre än någon annan, utan listan är till för att utvecklaren ska kunna skapa sig en uppfattning om vilka reaktioner spelet väcker hos spelaren.

3

Förstudie

Inledningsvis genomfördes en relativt bred litteraturstudie i helgrupp där alla gruppens medlemmar tog del av samma material, för att tillsammans kunna utforma spelkonceptet, samt ytterligare sätta oss in i hur man programmerar för plattformen Android. Under projektets gång kom sedan projektgruppens deltagare att genomföra egna litteraturstudier för att kunna lösa problem inom olika delområden.

3.1 Identifiering av delproblem

Att utveckla ett strategispel till Android kan delas upp i många olika deluppgifter. Till exempel så krävs det att det utvecklas både en spel- och grafikmotor samt förnuftig nätverkskommunikation för att få ett fungerande spel. Några av de mest signifikanta delområdena i spelutvecklingen som projektgruppens medlemmar identifierade beskrivs nedan.

Spelkoncept
För att en användare ska få en underhållande upplevelse är det en förutsättning att ha en väl genomarbetad spelidé. Det gäller dessutom att hitta ett inspirerande tema som kan locka flera olika typer av användare. Huvuddelen i uppgiften kommer att gå ut på bestämma vilka element som ska ingå i spelet, bestämma spelregler och spelets mål. Lite senare kommer vi också behöva skissa upp de enheter som ska ingå samt deras egenskaper. Det är värt att nämna att valet av tema eller balansering inte har någon större påverkan på det tekniska utförandet.

Spelmotor
Att spelet har en väl fungerande spelmotor, med en tydlig och bra arkitektur är fundamentalt för ett lyckat spel. Samtidigt ställs speciella krav på utnyttjandet av processorkraft och minne i detta projekt eftersom dessa resurser är begränsade på smartphones.

Spelmotorns struktur påverkar också säkerheten och robustheten i spelets andra delar, vilket ökar betydelsen av en väl implementerad spelmotor. Eftersom avsaknaden av en bra struktur i spelmotorns uppbyggnad leder till svårigheter att verifiera speltillstånd samt undvika buggar och fusk, så ansågs betydelsen av en effektivt implementerad spelmotor vara värd att understrykas ytterligare.

Grafikmotor
I likhet med spelmotorn är en effektiv implementation av grafikmotorn viktig. Grafikmotorns uppgifter består av att sköta utritningar och den visuella presentationen på skärmen. Här är resursfrågan än mer i fokus då tillgången på resurser i form av främst processorkraft blir avgörande för hur bra grafik en telefon kan uppvisa. Även om grafiken inte är det mest centrala i ett strategispel, så bör det eftersträvas en effektiv grafikmotor med arkitektur som klarar av att upprätthålla uppdelningen mellan logik och användargränssnitt. Detta för att underlätta utveckling av logik respektive användargränssnitt individuellt utan att det påverkar övriga delar av applikationen.

Nätverk
Inom ramarna för detta projekt anses flerspelarläget utgöra central funktionalitet. Därmed krävs det bra stöd för nätverksspelande och funktioner såsom att man enkelt skall kunna utmana andra spelare på en match. Det är av yttersta vikt att ha ett väl fungerande nätverk som ser till att såväl onödigt långa väntetider, som borttappad data i sändningarna till och från servern undviks. En utmaning som också ställs på nätverksimplementationen är att hålla nere mängden data som skickas för att inte få för stor minnesbelastning på servern och nätverket. Likaså är det en icke oväsentlig uppgift att se till att man effektivt använder processorkraften från servern och undviker redundanta beräkningar och lagring av onödig data.

AI
Utöver nätverksspelandet erbjuder spel mot AI:n ett alternativt spelläge. Det är dock ett komplicerat arbetsområde som kräver mycket efterforskning för möjligheten att åstadkomma en bra implementation. En ytterligare fråga man måste ställa sig när man utvecklar en AI till ett strategispel är huruvida AI:n skall vara implementerad som en statisk eller dynamisk AI.

Grafiskt användargränssnitt
Ett ytterligare delproblem för att utveckla ett lyckat strategispel till Android är att designa ett funktionellt grafiskt användargränssnitt på en liten skärm. I ett strategispel är det ofta mycket som ska presenteras och det är viktigt att detta görs på ett intuitivt och lättbegripligt sätt. Här inses att det finns utrymme för smarta lösningar när det gäller hur mycket som ska presenteras samt när och hur man ska prioritera information. Det finns också mycket teori på detta område som bör tas i beaktning.

Arkitektur
Arkitekturen för programmet måste i grunden vara byggd för att kunna hantera nätverkskommunikation och effektivt kunna säkerställa att varje klient i samma spelomgång befinner sig i samma tillstånd. Utmaningen med den här biten av arkitekturen är att om de betydelsefulla förändringarna i speltillståndet tappas bort så kommer det inte gå att återkonstruera speltillståndet på motståndarens klient, och därmed går det inte att uppnå ett fungerande flerspelarläge. Utöver detta så bör arkitekturen vara utformad på ett sätt som undviker cirkulära beroenden mellan olika klasser.

Verktyg
Utöver själva spelet finns en del övriga hjälpmedel som kan övervägas att skapas till ett strategispel. Framförallt är en kartredigerare ett verktyg som kan vara användbart. Den kan implementeras för att underlätta skapandet av de kartor som spelet ska utspela sig på samt ge användaren möjlighet att skapa nya kartor om denne ges tillgång till verktyget. Att ha många kartor är ett bra sätt att öka spelets omspelsvärde.

Kartrepresentation
Kartrepresentation handlar om utformandet av spelplanen och av vilka delstrukturer den byggs upp av. Spelplanen som enheter och byggnader ska befinna sig, röra sig samt interagera med varandra på. Kartor för strategispel byggs vanligen upp av ett flertal geometriskt formade rutor och innehar en stark koppling till den för ett strategispel betydelsefulla grafiska representationen. Huruvida den grafiska presentationen anses tilltalande bygger ofta till stor del på hur utformandet av spelets karta ser ut och hur väl man lyckas integrera den till att underbygga ett intuitivt och sofistikerat användargränssnitt.

3.2 Metodik

Projektetgruppen består av 6 studenter som tillsammans arbetar med detta spel. Arbetet kommer att ske delvis i helgrupp men även i delgrupper för effektivare implementering, där olika delgrupper får ansvar för implementering av olika saker. Vi har dock ett flertal möten varje vecka i helgrupp för att hela gruppen ständigt ska vara uppdaterad på hur arbetet fortlöper för alla i gruppen.

Ansvarsområden
Efter att projektets problemområden identifierats så delades projektets huvudsakliga problemområden upp i olika ansvarsområden och tilldelades en ansvarig och en vice ansvarig person. Uppdelningen är konstruerad så att den ansvariga personen har ansvar för att arbetet inom det området fortskrider i en godkänd takt, men är inte nödvändigtvis den som utför arbetet. De huvudsakliga rollerna som resonerades fram var:

Projektledarroll
Ett övergripande ansvar för hela projektet

Speldesignroll
Ansvarig för framtagandet av ett spelkoncept och dess speldesign

Arkitekturroll
Ansvarig för applikationens arkitektur

Användargränssnittsroll
Ansvarig för applikationens användargränssnitt

Nätverksroll
Ansvarig för utvecklande av applikationens nätverkskommunikation

Externa verktyg & Bug-roll
Ansvarig för framtagandet av externa verktyg samt att ha översikt över test och felsökningsverktyg

Varje möte inleds med att alla får presentera hur arbetet går inom personens ansvarsområde, vad som behöver göras och vilka hinder som finns för att genomföra det. Den personen som är vice ansvarig har också ett visst ansvar för arbetet inom området och är ställföreträdare om den ordinarie personen skulle vara borta under ett möte.

3.2.1 Process

Tanken är att projektet ska följa en iterativ utvecklingsmetod influerad av Scrum, där arbetet fördelas över iterationer. Enligt Scrum är en iteration - eller så kallad sprint en tidsperiod begränsad till en månad eller mindre där i slutet av sprinten ett produktinkrement ska kunna levereras. Tanken är att produkten på detta sätt växer fram för varje sprint [59].

Alternativet till en iterativ utvecklingsmetod är huvudsakligen en äldre metod förankrat i framförallt byggnadsindustrin och som ofta brukar refereras till som vattenfallsmetoden. Detta eftersom upplägget kan liknas vid ett vattenfall där man först genomför en förstudie och specifikation för att komma fram till vilken funktionalitet som ska finnas med och hur den ska se ut. För att sedan övergå till implementation av den funktionalitet som ska finnas med, projektet avslutas sedan ofta med testning och integration. Fördelarna kan vara att man får en kostnad för projektet när det inleds, eftersom hela projektets omfattning är specifierad. Somliga förespråkare menar också att det erbjuder kostnadskontroll och en valfrihet att byta utvecklare mellan olika steg. De största fördelarna med vattenfallsmetoden ligger ofta hos väldigt stora projekt och vid samordnandet av stora mängder resurser [60].

Vattenfallsmetoden är dock en allt mer sällan använd metod inom mjukvaruutveckling, framför allt när det gäller mindre och halvstora projekt. Skälet till det är att det ofta är svårt att överblicka hur hela systemet ska implementeras i början av projektet, samt framförallt att metoden inte är öppen för förändringar. Med den komplexitet som återfinns i dagens system är det närmast omöjligt att redan från början kunna förutse vilka anpassningar som kommer att behöva ske under projektets gång. En metod som har svårt att ta till sig förändringar har därför en väldigt tydlig brist.

Motiveringen för att välja en iterativ utvecklingsplan är alltså att det inom projektgruppen anses ge högre effektivitet, större flexibilitet och minska risken för att inte leverera en slutprodukt i sluten av projektet. Genom att ständigt jobba i kortare iterationer blir det mycket lättare att överblicka hur snabbt arbetet fortskrider tidsmässigt och anpassa implementeringsnivån därefter. Eftersom metoden bygger på att man redan från början har ett körbart programskal så elimineras mer eller mindre risken för att man i slutändan inte har något att leverera [61].

Nackdelen med en iterativ process är att det är svårt att i början av projektet få en exakt tidsbedömning eller en klar bild av hur slutprodukten kommer att se ut. Problemet behöver dock inte vara direkt knutet till den iterativa processen utan kan även bero på att problemet som ska lösas till stor del är okänt. En iterativ process erbjuder en naturlig lösning genom att tillåta att vissa detaljer lämnas öppna för nästkommande iterationer. Man kan då undvika att ta felaktiga beslut i ett tidigt stadie, något som kan få allvarliga konsekvenser.

I detta arbete kommer varje iteration att börja med en uppföljning av den föregående iterationen. Med hjälp av tidsplaneringen och en utvärdering av den föregående iteration kommer sedan nästa iteration specifieras. Kontinuerliga möten hålls för att hålla hela gruppen uppdaterad och för att samla in och uppdatera oss gällande uppgifter som ska genomföras inom varje iteration.

3.2.2 Extern hjälp

Efter en översiktlig bedömning av gruppmedlemmarnas färdigheter konstateras det att det saknas tillräcklig grafik- och ljudkompetens. Då det även planeras att ställa upp i tävlingen Swedish Game Awards samt spelet skall säljas på Google Play beslutas det att försöka ta in extern expertis för dessa områden.

3.2.3 Referenser

Spelutveckling och mjukvaruutveckling skiljer sig generellt mycket från andra akademiska arbeten med avseende på referenser. Detta beror på att mycket av informationen som behövs härstammar från internetforum, bloggar och andra liknande mindre akademiskt dokumenterade källor. Dessutom är området för en framgångsrik och användarvänlig speldesign oerhört subjektivt, vilket gör att spelutveckling kräver att även subjektiva källor undersöks samt att många av designvalen baseras på utvecklarnas tidigare erfarenheter.

Ofta är problemställningarna under implementationen sådana att det är svårt, om inte omöjligt, att alltid kunna understryka ett designbeslut med en källa. Istället måste man som utvecklare konsultera en mängd olika källor, både akademiska och rent subjektiva källor.

3.3 Tidsplanering

Projektets utvecklingsdel är uppdelad i 8 stycken iterationer, där vardera iteration varar i 2 veckor. Innan utvecklingen delades upp i iterationer, var projektet i en inledningsfas. Målet med inledningsfasen var att identifiera delproblemen och en övergripande rollfördelning för projektets deltagare.

Nedan följer projektets planerade iterationsspecifkation.

1. Simpel spelplan och ett programskal implementerat. Med programskal så åsyftas någon simpel meny att navigera i när man startar spelet, samt en spelplan. Enheter ska kunna flyttas runt på spelplanen, även om deras grafiska representation kan vara mycket enkel. Parallellt med det så ska även en enklare server/klient-modell implementeras för att kunna hantera enklare nätverkstrafik samt ge en bedömning för hur mycket tid som måste läggas på fungerande nätverkskommunikation.

2. Vid andra iterationens slut är tanken att ha ett simpelt serverprogram färdigt samt en första upplaga av spelmotor och en kartredigerare klar.

3. Vid tredje iterationens slut är tanken att ha en enkel grafikmotor färdig, något presentabelt att visa upp. I målet för denna iteration ingår också ett spelbart spel, definierat som ett spel som går att starta, spela och som avslutas genom en vinst eller förlust.

4. Vid fjärde iterationens slut är tanken att det ska det finnas en fungerande AI algoritm, nätverksspelande för två spelare samt en algoritm som slumpgenerar kartor. Det ska även finnas en komplett uppsättning av enheter och byggnader definierade och implementerade.

5. Vid femte iterationens slut är tanken att det ska finnas ett grafiskt gränssnitt och funktionalitet för att hantera flerspelarläget, där spelaren kan ansluta till spel men även skapa egna spel. Det ska även vid slutet av denna iteration vara möjligt att presentera en Beta-version som användare kan börja testa.

6. Vid sjätte iterationens slut ska det finnas nätverksspelande för fyra personer och en första version av rapporten.

7. Vid sjunde iterationens slut är tanken att ha uppdaterat ljud och grafik.

8. Vid åttonde iterationens slut är tanken att ha en färdig applikation.

Ett Ganttdiagram konstruerades som åskådliggör hur arbetet planerades att fördelas mellan olika områden och iterationer. *Se bilaga B: Tidsplanering.*

3.4 Flerspelarlösning

Inom projektgruppen anses att utformandet av spelet så att det lämpar sig väl för att spela mot varandra på androidtelefoner utgör central funktionalitet. Det traditionella

sättet att lösa det på när det gäller strategispel är att man startar ett spel tillsammans. Därefter väntar varje spelare medan motspelaren utför sitt drag. Detta innebär dock att en användare måste spela en hel spelomgång åt gången och att båda spelarna måste vänta en längre tid för att kunna starta ett annat spel. Ett sådant krav är inte hållbart för ett androidspel eftersom användaren i regel endast använder telefonen korta stunder åt gången. Det är troligtvis av denna anledningen som det knappt existerar några sofistikerade strategispel till plattformarna Android eller iOS. Vår plan var därför att ta inspiration från det framgångsrika spelet *WordFeud*, som nyligen spritt sig explosionsartat. Projektgruppen tror att framgången för *WordFeud* till stor del beror på det innovativa sättet att spela flera omgångar samtidigt och att man som användare i regel bara utför ett drag varje gång spelet startas upp. Detta innebär att spelaren ska kunna spela med allt från ett par minuter upp till ett par dagars väntetid mellan varje drag, ett relativt unikt speltempo som vi tror passar mycket bra på en smartphone. Denna lösning kräver dock en central server där alla spelltillstånd sparas, för att en spelare ska kunna utföra ett drag även om motståndarens mobiltelefon inte är ansluten till internet.

För att applikationen skall kunna stödja nätverksspelande över huvud taget är det nödvändigt att även den grundläggande arkitekturen gör det. Vi utforskar möjligheten att återanvända och anpassa en modell från ett tidigare projekt, nämligen en modell som bygger på att instruktionsobjekt skickas mellan alla olika klienter för att varje klient skall utföra samma saker, och därav behålla samma speltillstånd.

3.5 Analys av liknande spel

(a) Battle for Wesnoth **(b)** PoxNora **(c)** Civilization IV

Figur 3.1: Bilder på liknande spel.

I början av projektet kommer en del av arbetet gå ut på att söka inspiration och idéer från andra strategispel.

Battle for Wesnoth [62] (figur 3.1a) har precis som projektet en hexagonbaserad spelplan. Under de tidigaveckorna i utvecklingen var Battle for Wesnoth en utmärkt källa där tillfällig grafik kunde lånas.

PoxNora [63] (figur 3.1b) till PC har en del intressanta lösningar med sitt Action Point-system (AP) som vi kommer att utnyttja i vår egen implementation. Det bygger på att varje enhet tilldelas en mängd AP varje ny runda som sedan spenderas för att

utföra uppgifter. Sätter man en maxgräns på hur mycket AP en enhet kan spara och ser till att enheten inte genererar all AP inför varje runda får man en speldynamik där man kan välja hur man vill spendera sitt AP. Nämnvärt är att användargränssnittet för *PoxNora* är anpassat för en musknapp eftersom spelet är implementerat i Flash. Detta är likt ett pekskärmsgränssnitt på en Android telefon. Däremot har användargränssnittet tydliga brister; det är alltför lätt att av misstag utföra oönskade kommandon, och det kan vara svårt att överblicka sina valmöjligheter. Det finns dock positiva egenskaper i deras design, men eftersom det finns tydliga svårigheter att överföra detta till vårt projekt så kommer detta endast kunna implementeras i väldigt liten utsträckning. Ett annat element i *PoxNora* som är intressant är att förflyttning av enheter försvåras när de väl hamnat i närstrid. Det koster mer AP för en enhet som står nära en fiendeenhet att flytta på sig. Detta gör att det blir svårare för utsatta enheter att komma undan från en strid. Sammantaget har *PoxNora* ett flertal idéer som är värda att ta med och bygga vidare på.

Civilization-serien [2] (figur 3.1c) är en av de mest framgångsrika strategispelen genom tiderna [64]. *Civilization* utspelar sig över hela människans existens, från stenåldern till en nära framtid. Mängden spelelement och information i spelen är enorm, och segraren utnämns genom att uppfylla mål i allt från diplomati och forskning till erövring av världen. Delar av *Civilization*konceptet går att ta inspiration ifrån, men spelets komplexitet medför enorma mängder information som skulle vara opraktiskt att visa på en liten smartphone-skärm.

(a) Advance Wars

(b) Battle for Mars

(c) Wordfeud

Figur 3.2: Bilder på liknande spel.

Det finns även andra plattformar som det sökts information ifrån. Nintendo DS är ett sådant exempel, där *Advance Wars*-serien [7] (figur 3.2a) har slagit igenom stort. Det finns en rad betydande skillnader mellan denna serie och *Civilization*-serien, men de har båda nått stor framgång inom sin genre. Det visar att det finns flera olika vägar att nå framgång inom denna genre, samt att det är väldigt viktigt att ett spel anpassas efter plattformen.

Det finns också spel i denna genre till Android. *Battle for Mars* [65] (figur 3.2b) är ett exempel där inspiration och vissa lärdomar kan hämtas. Det bör påpekas att de spel som finns till Android inom strategigenren inte har nått någon vidare framgång, vad det beror på är dock oklart. En möjlig förklaring är att strategigenren i sig är svår att anpassa

för en smartphone eftersom en smartphoneanvändare inte har möjlighet att lägga ner den tid som ett strategispel ofta kräver. Studierna av strategispel på Android kommer främst att fokusera på hur man anpassar användargränsnittet till en liten skärm.

Nämnvärt är också smartphonesuccén *WordFeud* [66] (figur 3.2c), ett spel som bara till Android finns installerat på över 10 miljoner telefoner [67]. Även om spelgenren skiljer sig från detta projekts spelkoncept kan man dra lärdom av deras nätverkslösning där ett flertal spel kan vara igång samtidigt. Detta löser effektivt problematiken med att telefoner används under korta stunder samtidigt som en hel spelomgång tar väldigt lång tid.

4

Genomförande

Projektet går i sin helhet ut på att utveckla ett fullständigt spel till mobilplattformen Android. Utvecklingen börjar från grunden, med andra ord är inget givet angående hur spelet ska fungera eller vilken arbetsmetod som ska användas. Eftersom spelets ramar inte är definierade så är det inte heller till en början känt vilka delproblem som måste angripas, och inte heller vad som krävs för att dessa ska lösas. Under projektets gång måste projektgruppen systematiskt angripa och lösa alla delproblem som dyker upp, oavsett om problemen består av prestandabegränsningar eller krav från den valda spelmekaniken.

4.1 Speldesign

Vad som gör ett turordningsbaserat strategispel bra är väldigt subjektivt och svaret varierar från person till person. I detta projekt är designens mål framförallt att finna vad målgruppen anser vara underhållande, samt väva in projektgruppens egna värderingar kring strategispel.

Som ett första steg i utformningen av speldesignen testades en rad liknande strategispel, inklusive *Civilization 4*, *Pox Nora* och *Battle for Wesnoth*. Spelen behöver inte nödvändigtvis vara väldigt avancerade; ofta är de byggda på simpel speldesign som låter användaren fatta strategiska beslut. Slumpelement verkar inte vara en viktig del av spelen, men finns ändå med i bakgrunden i till exempel *Civilization*. Spelen tycks vara designade för att vara lätta att lära sig men svåra att bemästra, vilket i slutändan premierar skicklighet och ger en skickligare spelare större chans att vinna.

En ytterligare viktig aspekt för projektgruppen är att spelet blir originellt, alltså att dess spelmoment skiljer sig ifrån vad andra spel innehåller. Det anses att spelet bör ha ett strategiskt djup, alltså att spelarna kan använda sig av olika strategier för att uppnå vinst. Det skall alltså inte finnas en överlägsen strategi som slår alla andra, utan strategier måste anpassas gentemot motståndarens spelstil.

Redan i början av projektet fanns det ett dokument som beskrev vad för sorts spel som var ämnat att skapas. Det visade sig att efter läsning av dokumentet så hade alla i gruppen olika tolkningar på spelidén. Eftersom det uppstod förvirring kring spelidén som presenterats så bestämdes det att det skulle arbetas fram en ny spelidé baserad på det befintliga dokumentet. Det ansågs viktigt att ha en tydlig spelidé för att kunna uppnå en för målgruppen underhållande slutprodukt, eftersom planeringen innefattade kommersiell lansering av produkten i ett senare skede.

4.1.1 Spelidé

Att spelet skulle vara ett turordningsbaserat strategispel för mobiloperativsystemet Android med byggnader och enheter var dock något som redan var givet och som gruppmedlemmarna var överens om. Med enheterna skulle man sedan ta över byggnader, och likt Advance Wars skulle det slutgiltiga målet i spelet vara att ta över motspelarens eller motspelarnas huvudbas. Detta då konceptet kändes mer intuitivt än alternativet att vinna när man besegrat fiendens alla enheter; samtidigt möjliggör man att en något militärt svagare men taktiskt skicklig spelare kan smyga förbi fiendens enheter och vinna genom att ta över basen istället för att behöva besegra hela dennes armé. Det togs beslut om att inte inkorporera osynliga enheter eller större enheter som tar upp mer än en ruta på spelplanen, samt att en ruta på spelplanen maximalt skulle kunna ockuperas av en enhet åt gången för att undvika att de skymmer varandra och därmed orsakar förvirring.

Inspirerade av spelet *Pox Nora* bestämdes det att alla enheter skulle ha "action points" att användas till förflyttning, anfall samt specialförmågor. Detta ansågs vara ett intuitivt sätt att representera hur mycket enheter kan göra per runda, och framförallt ger det på ett simpelt sätt enormt mycket mer valmöjlighet i hur man positionerar och anfaller med enheter.

Den ursprungliga idén var också att spelet skulle innehålla många olika former av resurser med strategiska värden samt ett större antal byggnader som man skulle kunna bygga. Det valdes dock att spelet istället skulle vara mer fokuserat på stridssystemet än resurshanteringssystemet. Den nya spelidén dikterade också ett behov av ett lågt antal olika byggnader och inte heller mer än några få resurser.

4.1.2 Utvärdering av spelidé

Det finns flera sätt att testa spelidén innan de andra processerna i projektet startar. En möjlighet är att göra en brädspelsversion av spelet, som är en praktisk metod som ger en bra överblick över spelets element. Detta passar inte alla spelgenrer men lämpar sig väl för ett turordningsbaserat strategispel och var något som övervägdes. Ett annat alternativ är att programmera en simpel spelversion utan avancerad grafik eller avancerade algoritmer där man kan testa om speldesignen är korrekt.

Det bestämdes att testandet av spelidén skulle fortgå kontinuerligt under arbetet. Idén med en brädspelsvariant av spelet genomfördes inte då det ansågs lika enkelt att testa spelidén i en programmerad alfaversion av spelet.

En bit in i projektet uppmärksammades problematiken med spelupplevelsens subjektivitet. Därför gjordes en del efterforskning om hur man formellt kan utvärdera spelkonceptet eller försöka kvantifiera användarupplevelsen vilket ledde till att MDA (se 2.6.2) togs upp som ett ramverk för att kunna göra just detta. Spelet har dock inte hunnit utvärderas med hjälp av detta ramverk under projektets gång men kommer användas i framtida arbete.

4.1.3 Tema

Det fanns en hel del olika idéer om vilket tema spelet skulle ha i uppstartsfasen av projektet. Det största dilemmat var i vilken tidsperiod som spelet skulle utspela sig i samt om miljön skulle vara verklighetstrogen eller fiktiv. En konkret idé om att det skulle utspela sig i den amerikanska kolonialtiden dök också upp i denna fas, men fick inte tillräckligt med stöd bland projektgruppens medlemmar. Gruppen enades till slut om att spelet ej skulle vara verklighetstroget då en fiktiv miljö skulle ge mer alternativ för enheter och specialförmågor samt mer konstnärlig frihet.

Efter ett förslag om att spelet kunde ha ett mytologiskt tema så kom gruppen fram till att ett vikingtema skulle vara något som passar bra för spelet. Det var ett tema som väckte projektgruppsmedlemmarnas intresse och som ansågs kunna attrahera vår målgrupp.

4.1.4 Anpassning av spelkonceptet till Android

För att få en fungerade speldesign är det viktigt att förstå smartphoneplattformens begränsningar och vilken målgruppen är. Ett strategispel på en smartphone har av naturliga skäl ingen möjlighet att ersätta eller konkurrera med liknande spel på en stationär plattform. I och med detta är det viktigt att hitta en balans i hur man ska skala ner spelmekaniken jämfört med traditionella strategispel för PC för att det ska fungera på en smartphone.

Det exemplifieras i valet om spelets byggnader där vi först hade en idé om att det skulle finnas städer utplacerade på kartan som man kunde ta över och konstruera byggnader i för olika ändamål. Till exempel så skulle man kunna konstruera en byggnad för att öka sin inkomst, en annan byggnad skulle kunna utnyttjas för att producera nya enheter. Idén övergavs dock ganska snabbt då den ansågs onödigt komplicerad. För att uppnå målet att skapa ett säljbart spel så fattades det beslut om att reducera antalet byggnader och resurser. Den ursprungliga idén ersattes istället av tre enskilda byggnader, var och en upptagandes en ruta på spelplanen; en huvudstad där spelaren producerar sina enheter, byar som genererar guld varje runda och tempel som ger mana till magi eller speciella enheter. Spelaren ska inte kunna anlägga nya byggnader då fokus ska läggas på stridssystemet snarare än resurshantering och byggnation.

En av anledningarna till detta skifte av fokus var att få ett aggressivare och därmed snabbare spel. Detta eftersom det gör att en match kräver mindre drag och att dragen i sig går snabbare att utföra. Om man undersöker andra turordningsbaserade strategispel som till exempel Wordfeud så kan det ibland kan dröja upp till 3 dagar mellan dragen.

Wordfeud har hittat ett spelrytm som fungerar väl på en smartphoneapplikation och det ansågs att en liknande spelrytm skulle vara att föredra med motiveringen att man ska kunna spela spelet under kortare perioder och däremellan lägga ifrån sig det. Ett scenario är där man spelar spelet under sina bussfärder och hinner genomföra en runda innan bussfärden är slut. Det innebär att nästa runda möjligtvis inte kommer utföras förrän nästa bussfärd som leder till att det kommer att passera en del tid mellan rundorna spelaren utför.

Utan rimlig begränsning av antal byggnader och resurser skulle spelet bli mer komplicerat, dels från en utvecklingssynpunkt, men framförallt för användarna. Detta skulle troligen tilltala en mindre publik, vilket är negativt för projektets tidigare nämnda mål om att skapa ett kommersiellt gångbart spel.

4.1.5 Spelmekanik

Speldesignmässigt så är stora mängder enheter på spelplanen samtidigt inget önskvärt scenario, då det blir mycket för spelarna att hålla reda på. Projektgruppen arbetade därför med att designa en modell där det relativt lätt går att förstöra fiendens enheter. Det ansågs viktigt att aggressivt spel skulle premieras så att matcherna inte drar ut på tiden, samt då det uppmuntrar ett mer intressant spel om varje runda är händelserik redan från första rundan av spelet.

Samtidigt gäller det att fortfarande behålla strategimoment och möjligheter att använda sig av taktiska finesser. Positioneringen av enheter inför attacker mot motståndaren skall vara av stor vikt. Målet är att få ett intensivt men, ur ett strategiskt perspektiv, intressant spel som möjliggör flera alternativa strategier för spelaren.

Det bestämdes att en nivå av slump i stridssystemet skulle implementeras, som gör att spelarna inte kan förutsäga exakt hur mycket skada en attack kommer göra. Detta öppnar för fler oväntade händelser, till exempel att en enhet överlever längre än vad den ena spelaren räknat med och som därmed kan påverka spelomgångens utgång. Detta leder även till att om en spelare leder så är dennes seger mer osäker; involveringen av slumpmoment gör att spelet i sådana lägen blir mer intressant. Nackdelen är att om slumpelementet blir för stort så kan spelare bli frustrerade över att inte alls kunna förutsäga hur striderna kommer att sluta. Det ansågs därför vara viktigt att under betatestningen utreda vad spelarna tyckte om slumpmomentet och dess påverkan.

I Vengeful Vikings bör en match uppskattningsvis ta några tiotal rundor att genomföra på en normalstor karta. Detta innebär att en match kan teoretiskt sett kan ta ett flertal veckor att spela färdigt, beroende på hur lång tid varje runda tar. Spelarna skall dock själva kunna bestämma hur lång tid de har på sig att utföra dragen på. Detta gör att spelet passar både de som föredrar snabba matcher, som är färdiga på någon timme, samt de som inte vill stressa när de spelar och inte har något emot att det ibland kan ta en dag eller två innan motspelaren gjort sitt drag. Som en ytterligare konsekvens av detta beslutades att man ska kunna ha mer än ett spel i gång samtidigt, likt hur det fungerar i *Wordfeud*. Spelaren bör då även få notifikationer när det är dennes tur att göra ett drag.

Den bakomliggande mekaniken bakom alla strategispel, såväl realtid som turbaserade, har enorm påverkan på hur spelet fungerar. Det finns en stor mängd val för vilka modeller som ska användas för stridsberäkningar och hur stor inkomst spelare ska ha. Det finns ett flertal välanvända modeller för skada och försvar, alla med för- och nackdelar.

Försvarsmodell

Gemensamt för i stort sett alla strategispel är att alla enheter har en viss mängd hälsa. Det som skiljer sig är hur uträkningen av skada och försvar påverkar hälsan. Under projektets gång övervägdes ett flertal modeller. Dessa modeller påverkar enheters effektiva hälsa på olika sätt.

Det är rimligt att enheter har värden för Hälsa **H** och Pansar **P**. Med hjälp av dessa värden är det möjligt att räkna ut enhetens effektiva hälsa **EHP** som avgör hur mycket inkommande skada **S** enheten tål innan försvarsuträkningar genomförs. **E** beror kraftigt på vilken försvarsmodell som används. I nedanstående formler används även resulterande skada **R** för hur mycket hälsa en enhet förlorar efter att ha blivit anfallen.

Procentuellt pansar

En procentuell uträkning reducerar den inkommande skadan med ett visst antal procent. Det går bra att förmedla denna mekaniken till användaren genom att tydligt markera ett procenttecken efter siffervärden. Denna modell används av bland annat strategispelet *Battle for Wesnoth*.

$$R = S * (1 - P), P \leq 1$$
$$E = H/(1 - P)$$

Den effektiva hälsan påverkas kraftigt vid höga värden på pansar, vilket bör undvikas. Det finns dessutom inte stora styrkor jämfört med att inte ha någon försvarsmodell och endast öka enheters hälsa efter behov.

Exponentiellt pansar

En exponentiell uträkning för pansar leder till att pansar alltid har lika stor påverkan på hälsa oavsett hur mycket pansar man har tidigare. Det går exempelvis att låta den effektiva hälsan vara 100% av bashälsan vid 0 pansar, och för varje X pansar dubbla effektiva hälsan. Fördelen med en sådan modell är att spelet kommer vara balanserat även ifall man låter spelaren uppgradera enheterna med väldigt mycket pansar. Denna modellen används bland annat av realtidsstrategispel som *Warcraft 3* och *Heroes of Newerth*.

$$R = S * e^{-P}$$
$$E = H * e^{P}$$

I Vengeful Vikings planeras väldigt begränsade möjligheter att uppgradera enheter, vilket gör att fördelen med modellen inte utnyttjas. Dessutom är det inte tydligt för användaren hur en logaritmisk modell fungerar.

Subtraktivt pansar
Denna modell reducerar den inkommande skadan med pansarvärdet. Det är en rättfram uträkning, och användargränssnittet som krävs för att visa mängden pansar kan göras väldigt kompakt. Att räkna pansar på detta sätt gör att man behöver ta hänsyn till hur mycket enheter skadar för att sätta värdet på pansar - samma pansar och inkommande skada resulterar i att försvararen inte tar någon skada. En effekt av denna modell är att begreppet effektiv hälsa tappar styrka, det går inte att räkna ut ett värde på effektiv hälsa. Detta är för att enstaka starka attacker påverkas betydligt mindre av pansar än ett flertal svagare attacker. Subtraktivt pansar används bland annat av realtidsstrategispelet *Starcraft II*.

$R = S - P$
$E =$ Odefinierat

I slutändan valdes subtraktivt pansar ut för användning i Vengeful Vikings. Det har få nackdelar, och eftersom snabbheten på enheters attacker är någorlunda likvärdig och baserad på AP så kommer balanseringen av snabba enheter inte att påverkas i alltför stor utsträckning.

Mängd AP

Antalet handlingar som enheter kan utföra baseras på *Action Points*, som i denna rapport förkortas AP. Detta antal gäller både förflyttningar, attacker och aktivering av förmågor. Mängden AP som olika enheter har är balanserad utifrån antagandet att den minsta möjliga mängd AP man kan spendera är 1. Detta är vad det kostar att förflytta enheten en ruta på öppet landskap. Den snabbaste enheten har tilldelats en maxgräns på 9 AP. Den mest långsamma enheten måste fortfarande kunna förflytta sig någorlunda väl och har fått en maxgräns på 5 AP.

Det har valts att enheters AP inte ska fyllas på helt i början av varje runda. De återhämtar istället bara en del av maxvärdet. Detta medför den strategiska effekten att en enhet kan spara AP som den i en framtida runda kan använda för att förflytta sig långt. Samtidigt bör det inte vara möjligt för enheten att spara AP i alltför många rundor, för att därefter kunna förflytta sig och anfalla andra änden av kartan. Resultatet är att enheter varje runda återhämtar i regel strax över hälften av sin maxgräns.

Mängden AP som krävs för att anfalla har balanserats till fördel för långsamma enheter. Första anfallet kostar 2 AP, och varje ytterligare anfall under samma runda kostar 1 AP mer. Det gör att få attacker ger större utdelning per AP än flera attacker. Långsamma enheter påverkas därmed kraftigt med hänsyn till hur långt de förflyttar sig, men inte fullt lika kraftigt med hänsyn till antal attacker.

4.1.6 Oimplementerade idéer

Det planerades i tidigt stadium att enskilda enheter möjligvis skulle kunna få erfarenhet av stridande, och genom detta erhålla mer styrka på ett eller annat sätt; förslagsvis genom att kunna göra lite mer skada, få lite mer försvar eller lite mer AP. Detta prioriterades dock bort till förmån för annan funktionalitet. Det är inte uteslutet att erfarenhetssystemet kommer att vara med då spelet släpps på marknaden efter detta projekts avslut.

Ytterligare funktionalitet som planerades att ha med i spelet var förmågor. En lista med 15 förslag på olika sorters förmågor samt hur de skulle påverka spelet togs fram. Exempelvis skulle de kunna vara passiva och verka i bakgrunden eller kunna användas aktivt av spelaren. Det utröntes även om effekten av dessa förmågor skulle kunna drabba större områden och huruvida de skulle vara bundna till specifika enheter eller inte.

Det ansågs att spelet med fördel skulle kunna stödja förmågor som påverkar enheters egenskaper och hälsa. Dessa förmågor skulle både kunna vara passiva såväl som aktiva. Dessutom skulle det finnas stöd för dem att kunna drabba större områden såväl som en enskild ruta eller enhet. Ett exempel på en sådan spelaraktiverad förmåga skulle kunna vara att framkalla ett kometnedslag någonstans på kartan som skadar omkringliggande enheter. Dessutom övervägdes att eventuellt ha stöd för spelare att kunna ändra terrängtyp på enskilda rutor på kartan med hjälp av förmågor.

Dock kommer spelarbaserade förmågor inte vara med i slutprodukten av detta projekt då det på grund av tidsbrist inte var färdigimplementerat vid betaversionens släpp. Spelarbaserade förmågor kommer dock att finnas med i senare versioner av spelet. Däremot så finns det förmågor på enheter som går att aktivera, till exempel så kan schamanenheten hela både sig själv och andra enheter.

4.2 Arkitektur

För att kunna utveckla ett komplext program krävs det en bra struktur på hur alla delar av programmet ska fungera. Utan denna struktur förloras kontrollen över effekterna av ändringar i koden. Arkitekturen krävs även för att kunna försäkra att programmets körning resulterar i kontrollerade och korrekta sluttillstånd. Val av lösningar har också stor påverkan på hur programmet kan utvecklas vidare och hur effektivt programmet blir i hänsyn till prestanda.

4.2.1 Uppdelning mellan klient och server

Ett av de främsta kraven för projektet är stöd för att kunna spela över nätverk. För att kunna lösa detta måste arkitekturen i sin grund kunna hantera nätverkskommunikation. Det främsta problemet som måste lösas är att alla klienter som spelar mot varandra måste ha samma speltillstånd.

När det gällde uppdelningen mellan server och klient kom två konkreta alternativ upp.

Lägga ansvar på servern
Första alternativet som övervägdes var att använda en allvetande server som känner till hela speltillståndet. Till detta så skulle det användas en relativt simpel klient som hämtar informationen på servern för att visa den för användaren. Detta skulle innebära att servern har fullständig kontroll och tillgång till all information. Säkerheten mot fusk kan då bli väldigt hög, eftersom endast en begränsat del av speltillståndet skickas till klienterna.

Denna lösning resulterar däremot i ett flertal problem. Allteftersom spelet får fler användare så kommer det riskera att ge servern en mycket hög belastning. Främsta problemet skulle vara att man inte får respons på att man utfört ett drag förrän man kontaktat servern och fått bekräftat att det du försökte utföra faktiskt är tillåtet. Detta skulle innebära att användaren märker av och upplever en signifikant fördröjning på serverns responstid. Det skulle dessutom medföra att man alltid måste vara uppkopplad medan man spelar. Sammanfattningsvis så är detta alternativ altför begränsande för användaren.

Lägga ansvar på klienten
Det andra konkreta förslaget var att varje klient har all information om speltillståndet och själv kan avgöra ifall ett drag är tillåtet eller ej. Klienten blir då marginellt större, och man skulle teoretiskt sätt kunna använda tredje-parts programvara för att fuska genom att läsa otillåten information. Det kommer däremot inte vara möjligt att ändra på värden, exempelvis för att öka sin mängd resurser, eftersom motståndarens klient kommer märka att något gått fel när den tar emot otillåtna drag. Servern skulle endast agera som en koppling mellan klienterna. Servern vet vilka spelarna är och ansvarar för att om en spelare utför ett drag så får alla andra spelare information om draget. Det skulle vara överflödigt för servern att avgöra ifall ett drag är tillåtet. Detta för att varje klient enskilt kan avgöra ifall draget är tillåtet, och ifall ingen använder tredje-parts programvara så kommer alla att nå fram till samma resultat.

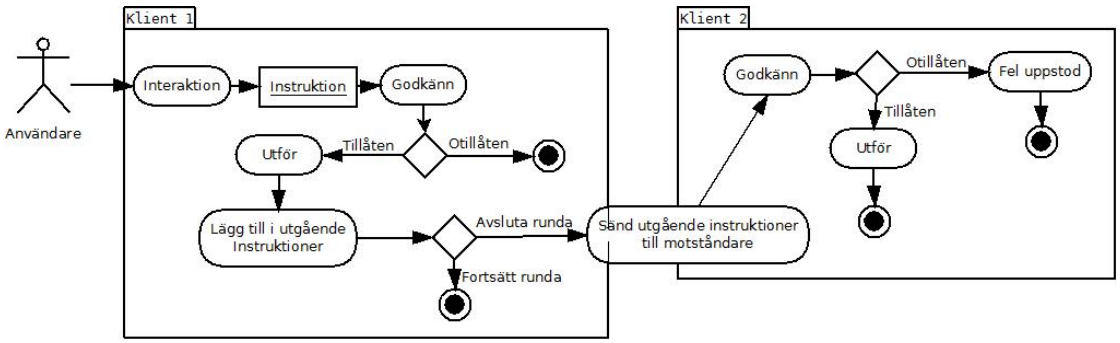
En av effekterna av att låta klienten utföra alla beräkningar är att programmet endast behöver skicka information om dragen, och inte speltillståndet. Detta gör att man inte skickar överflödig information, vilket gör att servern utan problem kommer kunna hantera ett betydligt större antal klienter. Dessutom möjliggör denna lösning en ytterst viktig typ av funktionalitet, nämligen att man inte ens behöver vara uppkopplad medan man utför en runda. Ta exemplet att man åker buss, ett tänkbart scenario då man kan spela det här spelet. Telefonen har tidigare hämtat motståndarens runda, och spelet befinner sig i rätt tillstånd. Användaren kommer då att kunna utföra en hel runda och avsluta den utan att vara uppkopplad. När telefonen så småningom får kontakt med internet så kan den skicka rundan till servern utan att användaren märker något. Eftersom en del användare kommer spela spelet på detta sätt, så är värdet av att stödja denna funktionalitet väldigt högt. Begränsningar för var och hur man kan spela spelet bör undvikas i så hög utsträckning som möjligt.

Att låta klienten ansvara för speltillståndet ansågs som det definitivt bästa alternativet. Trots att inga garantier kan lämnas kring att det inte går att skriva en fuskklient till

spelet så har den senare lösningen fördelarna att servern blir väldigt simpel och att klienten endast behöver kontakta servern en gång per runda vilket minimerar nätverks- och serverbelastning. Dessutom så är en fulltimplementerad klient nödvändig om vi vill ha "hotseat", ett spelläge mellan flera olika spelare på samma telefon. Den senare lösningen klonar inte heller data mellan servern och klienten.

4.2.2 Hantering av instruktioner

Med definition på var olika sorters information ska finnas så krävs fortfarande en lösning på vilken information som ska skickas mellan server och klienter. Det behövs tillräckligt med information för att alla klienter ska kunna nå samma tillstånd, men att skicka all information hela tiden skulle innebära enormt mycket mer datatrafik än nödvändigt. Även att bara skicka den information som har ändrats skulle innebära mycket trafik och ha svårt att ge någon bra lösning på animationer. Problemet går att lösa väl med *Command pattern*[68]. Just den här modellen har två av gruppmedlemmarna tidigare erfarenhet av för ett liknande problem och där det visade sig vara väldigt pålitligt. Modellen bygger på att det grafiska gränssnittet och grafikmodellen jobbar mot varandra för att skapa en konkret instruktion, enligt modellen kallat *Command*, vilket skulle kunna vara att flytta en specifik enhet, anfalla eller att avsluta rundan. När en instruktion sedan var skapat så skulle det utföras på den egna klienten, samtidigt som det sparas undan för att skickas till servern och så småningom också utföras på alla andra klienter i samma spelomgång (se figur 4.1). Detta innebär att alla klienter alltid kommer utföra precis samma instruktioner i precis samma ordning. Eftersom modellen endast kan påverkas via instruktioner, så kommer alla klienter att nå samma tillstånd.



Figur 4.1: Hantering av instruktioner.

Värdet i att skicka instruktioner är att det är minimalt med information som behövs för att uppdatera speltillståndet mellan klienterna, samtidigt som de är väldigt logiska att resonera och prata om.

Vi kan exemplifiera detta med en enhet A som anfaller en enhet B. För att utföra detta behövs endast informationen att man vill specifikt anfalla, med argumenten att man vill anfalla med enhet A och att man vill anfalla enheten B. Instruktionen går således inte att tolka på olika sätt. Instruktionen har i praktiken mängder med konsekvenser.

B kanske dör beroende på en mängd olika faktorer. Den informationen behövs däremot inte skickas med. Alla klienter har redan samma information, och givet instruktionen om vad spelaren utför så kommer alla klienter att utföra samma beräkningar och komma fram till samma öde för B.

Denna modellen är en lösning för att alla klienter ska behålla samma tillstånd. En viktig förutsättning för detta är att de måste börja i samma speltillstånd. Ifall något går fel så kommer det innebära svårlösta konsekvenser. För att kunna upptäcka när något gått fel så behövs ett sätt att jämföra klienternas speltillstånd, vilket är en uppgift som haft skiftande prioritet under projektets gång. Efter projektets halvtidsavslut granskades verifieringen av speltillståndet. Det hade redan diskuterats på föregående möten och det beslutats att detta skulle prioriteras ned eller strykas helt. Anledningen till nedprioriteringen är att även om man hittar ett fel så går det fortfarande inte att rätta till felet, den enda lösningen är att stänga ned spelomgången. Under utvecklingstiden så uppkom problemet någon enstaka gång som följd av uppdateringar som inte tog hänsyn till arkitekturen. Det var också fullt möjligt att upptäcka problemet vid körning av programmet i utvecklingsmiljön. Detta ger utvecklaren mycket respons på vad som gått fel även utan en implementerad verifiering av tillståndet.

Denna implementation går också att utnyttja för att återskapa fullständiga spelomgångar och visa dessa för användaren. Detta går att göra genom att spara starttillståndet samt alla instruktioner. Genom att sedan utföra alla instruktioner på starttillståndet så kommer hela spelomgången spelas upp automatiskt.

4.2.3 Separering av spelmekanik och grafik

En enhet kan endast vara placerad på en ruta, vilket innebär att när den flyttar sig till en annan ruta så sker detta ögonblickligen. Problemet med detta är att det står i konflikt med grafiken, som vill visa en mjuk rörelse av förflyttningen för användaren.

Lösningen som arbetades fram är att kraftigt separera spelmodellen och grafikmodellen. Spelmodellen innehåller all konkret information om speltillståndet; vilka enheter som finns på spelplanen och var, vad de har för hälsa och all annan information som krävs för att spela spelet utan hänsyn till grafik. Det är endast spelmodellen som ansvarar för vilka instruktioner som är tillåtna, och hur de utförs. Eftersom instruktioner utförs ögonblickligen så har spelmodellen ingen uppfattning om tid.

Grafikmodellen innehåller alla grafiska delar. Det innebär alla bilder, animationer, knappar, informationsrutor med mera. För att lösa problemet att visa information med korrekt tidsfördröjning behöver grafikmodellen dubblera en del information från spelmodellen. Varje enhet har en spelmodell-version samt en grafikmodell-version. Grafikenheten känner alltid till spel-enheten, men eftersom spelmodellen inte tar hänsyn till grafik så känner spel-enheten inte till grafik-enheten. Grafik-enheten innehåller all information som krävs för utritning, och är inte starkt bunden till regeln att bara kunna stå på rutor, den kan placeras på vilken punkt som helst på spelbrädet.

Resultatet av detta är att instruktioner kan skapa grafiska animationer som visar vad som utfördes. Om man genomför en gå-instruktion så kommer spelmodellen omedelbart veta att enheten står på den nya rutan, men grafikmodellen reagerar genom att långsamt

visa det nya speltillståndet genom att grafiskt dra enheten från den gamla rutan till den nya. Ytterligare en fördel med denna arkitekturen är att spelmodellen inte är medveten om användarens handlingar. Användaren jobbar mot grafikmodellen för att genom en serie klick få den information han vill ha. Grafikmodellen kan då visa hur långt olika enheter kan gå och vad som är tillåtet. Det är inte förrän användaren ger en fullständig instruktion som instruktionen skickas till spelmodellen och utförs.

4.2.4 Ramverk

Det första som undersöktes var huruvida ett ramverk skulle användas, då det inte alls är ett självkart val eftersom det i Android går att programmera direkt med OpenGL. Fördelen är att man inte behöver lära sig och i värsta fall låsa in sig till ett specifikt ramverk samt att det ger en ökad förståelse för det underliggande OpenGL. Dessutom så abstraherar inget av de ramverk som undersöktes bort OpenGL fullständigt. Nackdelen är dock att det tar tid från allt annat, det finns mycket att få gratis från ett ramverk och det finns därför nästan helt säkert delar som ett ramverk erbjuder som ändå hade behövts implementeras under projektets gång. Då fokus för projektet främst inte ligger på grafikprogrammering ansågs det att någon form av ett ramverk skulle användas för att undvika ta för mycket tid från de övriga tekniska problemen.

Det finns en mängd ramverk hanterar interaktionen med OpenGL samt hanterar en Android applikations livscykel. Det finns numera också många alternativ till traditionell utveckling i Java till Android, bland annat C# och .NET via Mono. I valet av ramverk var det främst följande som togs i åtankte.

Utvecklingsmiljö och programmeringspråk
Android stödjer officiellt utveckling i Java och Eclipse via Android SDK och detta var därför det primära alternativet. *Mono for Android* och utveckling i C# mot .NET ramverket var ett alternativ, men ingen i projektgruppen hade någon erfarenhet av Mono och det tycktes vara en onödigt komplicerat omväg. LibGDX, AndEngine och PlayN kan alla utvecklas i Eclipse och Android SDK. Unity har sin egen utvecklingsmiljö som endast går att köra på Windows eller OS X [69].

Storleken på ramverket samt plattformar som stöds
Flera av ramverken som undersöktes har som mål att genom en unifierad utvecklingsprocess nå en rad olika plattformar. Det tydligaste exemplet är Unity som erbjuder en helt generaliserad utvecklingsmiljö, fri från specifika detaljer från den mängd av plattformar den stödjer. Eftersom fokus för detta projekt ligger på Android föll Unity tidigt bort huvudsakligen av denna anledning. De två mindre ramverken, AndEngine och LibGDX, är jämfört med Unity mycket enklare och har ett fokus på Android. Storleken på ramverket och mängden plattformar den stödjer behöver därför inte vara en fördel utan kan snarare vara en nackdel då användningskomplexiteten ökar.

Dokumentation och möjlighet till att få hjälp
Bristen på dokumentation är ett återkommande problem i flera av ramverken.

Fördelen med att välja ett större ramverk är att det ofta finns mer komplett dokumentation, samt att de har en aktiv användarbas där möjligheten till att få hjälp är större. Här har Unity en klar fördel som ett kommersiellt projekt med en stor användarbas. AndEngine och LibGDX är däremot relativt små open source-projekt med en begränsad användarbas. PlayN har som avigsida att det ligger i ett tidigt skede där fokus just nu inte ligger på dokumentation.

Slutsats och val
I slutändan valdes LibGDX ut som det ramverk som passade projektet bäst. En bidragande orsak är att LibGDX är utvecklat av Mario Zechner, samma person som skrivit boken *Beginning Android Games* [34]. Det är dessutom gratis och tack vare dess stöd för andra plattformar kan spelet utan större modifikationer även köras på datorer vilket underlättar testning av spelet.

4.3 Spelplan

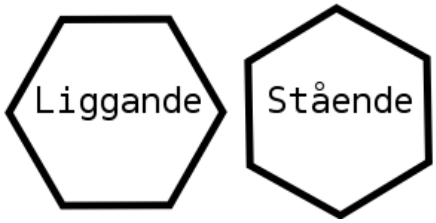
4.3.1 Rutnät & kartor

Det bestämdes tidigt att kartorna skulle representeras som hexagonala rutsystem (likt det i figur 4.2), i kontrast till det vanligast förekommande fyrkantiga rutsystemet. Anledningen till att detta valdes var för att alla former upplevs mjukare - givet en räckvidd så kommer man ha fler valmöjligheter, och formen på området man kan nå kommer också ha sex kanter istället för fyra, vilket ger en mer naturligt känsla. Hexagonala rutor ger dessutom överlag en mer naturlig förflyttning av enheter, då de i ett fyrkantigt rutnät skulle röra sig ca 40% snabbare diagonalt än längs en av axlarna medan enheter i ett hexagonalt rutnät rör sig lika snabbt i varje riktning då det är lika långt till mitten av alla grannar till en hexagon.

Figur 4.2: Ett exempel på en spelplan med hexagonala rutor. [70]

Det fanns ett flertal alternativ till hur hexagonala spelplanen skulle implementeras. Det första beslutet gällde orienteringen av hexagonerna. De val som stod till förfogande

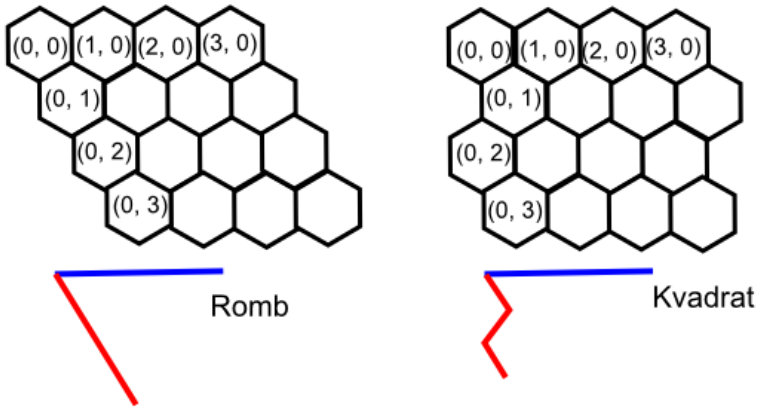
att välja mellan var liggande samt stående hexagoner, representerade i bilden nedan.



Figur 4.3: Liggande och stående orientering av en hexagon.

Efter jämförelser mellan de två orienteringstyperna valdes hexagoner med liggande orientering då man ser man något fler hexagoner i sidled, vilket var fördelaktigt eftersom det utgås ifrån att spelet visas i portrait mode på telefoner. Dessutom får man att enheterna rör sig rakt när de går i uppåt eller neråt på skärmen vilket passar bra med hur avlånga telefonskärmar är.

När det gällde kartan själv fanns två möjliga alternativ, rombisk eller kvadratisk (se figur 4.4).



Figur 4.4:
Romb - Resulterar i en helt hexagonal karta om man skär bort hörnen. Något simplare uträkningar.
Kvadrat - Resulterar i en kvadratisk karta, kräver lite mer komplicerade uträkningar.

Rombformatet i sig skulle inte göra sig väl på en telefon, i och med att en stor del av skärmen skulle bli outnyttjad i form av tomma ytor utanför romben. Detta även om hörnen på romben klipptes bort för att ge kartan en hexagonal form.

Möjligheten att lätt kunna ta fram en hexagonal karta skulle kunna vara användbar för spel med fler än fyra spelare, men med utgångspunkten att spelet troligtvis inte ska stödja mer än fyra spelare så kändes detta som en irrelevant detalj. Dessutom skulle bortskärning av hörnen innebära ett behov av programmatiskt stöd för oregelbundna kartor;

antingen genom att de icke-existerande hexagonerna representeras med null-pekare, eller som entiteter markerade som tomma på ett eller annat sätt. Oavsett representation så skulle det komplicera underhållandet och presentationen av spelplanen.

Till slut beslutades det att inte ha något explicit stöd för oregelbundna kartor. Vid de få tillfällen det skulle behövas ha mycket ogenomtränglig terräng skulle vi kunna representera det i form av redan existerande svårpasserad terräng såsom berg och vulkaner. Därmed konstaterades att den kvadratiska representationen var den bästa och således även den som skulle implementeras.

4.3.2 Terräng

Till en början övervägdes kanter runt rutorna. Inkorporering av det skulle bidra till en mer kantig och onaturlig stämning, men med det begränsade användargränssnittet på telefonskärmar i åtanke så stod det snabbt klart att det var viktigare att få fram tydliga markeringar för hexagonerna än att undvika en aning kantig känsla.

Sedan var frågan hur rörelsekostnaderna för enheter skulle implementeras på de olika terrängtyperna. Efterforskning på detta ledde fram till två huvudalternativ:

- Statiskt per terrängtyp, det vill säga, det kostar lika mycket att gå på en viss terräng oavsett enhetstyp.
- Dynamiskt bestämt för varje enhet och terrängtyp, det vill säga, varje enhetstyp har sina egna rörelsekostnader för varje terrängtyp.

Fördelarna med det första alternativet skulle vara att beräkningarna blir väldigt simpla, och allting är väldigt konsekvent samt lätt att lära sig. Alternativ två skulle innebära mer beräkningar och tillföra spelet komplexitet, vilket kan vara på gott och ont, framför allt så blir det svårare för användaren att överblicka alla alternativ.

Den stora fördelen med alternativ två är att det tillåter att till exempel flygande enheter kan ha en helt annan rörelsekostnad än enheter på land. På så sätt kan man ha enheter som kan ta sig över terräng som för andra enheter är oframkomlig.

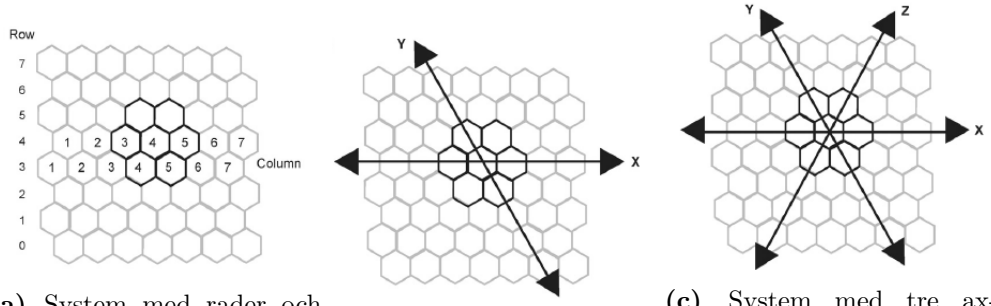
4.3.3 Navigeringsalgoritm

Valet av algoritm för att få fram kortaste väg föll till slut på Dijkstras Algoritm då den förutom att kunna ge oss den kortaste vägen också lätt kan modifieras till att även beräkna en enhets räckvidd. På så sätt behövs inte två olika algoritmer för att lösa de problemen. Dijkstras algoritm implementerades med en prioritetskö för att få ner komplexiteten, vilket är extra viktigt för mobilapplikationer för att spara på processorkraft och batteritid. Algoritmen har även modifierats så att den, när den skall lägga till en ny nod i sin körning, kontrollerar om avståndet till den nya noden skulle överstiga enhetens AP och i så fall ignorerar noden.

Kortaste väg-algoritmen kom även senare att användas till att rita ut hur mycket det kostar en enhet i AP att ta sig till de rutor som finns inom dess räckhåll. Men då den inte var anpassad för sådant beräknades värdena först genom att räkna ut den kostnaden

för den kortaste vägen till varje enskild ruta som var inom enhetens räckhåll. Efter ett tag märktes det att spelet stannade till varje gång en enhets räckvidd beräknades och visades. Efter en stunds debugging upptäcktes det att algoritmen kördes en gång för varje nådd ruta, även om den har potential att räkna vägen till alla nådda rutor i en körning. Detta är kostsamt eftersom algoritmen i sig tar mycket prestanda samt att enheter potentiellt kan nå över ett hundra rutor. Lösningen var emellertid enkel då algoritmen vid sin körning håller reda på alla avstånd och kunde lätt modifieras till att ta vara på informationen.

4.3.4 Avståndsberäkning



(a) System med rader och kolumner.

(b) System med skev y-axel.

(c) System med tre axlar (Symmetriskt Hexagonalt Koordinatsystem (SHCS))

Figur 4.5: Tre olika koordinatsystem för hexagonala rutnät. [18]

För att beräkna Chebyshevavståndet (se 2.1.1) mellan två rutor i ett hexagonalt rutnät måste rutornas koordinater först konverteras från systemet som spelet använder (figur 4.5a) till ett med tre axlar (SHCS, figur 4.5c). Detta görs genom att först omvandla koordinaterna till systemet i figur 4.5b genom att för en ruta p beräkna $p_{y'} = p_y - \frac{p_x}{2}$ för p med jämna värden på x, och $p_{y'} = p_y - \frac{p_x - 1}{2}$ för p med ojämna värden. Det sista steget för att komma till SHCS är att lägga till en z-axel och utnyttja en av SHCS egenskaper, att $\forall(x,y,z) : x + y + z = 0$ [18, 71], för att få fram z för rutorna innan avståndet kan beräknas med $d(a,b) = max(|a_x - b_x|,|a_{y'} - b_{y'}|,|a_z - b_z|)$.

4.4 Användargränssnitt

Användargränssnittet i ett spel är avgörande för dess framgång hos användarna. Gör gränssnittet inte som det ska, är förvirrande eller kräver mer än nödvändigt av användaren förstör det användarupplevelsen. Detta är framförallt viktigt att ta hänsyn till när man utvecklar till en mobilplattform, eftersom mobiltelefoner har mindre skärm och att det är svårare att mata in text på dem jämfört med skrivbordsdatorer.

I detta avsnitt kommer först en sammanfattning om gruppens arbetsprocess när det gällde användargränssnittet. Därpå kommer några av begränsningarna för en mobilplatt-

form som påverkar användargränssnittet jämfört med skrivbordsdatorer att beskrivas, samt hur de hanterades i projektet. Därefter följer lite allmän info om hur användargränssnittet i menyn och själva spelfönstret fungerar.

4.4.1 Skärmbegränsningar

Som klargjordes i *2.2 Begränsningar för smartphones* så är en smartphones skärmstorlek mindre än en skrivbordsdators skärm. Även om det till viss del kompenseras av att smartphones hålls närmare ögat. Skillnaden är dock markant och något man får ta hänsyn till; det kan inte visas lika mycket information på en mobilskärm samtidigt som det kan på en datorskärm.

I projektet bestämdes därför att försöka "gömma" information i själva spelet. Bara den viktigaste informationen bör vara presenterad för användare under en och samma tidpunkt. Själva spelkartan bör vara det som användaren ser. Övrig information som en enhets exakta attribut kan vara viktig information för användaren, men dessa behöver inte synas hela tiden. Annan information som hur mycket resurser användaren har kan dock vara motiverat att ständigt kunna ha uppsikt över eftersom det ändrar sig dynamiskt mellan varje spelomgång och informationen inte är av sådan karaktär att den går att memorera. Samtliga spel som testats har konstant visat hur mycket resurser användaren har.

4.4.2 Skärmorientering

Det är lättare för användaren att hålla mobiltelefonen med en hand om man håller den upprätt, i s.k. portrait mode, snarare än om man håller den i landscape mode. En design med få knappar samt ett minimerande av hur mycket yta användargränssnittskomponenterna ska uppta i normalläget eftersträvades. Därmed är portrait mode att föredra eftersom den uppfyller detta bäst och därmed erbjuder en större del av skärmens yta för uppvisande av spelplanen, se figur 4.6a. Följaktligen beslutades det att spelet enbart kommer att stödja portrait mode.

(a) Portrait mode.

(b) Landscape mode.

Figur 4.6: Olika skärmorienteringar.

4.4.3 Begränsningar för inmatning av text

Det tar längre tid att mata in text i ett textfält på en smartphone jämfört med en skrivbordsdator med tillhörande tangentbord [72]. Eftersom inmatningen tar längre tid så kan det irritera användaren om denne behöver mata in text ofta. Därför bör textinmatningen minimeras, bland annat genom att ha en autosparfunktion där spelnamnet sätts per automatik istället för att användaren ska behöva skriva in spelnamnet.

Textinmatning bör dock inte helt uteslutas då kartor kan behöva namngivas av användaren för att underlätta navigationen mellan dem. Dessutom antas det att man vill kunna söka efter sina vänners spel när man ska spela online. Det kan vara förvirrande för användaren om det i dessa fall alltid är autogenererade namn.

Spelet förutsätter att spelaren använder sig utav en telefon med pekskärm, då det återfinns på alla telefonmodeller inom detta projekts begränsning. En pekskärm är inte lika precisionssäker när man klickar som med muspekare på en dator. Detta eftersom man inte ser exakt vart man träffar, då fingret är i vägen samt har en viss bredd som gör träffytan oprecis.

Därför bör knappar och andra gränssnittelement ha förlåtande design. Man ska alltså inte kunna klicka fel så lätt och om man råkar göra mindre fel så ska det inte innebära att användaren gör något helt annat än vad denne tänkt sig. Därför placerades “End turn”-knappen längst upp i vänster hörn, långt bort ifrån de andra knapparna, som går att se i figur 4.6a. Det kommer att minimera risken för att man råkar avsluta sin runda tidigare än vad man tänkt sig.

Ett annat alternativ hade varit att ha “end turn”-knappen bland de andra knapparna och ha en dialogruta som dyker upp där användaren får bekräfta att denne faktiskt vill avsluta sin runda. Vi kom fram till att det i längden troligen skulle vara irriterande för användaren att gång på gång behöva bekräfta sina val - användarupplevelsen skulle påverkas mer negativt av detta än att behöva leva med den relativt låga risken att klicka fel. Därmed förekommer inga bekräftelserutor i Vengeful Vikings.

4.4.4 Implementation

Tidigt så lades fokus på att få med användargränssnitt för nödvändig funktionalitet och som möjliggör testning av spelets funktioner. Det var först några veckor in på projektet som det började läggas mer fokus på mindre essentiella knappar och förbättring av de redan existerande knapparna.

Klasser skapades för knappar, textinmatningsfält, dialogrutor och så vidare. Dessa visade sig dock vara något bristfälliga, i synnerhet textinmatningsfälten. De saknade även flexibilitet, så under den tredje iterationen skrevs användargränssnittet om till att använda sig av Scene2d, LibGDX:s paket för användargränssnitt. Omskrivningen gav tydligare kod och mycket bättre fungerande användargränssnitt.

Meny

Menyn är det första användaren ser när denne startar spelet. Det är högst viktigt att användaren får ett gott första intryck av spelet; om användaren har svårt att hitta i

menysystemet så kan det avskräcka denne från att vilja fortsätta spela. Menysystemet bör därför vara så enkelt och intuitivt som möjligt. Gruppen diskuterade många gånger under projektets gång om vad man skulle behöva visa i menyn samt när. I början fanns en väldigt simpel meny där enbart alternativen "New game", "Map editor" och "Exit" visades som går att se i figur 4.7a. Det var då väldigt enkelt för användaren att navigera.

Användarens val var då väldigt begränsade. Det gick endast att starta ett nytt spel, skapa kartor samt stänga ned spelet. Användaren hade inte tillgång till gamla spel som ännu inte avslutats. Det fanns inte heller några möjligheter att logga in på servern, eller att spela via fler spelsätt än *hotseat*. Det ansågs vara nödvändigt att utöka menyn med många fler alternativ inför en slutgiltig version.

(a) Tidig version.

(b) Slutgiltig version.

Figur 4.7: Utveckling av spelets huvudmeny.

Gruppen tog mycket inspiration av *Wordfeud*, även det ett turordningsbaserat spel, när den slutgiltiga menyn designades. I *Wordfeud* återfinns alla matcher som användaren har igång i en lista på startmenyn. För att starta nytt spel används en tydlig knapp som vid tryckning erbjuder ytterligare alternativ för spelet, till exempel möjlighet att kunna bjuda in vänner. Det bestämdes att även vi skulle använda oss av en lista över nuvarande matcher i startmenyn, men att erbjuda mer funktionalitet än vad *Wordfeud* gör. Man ska kunna logga in som olika användare, man ska kunna spela över internet, och lokalt på telefonen både mot AI och mänskliga motspelare. Man ska även kunna välja om man ska matchas mot slumpmässiga spelare alternativt starta en lösenordsskyddad match som vänner sedan kan ansluta till. Betaversionens menysystem (figur 4.7b) har likheter med det *Wordfeud* erbjuder, men är anpassad för den infomation och de funktioner som Vengeful Vikings tillhandahåller.

Spelplanen

Som en konsekvens av att det i början fokuserades mer på funktionalitet snarare än

användarvänlighet så var den första versionen av spelplanen inte så lätt för en oinsatt användare att förstå. Den främsta anledningen till detta var att även om det fanns en grundläggande struktur där alla nödvändiga knappar placerades på rätt ställe så saknade de korrekta ikoner. En grundläggande struktur fanns dock tidigt. Det fanns en knapp för att avsluta rundan, man kunde se enheters Action Points genom att markera dem och flytta runt dem, samt attackera. Relativt tidigt konstruerades även en meny inuti spelet för att rekrytera enheter som man nådde genom att trycka på den fysiska knappen "menu" på telefonen. Därav kunde det testas så att dessa funktioner fungerade som de skulle innan utvecklingen av användargränssnittet gick vidare.

Visning av möjliga handlingar
Enligt författaren Schneiderman bör användarens möjligheter att interagera med gränssnittet alltid visas tydligt [73]. Användaren bör inte heller bli förvånad över effekterna som en viss handling har. Det som begränsar antalet handlingar för enheter i Vengeful Vikings är AP-systemet. Till följd av detta behöver användaren få mycket information om hur mycket AP varje enhet har, men också information om hur mycket AP olika handlingar kostar för att exempelvis kunna veta huruvida en enhet kan anfalla en fiende efter att enheten gått ett antal steg mot fienden. Detta löses genom att tydligt markera kostnad för varje möjlig handling direkt på spelbrädet. Kostnaden är konsekvent placerad på samma plats för varje ruta, oavsett vilken handling som utförs.

4.5 Grafik

Under projektets start framgick det att ingen inom projektgruppen har erfarenhet av att rita grafik. Eftersom grafiken är en stor del av spelet beslutades det att projektgruppen behövde utomstående hjälp av en skicklig grafiker.

4.5.1 Grafisk stil

En utmaning med grafiken var att utforma stilen på spelet. Det viktigaste med grafiken identifierades till att vara dess är enhetlighet och stilmässiga klarhet snarare än avancerad grafik. Man har genom grafiken en mycket stor möjlighet att påverka hur spelet kommer att uppfattas och vilken stämning man vill framföra.

Ett beslut som hade stort inflytande på den grafiska delen var vad en ruta egentligen representerar. Det första alternativet är att kartan ska representera ett realistiskt landområde, alltså att man har tagit en del av en större karta och godtyckligt delat upp den i hexagoner. Ett problem som dyker upp då är att det krävs mjuka övergångar mellan hexagonerna för att få terrängen att kännas varierande och realistisk. Det finns flera olika lösningar men det är ett komplext problem. Med en realistisk stil på grafiken ställs också ett krav på en högre grafisk nivå överlag. Ett annat alternativ är att låta kartan representera en spelplan där varje ruta bara har en egenskap som beskrivs med en terrängtyp. Visar man detta tydligt så är det inget problem att samma bild används

för varje hexagon som har samma terrängtyp. Eftersom projektets fokus inte ligger på grafik beslutades det att den andra, mer symboliska stilen på terrängen skulle användas.

4.5.2 Animationer

Ett ytterligare problem var hur animationer av förflyttningar och attacker för enheter skulle hanteras. Ett rutnät med liggande hexagoner innebär att en enhet kan förflytta sig och attackera andra enheter i 6 riktningar där 4 är unika: norr, söder, nordöst (eller nordväst) och sydöst (eller sydväst). Norr och söder är unika riktningar eftersom perspektivet på grafiken är jämförbart med en isometrisk projektion. Detta innebär att då enheten går uppåt ska man se den bakifrån och då den går neråt ska man den framifrån. Då finns fortfarande problemet att bilden måste spegelvändas då en enhet går åt vänster vilket resulterar i att enheten blir vänsterhänt när den går åt vänster, men högerhänt då den går åt höger. Detta har dock visat sig vara en lättimplementerad, mindre detalj som spelaren inte märker av, och även spel som *Battle for Wesnoth* [62] använder denna lösning.

4.5.3 Texturepacker

Texturepackern som LibGDX tillhandahåller har en rad fördelar som förenklar hanteringen av texturer och är det av LibGDX rekommenderade sättet att ladda texturer. Detta hade mot mitten av projektet beslutats att skjuta fram av anledningen att det enbart är ett optimeringssteg. Flera fristående bilder packas ihop till en större sammansatt bild, och istället för att behöva ladda in varje enskild bild var för sig i grafikminnet laddas den sammansatta bilden in och sedan ritas regioner av den ut. Detta gör att man minskar antalet kostsamma skrivningar till grafikminnet. Problemet är att om man ska byta eller lägga till nya texturer så måste man generera om bilderna. En extra svårighet är att texturerna helst ska grupperas så att texturer som ofta ritas ut tillsammans ska packas i samma bild.

4.6 Nätverk

4.6.1 Grunder för nätverket

Tidigt i utvecklingen behövde beslut fattas för vad den grundläggande nätverkskommunikation skulle baseras på.

Det första var valet mellan TCP [55] och UDP [57]. UDP:s främsta fördel jämfört med TCP är hastigheten med vilken meddelandena kan skickas mellan olika parter, men till kostnaden av förlorade paket.

Då spelet inte innehåller något som måste ske snabbt i realtid, men däremot kräver en hög grad av synkronisering, föreföll det naturligt att välja TCP för att få robustare kommunikation utan att behöva hantera bortfallna paket på en högre nivå. Det är exempelvis mycket viktigt att alla drag som skickas över nätverket kommer fram till spelare intakta och i rätt ordning, och detta kan TCP garantera.

Med det bestämt hittades två alternativ som verkade rimliga för vårt koncept på en lite högre nivå: Java Remote Method Invocation [74], samt javas Sockets [75]. Java RMI förkastades snabbt till förmån för Sockets, då RMI ansågs ligga på en för hög abstraktionsnivå. Projektgruppen beslutade sig för att implementera kommunikationen på en lägre nivå dels för att kunna lära sig mer, dels för att kunna få mer kontroll över de skickade meddelandena.

4.6.2 Server

Modell
Det fanns flera saker att begrunda vid valet mellan klient-server-modellen [51] och peer-to-peer-modellen [54] för serverarkitekturen:

1. Spelarna bör kunna starta spel på en telefon för att sedan fortsätta på en annan.
2. Unika inloggningsuppgifter för spelare bör finnas, dels för identifikation och autentisering och dels för att göra det möjligt att i ett senare skede av spelutvecklingen kunna lägga till statistik för spelare.
3. Spelare bör kunna matchas mot varandra i realtid, och arkitekturen bör designas så att det senare skall kunna gå att starta spel även när alla spelare i en match inte är uppkopplade.
4. Det bör inte kunna skickas felaktiga drag, eller göras andra otillåtna modifikationer i spelomgångarna.

De tre första punkterna är i stort sett omöjliga att uppnå med en peer-to-peer-modell, då det enda sättet att ha informationen synkroniserad hos alla klienter är att replikera den hos alla, vilket skulle medföra väldiga mängder onödig data som skickas och sparas.

Det skulle vara möjligt att ha använda sig av båda modellerna samtidigt, där man använder servern till att hantera centraliserade uppgifter såsom spelarmatchning och sedan låter klienterna kommunicera direkt med varandra. Detta ansågs dock som ett onödigt komplicerat alternativ i jämförelse med att bara låta en centraliserad server ta hand om allting. Med dessa punkter i åtanke kom vi fram till att en klient-server-arkitektur skulle vara överlägset bäst att använda i vårt projekt.

Fjärde punkten, som både är relaterad till robusthet och fusk, skulle kunnat hanteras genom att låta servern verifiera alla kommandon som skickades till den. Dock valdes en spelarkitektur där servern är tunn och problematiken i fråga hanteras av klienterna (Se 4.2.1).

Server
I en tidig fas i projektet tillhandahöll Chalmers en fysisk server som projektgruppen kunde använda sig av. Denna visade sig fungera utmärkt från början, och det gick även förhållandevis snabbt att få servern att hantera den grundläggande kommunikationen med javas Sockets vilket ledde till att vi tidigt kunde ha ett fungerade spel med nätverkskommunikation.

4.6.3 Serverprogram

Genom projektets gång har serverprogrammet körts på den fysiska servern som en Java-applikation. Till en början var dess funktionalitet förhållandevis primitiv. Förutom den grundläggande kommunikationen kunde applikationen bara para ihop de två senast anslutna spelarna till en spelomgång på en slumpgenererad spelplan.

Nästa steg var att implementera ett autentiseringssystem där spelare kunde registrera sig och logga in med dessa inloggningsuppgifter. Detta gjordes till en början utan en databas och hanterades genom att det fanns ett antal statiskt inlagda kombinationer, och de dynamiska hamnade i arbetsminnet och försvann med varje omstart av servern. Detta var sedan det system som användes under en längre tid.

En databas behövdes givetvis läggas till mot den senare delen av projektet för att kunna stödja hantering av data på längre sikt. Efter undersökning av olika gratisvarianter av databaser konstaterades det att PostgreSQL [76] skulle möta våra krav bäst. En refaktorering av nätverket gjordes, där databasen installerades och kopplingarna till den skapades.

Slutligen implementerades en komplett algoritm för spelarmatchning, där spelarna genom ett antal olika alternativ kunde välja vilka spel de ville ansluta till eller vara värd för att sedan matchas mot varandra därefter.

4.7 Artificiell Intelligens

Ett ställningstagande som behöver tas när man utvecklar en AI till ett strategispel är huruvida AI:n skall vara implementerad som en statisk eller dynamisk AI. Beslutet bör grundas på vilken AI som passar bäst för det spel som utvecklas samt hur mycket tid det kräver från utvecklingen av spelets övriga delar.

Ett projekt som åsyftar att framförallt fokusera på flerspelarläget snarare än AI bör inte ha lika högt ställda krav på dess AI-implementation. Detta tenderar att leda till användandet av en statisk AI, som är betydligt enklare att implementera, men som emellertid inte är lika flexibel. Ett projekt där spel mot AI utgör spelets huvudattraktion bör istället överväga en implementation av en dynamisk variant.

4.7.1 Val av AI-implementation

Redan i början av projektets gång beslutades det att största fokus skulle vara på att spela mot andra personer över nätverk. Som ett resultat av detta så är utvecklandet av den artificiella intelligensen en nedprioriterad uppgift. Däremot tillför det självklart värde till spelet att kunna spela utan att behöva ha någon annan användare att spela mot. Eftersom mer resurser har lagts på att utveckla spelkonceptet och nätverksdelen av projektet har tiden att utveckla AI:n varit begränsad, och konsekvensen är att AI:n inte är särskilt utmanande. AI:n uppfyller dock en viktig funktion i form av att kunna lära ut grunderna av spelet till nya spelare utan att de skall behöva spendera sin första tid i spelet på att förlora mot mer erfarna motståndare.

Trots bristerna en skriptbaserad version [77] medför så är det den som slutligen valts ut för projektet. Anledningen till valet var att den var den enklaste och minst tidskrävande implementationen, i kombination med att även en enkel implementation ger ett robust handlande för den artificiella spelaren.

4.7.2 Implementation av AI

Kravet som ställs på AI:n är att varje gång den anropas ska den returnera en tillåten spelinstruktion. Genom denna definition kan spelmotorn under AI:ns tur anropa AI:n för att få en instruktion, utföra instruktionen, och fortsätta anropa den på samma sätt medan det fortfarande är AI:ns tur. Som resultat av denna definition så går det utan problem att byta ut hela AI:n med en ny implementation. Den skriptbaserade AI:n implementerades enligt principen beskriven i kapitel 2.4.1. AI:n bygger på en serie argument för huruvida den skall utföra specifika instruktioner eller inte. Genom att gå igenom dessa argument kommer AI:n att returnera den första möjliga instruktionen; argumentens ordning är sådan att den första möjliga instruktionen kommer att vara den som anses vara bäst. Nackdelen med denna implementation är att varje scenario måste programmeras manuellt, vilket gör att AI:n aldrig kommer kunna vara skickligare än vad programmeraren är förutseende, samt att den riskerar att bli väldigt förutsägbar.

4.7.3 Oimplementerade möjligheter

Flera olika upplägg för implementering av artificiell intelligens övervägdes men genomfördes aldrig.

Första tanke
Under projektets gång har ett flertal lösningar på AI funnits som förslag. Den första planen var en dynamisk AI(se 2.4.3) som agerar utifrån en rad regler, värdesätter varje regel och väljer den handling som har högst värde. De stora fördelarna är att det går att lägga in lärandemoment för att AI:n ska förbättra sin prestation, samt att det är väldigt lätt att modifiera svårighetsgraden både efter vilken svårighetsgrad man väljer och hur bra spelaren faktiskt spelar.

Omreviderad idé
Arkitektur designades och implementerades för att kunna stödja både en skript- och en regelbaserad AI, båda i form av statiska varianter. Arbetet med dessa båda tillvägagångssätt var också något som påbörjades. Utvecklingen av den regelbaserade varianten slutfördes aldrig, ett beslut som kan tillskrivas tidsbrist.

4.8 Betatestning

Målet i början var att ha åtminstone en fungerande betaversion innan projektets slut. Däremot var det i upppstartfasen svårt att säga exakt vad som skulle ingå i betaversionen, detta beslutades förhållandevis sent i projektet.

På grund av tidsbrist så blev strukturerade användartester skjutet på framtiden. Spelets tidiga betaversion distribuerades fortfarande till frivilliga intressenter som fick testa att spela samt komma med synpunkter och kritik i form av en informell betatestning.

Tack vare denna betatestningen mottogs intressant information om vad användarna uppskattade i programmet samt vetskap om en hel del nya buggar som inte upptäckts under tidigare interna tester. Dessa nya buggar gällde framförallt nätverkskommunikationen men även till viss del kartredigeraren som kunde få applikationen att haverera. Dessutom försvann grafiken i spelet ibland utan förklarlig anledning.

Rent spelmässigt tyckte användarna att de hade svårt att förstå vad som begränsade hur långt deras enheter kunde flytta. Användarna förstod ofta inte till fullo hur de skulle attackera med sina enheter. Detta tyder på att det i själva verket är vår implementation av AP-systemet som är tämligen svårförstådd. Däremot tyckte betatestarna att det var väldigt enkelt och intuitivt att spela när de väl förstått användargränssnittet.

En lösning på problematiken som beta-testarna själva ofta lade fram var att vi borde ha en form av tutorial där grunderna i spelet förklaras tydligt. Detta så att användarna inte direkt kastas in i spelet utan att ha en aning om vad de borde göra härnäst. Många av testarna förstod inte ens vad målet med spelet var utan att få en förklaring, än mindre hur man skulle göra för att uppnå detta mål genom att rekrytera enheter och anfalla motståndarens enheter och så vidare. Därför är det viktigt att användargränssnittet förbättras med ännu tydligare symboler för att minska på förvirringen, även om det införs en tutorial i kommande versioner.

Vad gäller införseln av en tutorial så är tanken att den antingen kan presenteras som en egen kategori när man väljer nytt spel, alternativt att den automatiskt startas upp första gången användaren startar spelet. Ett intressant alternativ skulle vara att använda sig av AI:n eller skapa en färdigskriven spelomgång där spelaren guidas genom sitt första spel.

En del av användarna tyckte också att det var konstigt att man inte kunde klicka på en by eller en tom ruta. Det gavs som förslag att man borde presentera information om en by man klickat på, som till exempel vem som äger den och vilken inkomst den ger.

Ett ytterligare resultat av testningen var iakttagelsen att den vanligaste handlingen för att försöka rekrytera nya enheter är att användaren klickar på sin huvudstad. Detta var ej något som var implementerat i användargränsnittet men som vi snabbt åtgärdade då det ansågs vara ett fullt naturligt förfarande för att rekrytera enheter.

5

Resultat

Arbetets slutgiltiga resultat är ett turordningsbaserat strategispel till Android som uppfyller syftet och målen från inledningen. Ett eget spelkoncept har först utvecklats och utifrån det har en speldesign tagits fram som sedan implementerats till ett spel. Resultaten inom de olika områdena och hur målen har uppfyllts beskrivs närgående inom sina respektive delar nedan, där kapitlet *Spelkoncept* beskriver spelet i teoretisk form och kapitlet *Produkt* om hur konceptet realiserats.

5.1 Spelkoncept

När det gäller spelkonceptet är de allra viktigaste bitarna implementerade. Spelet är spelbart och att döma av betatestningen är speldesignen lyckad på vis att användare inom målgruppen tenderar att tycka att spelet är underhållande att spela. Spelet uppmuntrar ett aggressivt spel och vi noterade att det är väldigt ovanligt att matcherna tar mer än 40 rundor under beta-testningen. Det händer saker redan i de första rundorna då uppbyggnadsfasen är så gott som icke existerande. Denna snabbhet i spelet är något som eftersträvats eftersom det kan dröja dagar mellan att rundor utförs.

I spelet finns det två resurser, guld och mana samt tre sorters byggnader, nämligen byar, tempel och huvudstäder. Spelaren inkasserar guld i början av varje runda. Mängden guld beror på hur många byar denne kontrollerar plus inkomsten genererad av huvudstaden. Med detta guld kan spelaren sedan rekrytera enheter. Mana har för tillfället inget användningsområde i spelet alls, men inkasseras också i början av varje runda och mängden beror på hur många tempel som spelaren kontrollerar. I senare versioner kommer mana användas för spelaraktiverade magier och möjligtvis rekrytering av specialenheter.

5.1.1 Strategiska aspekter

Spelet går precis som planerat ut på att ta över motståndarens huvudstad. Så fort man gjort det har man vunnit och matchen avslutas. Det är möjligt att använda olika strategier för att vinna. Fåglarna och vargarna är oerhört snabba men kan inte ta över byggnader. Det finns bärsärkar som är till för att åsamka mycket skada, men de kostar mycket och tål lite. Det finns svärdssoldater som tål mycket men som inte är särskilt mobila och de gör inte så mycket skada. Det finns yxkastare som kan attackera på avstånd men om motståndaren når dem så tål de inte mycket. Slutligen finns det en schaman som inte tål mycket, gör lite skada men som kan läka andra enheter.

Alla dessa enheter olika användningsområden, och vad som är bäst att rekrytera beror på en rad olika aspekter, till exempel vad motståndaren har rekryterat för enheter, hur motståndaren har formerat sina trupper, hur kartan ser ut, om det snabbt behövs förstärkningar någonstans på kartan och mycket mer. Detta i kombination med slumpmomentet gör att det är osannolikt att två matcher blir likadana. Det öppnar upp för ett varierande spel, vilket bidrar till spelets positiva upplevelse [58].

Det är viktigt att kontrollera vissa punkter på kartan, framförallt byarna eftersom de ger spelaren mer inkomst varje runda som gör att de kan rekrytera mer enheter. Om en spelare har större inkomst i allt för många rundor blir det svårt för den andra spelaren att vinna.

I strid är det viktigt att man själv är den som attackerar först, eftersom döda motståndarenheter inte kan slå tillbaka. Det gör att positionering av enheter till en av de viktigaste aspekterna i spelarens strategi.

5.2 Produkt

Vår produkt är ett fungerande turordningsbaserat strategispel till mobilplattformen Android. Man kan spela mot andra spelare antingen över nätverk eller på samma telefon. Man kan också spela mot en artificiell intelligens. Det finns ett fungerande kartverktyg där man kan skapa sina egna kartor att spela på, men spelet innehåller också ett antal färdigskapade kartor ifall man inte vill göra sina egna.

En nödvändig produkt förutom klienten är serverprogrammet som klienterna behöver ansluta till för att kunna spela mot andra spelare. Servern innehåller inte komplicerad funktionalitet eftersom dess främsta funktion är att skicka vidare all information till rätt mottagare. Servern har stöd för att hantera inloggningar, starta spel mellan par av spelare samt att behålla information om drag till dess att servern vid ett senare tillfälle försäkrat sig om att informationen nått klienten.

5.2.1 Spellägen

I Vengeful Vikings kan man spela mot en AI som körs på samma telefon, mot en vän på samma telefon genom att turas om att göra rundorna eller mot andra spelare över internet. I alla spellägen kan man välja vilken karta man vill spela på, eller om man vill spela en genererad karta av valfri storlek.

Vill användaren spela över internet kan denne välja att antingen skapa ett nytt spel eller gå med i ett existerande. Skapar man ett nytt spel får man välja vilken karta spelet skall utspela sig på, hur lång tidsgräns man vill ha per drag, samt vad spelomgången skall heta och vilket lösenord man vill ha på den.

Vill man istället gå med i ett specifikt spel så matar man in spelets namn samt dess lösenord. Med den givna informationen hittar servern spelet med det namnet och startar spelet mellan de två spelarna. Ifall inget spel med det givna namnet existerar informeras användaren om att spelet inte gick att hitta.

Om man går ut ur en pågående spelomgång eller om man stänger av spelet så sparas spelomgången automatiskt. Sparade spelomgångar återfinns i huvudmenyn för att snabbt kunna fortsätta pågående spel. Tröttnar man på en spelomgång och vill inte längre spela så kan man ta bort den via huvudmenyn. Gör man det så går segern automatiskt till motståndaren.

Spelet har också en fullt fungerande karteditor med vilken man kan skapa nya eller ladda gamla kartor. Skapar man en ny karta får man välja hur bred och hur lång den skall vara, inom vissa begränsningar. Alla spelets terrängtyper finns tillgängliga att placera ut på kartan, även spelets byggnader. Det finns även stöd för att ändra ägare på en utplacerad byggnad.

5.2.2 Funktionalitet

Här beskrivs de möjliga handlingar som användaren kan utföra. Alla handlingar har ett tydligt sätt att utföra dem på, samt en tydlig effekt på speltillståndet.

Rekrytering av enheter
Spelarna kan rekrytera enheter och placera ut dem på valfri ruta runt spelarens huvudstad. Detta kostar olika mycket guld beroende på vilken enhet man rekryterar. Har man inte tillräckligt med guld visas det tydligt att enheten inte går att rekrytera.

Förflyttning av enheter
Spelarna kan förflytta enheter genom att först markera dem och sedan välja vilken ruta de ska förflyttas till. Enheten kommer då att förflyttas dit, förutsatt att enheten har lika mycket eller mer AP som flytten kostar och att enheten kan stå på terrängtypen och att det inte redan står någon annan enhet där.

Attackering av fiendeenheter
Spelarna kan attackera fiendeenheter genom att först markera en av sina egna enheter. Om fiendeenheten står inom den markerade enhetens räckvidd så kan spelaren sedan markera fiendeenheten och en attack sker. Attack kan dessutom enbart ske om enheten som attackerar har tillräckligt med Action Points. För varje attack som enhet utför under en runda så kommer nästa attack under samma runda att kosta mer AP. Enheten som blir attackerad kommer inte att slå tillbaka.

Mängden skada som attacken gör på försvarande enhet beror på attackvärdet hos den attackerande enheten. Detta attackvärde modifieras med en slumpfaktor och slutligen dras försvarsvärdet hos den försvarande enhetet bort ifrån skadan. Får försvarande enhet noll eller mindre liv kvar efter attacken tas denna enhet bort ifrån spelet.

Aktivering av enheters abilities
Vissa enheter har speciella förmågor som kan aktiveras av den spelare som kontrollerar enheten. Effekten av förmågan behöver i vissa fall ett mål och spelaren behöver då markera den ruta som denne vill att effekten ska ske på. Ett exempel på en sådan förmåga är schamanens som kan läka skadade allierade enheter.

Övertagning av byggnader
Spelarna kan ta över neutrala och motståndarkontrollerade byggnader genom att förflytta en enhet till rutan som byggnaden står på. När enheten sedan står på byggnaden kan spelaren välja att spendera de AP som enheten har kvar för att försöka ta över byggnaden. Det krävs olika mycket AP beroende på vilken typ av byggnad som tas över. Detta innebär att det kan ta flera rundor för en enhet att ta över en byggnad. Enheter som är djur kan inte ta över byggnader alls.

Avslutning av runda
Spelarna kan välja att lämna över spelet till motståndaren så att denne får göra sin runda. Motståndarens får då inkassera resurser och dennes enheter får AP. Eftersom man endast får nytt guld och mer AP av att avsluta rundan, samt att alla handlingar kräver guld eller AP, så kommer spelaren till slut behöva avsluta rundan.

5.2.3 Utseende

Grafiken har skapats med flera viktiga riktlinjer i åtanke:

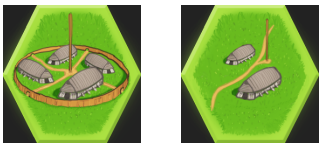
- Tydlighet - Grafiken bör vara tydlig även på en mindre telefonskärm. Starka konturer är positivt.
- Färgval - Färgerna bör reflektera bildernas natur och vara intuitiva.
- Enhetlighet - Bilderna skall inte avvika alltför mycket från varandra.
- Tillgänglighet - Grafiken bör inte vara onödigt brutal och blodig, utan kan med fördel anpassas till en bredare publik.

Alla enheter har fullständiga animationer för rörelse och anfall. Stilen för animationer i olika riktningar är så enkel som möjligt. Alla animationer ritade för att enheten pekar åt höger. För alla fall då animationen ska peka åt vänster så speglas animationen. Även anfall och rörelse uppåt och nedåt representeras av att enheten pekar åt sidan. Detta är för att kraftigt minska mängden bilder, samtidigt som det inte noteras alltför användaren eftersom animationerna är relativt små och snabba.

Nedan följer exempel på grafik. *Alla bilder är målade av projektgruppens externa grafiker.*

(a) Yxkastare **(b)** Bärsärk **(c)** Svärdsman **(d)** Hök **(e)** Varg **(f)** Schaman

Figur 5.1: Spelets enheter.



(a) Huvudstad **(b)** By

Figur 5.2: Spelets byggnader.

(a) Attack **(b)** Försvar **(c)** Hälsa **(d)** Guld **(e)** AP

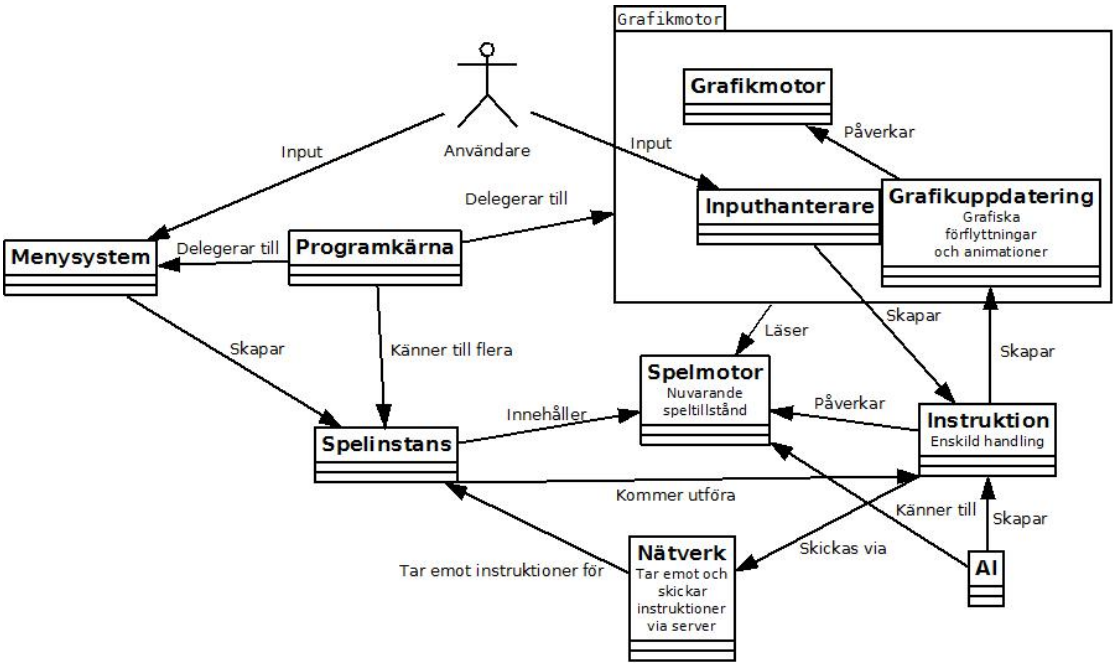
Figur 5.3: Spelets ikoner.

5.2.4 Systemdesign

Arkitekturen för Vengeful Vikings lägger allt ansvar för uträkningar och spelregler på klienten (se figur 5.4). Genom att göra så behöver servern inte ha mycket mer logik än vad som krävs för att skicka information mellan rätt klienter.

Spelmodellen
Innehåller all information om hur spelet ligger till, hur kartan är uppbyggd, all information om alla enheter och byggnader samt all information om alla spelare som är med i spelomgången. Spelmodellen har ingen uppfattning om tid. När en instruktion utförs så sker detta ögonblickligen.

Grafikmodellen
Innehåller duplicerad information av de bitar av spelmodellen som är relevanta att visa för användaren. Anledningen till att en del information är duplicerad istället för läst är



Figur 5.4: Den övergripande systemarkitekturen för klientsidan av Vengeful Vikings

för att grafikmodellen känner till konceptet av tid för att kunna uppdatera information i korrekt ordning för användaren.

Användarinteraktioner påverkar grafikmodellen enligt modellen model-view-controller. Alla interaktioner går till grafikmodellens controller. Controllern kan sedan antingen uppdatera grafikmodellen med vad som är markerat, eller skapa instruktioner att skicka till instruktionsmotorn.

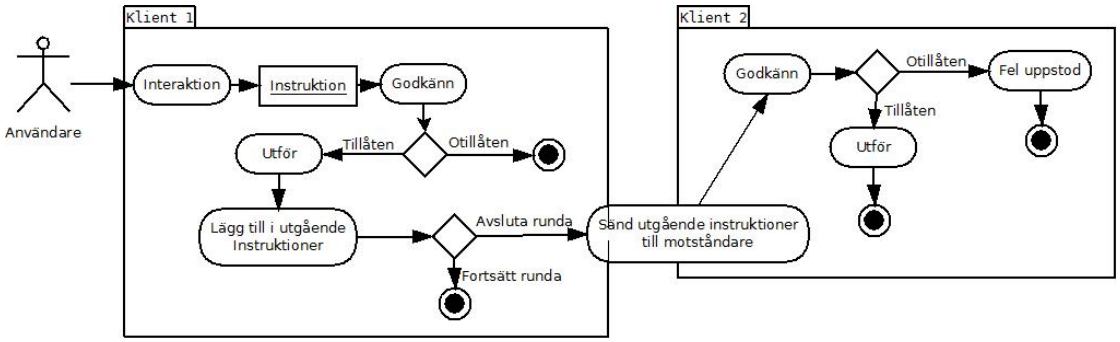
Grafikmodellen påverkas även då instruktioner utförs. Instruktionernas effekter resulterar i att en serie uppdateringar skickas till grafikmodellen. Dessa grafikuppdateringar har en strikt uppfattning av tid, och körs under ett fördefinierat antal sekunder. Endast en dominant uppdatering kan vara igång, resten placeras i en kö som utförs så snart den aktiva uppdateringen är genomförd.

Uppdateringar kan också skapa andra uppdateringar som utförs parallellt. Ett exempel på detta är när enhet A anfaller enhet B. Den dominanta uppdateringen är animationen av att enhet A gör ett anfall. I ögonblicket som attacken träffar skapas ytterligare en grafikuppdatering som visar hur mycket skada enhet B utsattes för.

Instruktionsmotorn

Alla konkreta handlingar användaren utför som påverkar spelmodellen sker genom instruktioner enligt Command-modellen. Anledningen till denna modell är att samma instruktioner skickas till motståndarens klient och utförs på så sätt identiskt på alla klienter enligt figur 5.4. När instruktioner utförs påverkar de spelmodellen, och genom att göra detta skickar de också uppdateringförfrågningar till grafikmodellen. Instruktionerna

har även logik för att verifiera att de är tillåtna. Denna verifieringen av instruktioner gör att grafikmodellen och AI:n inte har något krav på att skapa korrekta instruktioner, samtidigt som det effektivt samlar all spellogik i spelmotorn och instruktionsmotorn.



Figur 5.5: Hantering av instruktioner.

Artificiell intelligens

Den artificiella intelligensen är väldigt löst kopplad till resten av programmet. Den har behörighet att läsa all information i spelmodellen för att kunna räkna ut bra instruktioner. Endast en metod är definierad för AI:n. Detta är att givet att det är AI:ns tur returnerar den en instruktion som är tillåten att utföra, samt att den så småningom måste returnera en instruktion som lämnar över turen till motståndaren. Styrkan i denna modellen är att det går väldigt bra att byta ut hela den artificiella intelligensen till en helt annan implementation. Den nuvarande implementationen är en skriptad AI. Det innebär att den har en lista med logik som den utför tills den lyckas skapa en instruktion som är tillåten. Första instruktionen anses vara den bästa under omständigheterna och returneras.

Menysystemet

Menysystemet är en direkt tillämpning av de standardobjekt som finns i LibGDX. Menyn delegerar knapparnas funktionalitet till controller-klasser enligt model-view-controller. Menyn är till för att visa användaren funktionalitet för inloggning och att starta spel.

Nätverksgränsnittet

Nätverksgränsnittet är väl avgränsat från andra delar av programmet. Tack vare detta går det att köra hela programmet utan nätverk, samt att utvecklingen av spelet har kunnat fortskrida utan hänsyn till att nätverket inte var fullständigt implementerat förän mot projektets slut. Detaljer om nätverkets funktionalitet är beskrivet i nästa kapitel.

Programkärnan

Ansvarar för att starta en Android-process och delegera till antingen menysystemet eller grafikmodellen beroende på om ett spel är aktivt. Programkärnan innehåller också

nätverksgränsnittet, samt samlingsklasser för läsning av grafik, xml-filer och sparade spelomgångar.

5.2.5 Nätverksdesign

Grundstrukturen på nätverket är baserat på server-klient-modellen. Vi använder oss av en central server som har en databas för att spara all nödvändig data såsom inloggningsuppgifter, spel och drag.

Databasen som används är PostgreSQL 9.0. Servern sköter inloggning, kopplar ihop de spel som körs med sina deltagare samt hanterar drag som spelare skickar in.

Nätverkskommunikationen är TCP-baserad och sköts i praktiken med hjälp av javas Sockets. Konstruktionen är robust och klarar väl av att hantera olika typer av väntade och oväntade avbrott och anslutningsproblem.

Meddelandestruktur

De meddelanden som skickas följer ett eget protokoll, som separerar meddelandenas syfte från deras data. Meddelanden som inte följer protokollet slängs helt enkelt bort.

Meddelanden som rör specifika spelinstanser eller önskemål skickas med ett slumpgenererat tal, en "nonce". Denna sparas på klientsidan, och svar från serversidan innehåller denna nonce för att meddelanden från servern inte skall påverka andra önskemål än det meddelandet är ämnat till.

Matchmaking

Det finns ett extensivt system för att hantera spelmatchning, det vill säga, ihopparande av spelare till spelomgångar enligt deras önskemål.

Nedan följer en lista över de speltyper som finns:

- *Slumpmässig:* Spelomgången kommer att infinna sig på en helt slumpgenererad karta.
- *Standard:* Spelomgången kommer att infinna sig en karta utvald bland de standardkartor som finns på servern.
- *Slumpmässig standard:* Spelomgången kommer att infinna sig på en av standardkartorna på servern, men den kommer att väljas slumpmässigt bland dem vid spelets start.
- *Valfri:* Spelomgången kommer att infinna sig på en karta som värden skickar med.

Även om serverns spelmatchningsalgoritm stödjer speltyperna *Standard* och *Slumpmässig standard*, så finns det i nuläget inget stöd för dessa på klientsidan.

Den som vill vara värd för en match kan då välja en av dessa speltyper, och välja namn och lösenord för sin spelomgång om så önskas.

Vill man bara gå med i en redan existerande spelomgång så kan man välja mellan att:

1) Explicit skriva in namn och lösenord på spelomgången. Väljer man att inte skydda sin spelomgång med lösenord så matchas man mot spelare som väntar på slumpmässiga motståndare, och kommer eventuellt in i omgången direkt.

2) Söka efter en slumpmässig motståndare. Då matchas man mot alla de värdar som redan väntar på motståndare och inte har skyddat sina spel med lösenord. Hittas ingen matchning går man vidare till andra som står i kö för slumpmässiga motståndare och ser om det finns någon matchning där. Om inte, så placeras man i kön själv.

Det finns stöd för olika kartstorlekar för värdar, men man kan för närvarande inte välja storlek på kartan när man ska ställa sig i kön för slumpmässiga motståndare.

Om man loggar ut eller blir frånkopplad i väntan på motståndare så tas man bort från väntelistan.

Inloggningssystem

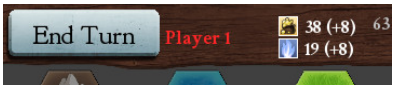
Inloggningssystemet är uppbyggt att ha en simpel layout och användning. De senast använda inloggningsuppgifterna sparas på telefonen för att man skall slippa skriva in dessa varje gång. Explicit registrering av nya användare stöds för tillfället inte, i och med vi har ett snabbinloggningssystem som tillåter registrering med slumpmässiga inloggningsuppgifter med endast en knapptryckning.

5.2.6 Användargränssnitt

Huvudmenyn består av tre knappar: “New Game” för att komma till ytterligare en meny med spelalternativ, “Map Editor” för att komma åt baneditorn och längst ner en knapp som visar ens onlinestatus och som när man trycker på den leder till en meny med olika alternativ för inloggning, registrering, etc. Onlinestatusknappen finns även längst ner i alla andra menyer för att användaren lätt ska kunna se om han/hon är online och komma åt dess funktioner. I mitten av skärmen visas ett område där användaren kan se alla spel som är igång, om det är dennes tur, samt annan allmän information. Alla spel har dessutom en knapp med vilken användaren kan välja att ge sig och ta bort spelet.

Information till användaren, som till exempel felmeddelanden, förmedlas via dialogrutor, som dessutom används till att låta denne välja karta att spela på, registrera sig, logga in, etc. När en dialogruta visas på skärmen mörkas bakgrunden ner och spelet slutar att ta emot kommandon tills användaren reagerat på den och stängt ner den.

Användargränssnittet inne i det aktiva spelet består av två paneler. Den ena uppe i toppen av skärmen som innehåller information som inte ändras så ofta som vems tur det är och spelarens inkomst samt en knapp för att avsluta pågående spelrunda (se figur 5.6a). Valet av position för knappen för avslutning av runda gjordes med åtanke på att den bara används en gång per runda. Det är också bra att ha den långt bort från den undre delen av skärmen för att undvika att den klickas på av misstag. Då ett avslut av ens omgång inte är något som man kan ångra är det bra att knappen är på en plats där man inte så lätt trycker på den av misstag.



(a) Den övre panelen.

(b) Den nedre panelen.

Figur 5.6: Användargränssnittets paneler.

Den nedre panelen består av kontextuell information och knappar för handlingar som används mer ofta under en spelomgång (se figur 5.6b). Dessa knappar är placerade i den nedre delen av skärmen för att användarens fingrar bara ska behöva täcka en liten del av skärmen då knapparna används [78]. I normaltillstånd visar den nedre panelen en knapp för att få upp menyn där man kan köpa nya enheter, men om markerar en enhet på spelplanen utökas storleken på panelen och mer detaljerad information om den markerade enheten visas, samt knappar för enhetens eventuella förmågor och tillfälliga effekter.

När användaren trycker på en enhet markeras den och rutor utanför dess räckvidd blir mörkare, medan de rutor den kan gå till får en siffra som visar hur mycket AP det skulle kosta att gå dit (se figur 5.7). Trycker spelaren sedan på en av rutorna inom dess räckvidd så flyttas enheten dit. Man attackerar genom att trycka på en fientlig enhet inom attackavstånd för en av sina enheter man markerat.

Figur 5.7: Markering av rutor inom räckvidd.

Man kan byta enhetsfokus genom att trycka på en annan vänlig enhet, och vill man avmarkera en enhet kan man antingen trycka utanför dess räckvidd eller trycka på en knapp i användargränssnittet som är till för att konsekvent avmarkera och avbryta pågående handlingar. Det går även att markera en fientlig enhet för att se information om den genom att trycka på den, förutsatt att man inte har någon av ens egna enheter markerade.

För att få överblick över kartan kan spelaren zooma ut eller in genom att utföra en pinch. Man kan fortfarande kontrollera enheter i zoomat läge, men i det mest utzoomade läget blir precisionen av ens tryckande lidande. Olika användare föredrar olika

zoomnivåer, så implementationen stödjer mjuk zoom till valfritt värde.

När motståndaren har utfört sina drag spelas de upp för spelaren i den ordning de utförts. Detta ger en snabb återblick av vad som skett sedan den förra rundan. Kameran centreras vid platsen varje handling för att man lättare ska kunna se vad som skett.

Vill spelaren köpa nya enheter gör den det genom att antingen trycka på en knapp i användargränssnittets nedre panel, trycka på menyknappen på sin telefon eller klicka på spelarens huvudstad. Sedan visas en meny där man kan välja vilken enhet man vill köpa samt få information om enheterna. När man valt att köpa en enhet markeras de rutor runt ens huvudstad där man kan placera enheten och kameran flyttas automatiskt till huvudstaden för att ytterligare förtydliga vilket område man kan placera enheten på.

(a) Meny för att köpa enheter.

(b) Rutor markeras där man kan placera en köpt enhet.

Figur 5.8: Användargränssnitt för att köpa enheter.

Skärmen där man köper nya enheter (se figur 5.8a) består av en stor ruta i mitten där man ser information om enheten man har valt, hur enheten ser ut, vilka attribut den har, en beskrivning av dess funktion samt hur mycket den kostar. Runt informationsrutan finns en knapp för varje enhet. Genom att trycka på dessa knappar visas information om den valda enheten. När man hittat en enhet man vill köpa trycker man på “Buy” och får sedan placera ut den nära sin huvudstad (se figur).

Kartverktygets användargränssnitt är inspirerat av det i spelet, och använder samma implementation för att visa kartan. Längst ner har det en permanent panel med knappar för att spara, ladda och skapa nya banor samt en knapp för att få upp ytterligare en panel för val av terrängtyp och en liknande knapp för byggnader.

6

Diskussion

Under projektets gång har det dragits en mängd lärdomar om vad spelutveckling till Android innebär. Resultatet av spelet anses enligt projektgruppens medlemmar lyckat, med goda förhoppningar om att kunna vidareutveckla spelet ytterligare och slutligen släppa det på Google Play. Detta är för att det kommer fortsätta implementeras funktionalitet som fortfarande är eftersträvad men inte hunnits med under detta projekt. Vi har dessutom vissa bekymmersamma buggar när det gäller telefonernas nätverkskommunikation som vi inte hunnit lösa och som kommer inkluderas en lösning till i en slutgiltig version.

I detta kapitel kommer vi i mer i detalj diskutera olika delar av projektet, hur resultatet blev i förhållande till vår planering och huruvida våra metoder verkligen var bäst lämpade. Huvudsyftet med diskussionen är att redogöra hur ett framtida liknande projekt skulle kunna göras bättre.

6.1 Resultatdiskussion

Det koncept och de planer som utvecklats i början är till stor del realiserade. Under projektets gång insågs successivt att all funktionalitet som planerades att ha med från början inte skulle hinna implementeras. Framförallt upptäcktes det att nätverksdelen var allt mer komplex och tidskrävande att implementera än vad som hade planerats för. Det var svårt att i början förutse vad man kan kräva att nätverksdelen ska kunna hantera. I slutändan så har ett fungerande spel trots allt implementerats, som går att spela både mot AI och mot andra spelare över internet, alternativt på samma telefon. Det är möjligt att ha igång flera spel samtidigt samt att avvakta med att utföra dragen tills man har tid. Därav anser vi att den viktigaste av projektets övergripande funktionalitet har implementerats.

6.1.1 Spelets koncept

Eftersom de viktigaste delarna av spelets koncept är implementerade i vår slutliga version är vi nöjda med vad vi åstadkommit. Fortfarande saknas framförallt spelaraktiverade magier, men dessa är inte avgörande för spelets underhållningsvärde. Det faktum att vi bestämde att spelets huvudfokus skulle vara på stridssystemet, snarare än på resurshantering och byggande tror vi gör spelet mer lättillgängligt för användarna, då det gör spelet mindre komplext.

Ett problem med spelkonceptet som uppkommit under testning är att matcherna ofta känns avgjorda tidigt då ena spelaren fått övertaget. Om denne har en större inkomst i guld kan spelaren också rekrytera mer trupper än den andre och på vis ta över fler byar och få ännu mer guld att rekrytera trupper för.

En lösning på problemet är att på något sätt ge fördelar till den spelare som ligger efter. Detta skulle dock förlänga matcherna, så det är en avvägning som vi har gjort att låta den ledande spelaren få mer och mer fördelar snarare än att förlänga känslan av att matchen inte är avgjord. Vill man vända en match får man göra det snabbt, då det blir svårare och svårare. Alternativt kan spelaren utnyttja de tillfällen då motståndarens gör ett större taktiskt misstag. Dessutom påverkar slumpen ibland utgången av strider, vilket kan ge spelaren som ligger under ett tillfälle att komma ikapp.

MDA

För att uppskatta underhållningsvärdet av Vengeful Vikings så kan ramverket MDA [58] användas. Ramverket har inte använts under projektets gång på grund av tidsbrist. MDA är beskrivet i kapitel 2.6.2.

Det positiva med att använda sig av ramverket för att sammanfatta spelupplevelsen från användarna är att man får en kvantifierbar lista över vad spelarna upplever i spelet och vilka element som är positiva ur användarsynvinkel. Inom MDA bör man inte sträva efter att täcka så många känslor som möjligt, och känslorna är inte rangordnade på något vis. Det som bör strävas efter är att de känslor som täcks av spelet ska vara kraftiga.

Däremot så kan man se hur spelet uppfattas och kanske även vilken målgrupp man kan attrahera efter vilka positiva känslor det väcker. Det kan även vara bra att använda sig för att se hur förändringar av element i spelet påverkar användarupplevelsen. Det finns egentligen ingen negativ aspekt med att försöka formalisera processen med att utvärdera spelkonceptet, utöver att det kan vara tidskrävande och uppta tid från projektets resurser.

Sammanfattningsvis är det ett bra tillskott för ett spel att försöka sig använda sig av ett ramverk som MDA eller någon annan formell definition för att utvärdera sitt spelkoncept. Första steget i ett förbättringsarbete är att mäta och utvärdera det som redan existerar för att se vad som kan förbättras samt att kunna verifiera hurvida en förändring också blir en förbättring [79].

Speldesign

Antalet enheter har reducerats i slutversionen jämfört med tidigare revisioner av speldesignen. Tidigare fanns hela tolv enheter i åtanke, medan den nuvarande version endast

har sex stycken. I efterhand så anser vi att det var ett bra val att minska antalet enheter då det gör spelet lättare att sätta sig in i och att varje enhet nu har en tydligare roll och funktion. Vi tror att detta blir mindre förvirrande för användaren och att det därmed minskar inlärningskurvan. Många av de mest lyckade spelen på Google Play har relativt simpla koncept, så vi har anledning att tro att fler spelare uppskattar vårt spel om det inte är alltför komplext.

Slumpmomentet i striderna är designat så att det inte ska påverka utgången alltför mycket. Detta då det strategiska elementet i spelet minskas med ökad slumpfaktor. Vi var tidigt överens om att spelet skulle vara strategiskt inriktat, men samtidigt ger det ett visst underhållningsvärde om man inte på förhand vet exakt hur striderna kommer sluta.

Spelarna kan för närvarande inte använda mana, vilket är förvirrande för spelare av den nuvarande versionen då mana ändå finns med i spelets användargränssnitt. Det kommer dock kommer att komma till användning senare när spelaraktiverade magier implementerats innan spelets lansering på Google Play.

6.1.2 Nätverk

Mycket av nätverksdelen, i synnerhet grunderna, har fungerat över förväntan. Dock har delarnas komplexitet ökat snabbt med införseln av ny funktionalitet och medfört ett antal icke-triviala problem. Nedan reflekteras över centrala koncept inom nätverket.

Klient-server
Klient-server-modellen har fungerat utmärkt för projektet. Med designen som inte lägger de tyngre beräkningarna på servern har nätverkstrafiken i sig fungerat smidigt, samtidigt som det blivit enklare att isolera vissa problem kopplade till kommandon över nätverk. Det skulle vara möjligt att lägga till funktionalitet för en peer-to-peer modell när spelarna har matchats mot varandra via servern, med resultatet att servern skulle belastas med mindre trafik. Detta skulle däremot bryta vår funktionalitet att kunna fortsätta spela och utföra drag utan att motståndaren är uppkopplad, en bieffekt som inte är önskvärd. I detta fall skulle det knappast vara värt att använda peer-to-peer, då mängden nätverkstrafik för applikationen på det stora hela är väldigt sparsam; i storleksordningen omkring tio kilobyte per minut för en intensiv match.

Ett tekniskt missöde orsakade att projektgruppens server gjordes otillgänglig under en tid. Även om en temporär server kunde läggas upp, förlorades tillgång till den data som låg på huvudservern, bland annat pågående spel. Detta reflekterar tydligt en av de största nackdelarna med en klient-server-modell - hela systemet är väldigt beroende av en enda enhet, och felaktigheter på den har menlig påverkan på hela arkitekturen.

TCP
TCP:s anslutningsorienterade protokoll är i grunden lätthanterligt och väldigt passande för vår applikation. Dock så har vi valt att inte implementera timeouts för de tillfällen då anslutningen av någon anledning bryts, vilket innebär att vid de tillfällena tas

autentisering och platser i kön för väntande spel bort helt och hållet.

Den här implementeringen, som är menad att främja serverns robusthet, skapar då ironiskt nog en mindre robust spelupplevelse för spelare som har en instabil anslutning - exempelvis om användaren sitter på ett tåg och tappar sin anslutning då och då.

Eftersom det är möjligt att spela flera spel samtidigt, har det även behövts läggas till funktionalitet i form av unika identifieringssiffror för att kunna se till att vissa snarlika nätverksmeddelanden bara påverkar de spelomgångar som de är menade att påverka.

Även bekräftningsmeddelanden för kommandon som skickas till servern har funderats på. TCP ser visserligen till att meddelandena når fram, men kan inte garantera att interna fel på servern hindrar dem från att hanteras korrekt.

Här kan vi dra paralleller till vad en UDP-implementation skulle kräva - nämligen timeouts samt responsmeddelanden för både meddelandenas mottagning och hantering. Med de lärdomar vi har fått från utvecklingen av nätverket, anser vi att det inte skulle bli besynnerligt svårt att byta protokoll från TCP till UDP för vårt projekt, givet mer tid. Inte heller skulle det vara svårt att använda UDP i framtida, liknande projekt.

Inloggingssystem och spelmatchning

Inloggningssystemet har fungerat på ett mycket tillfredsställande sätt. Införandet av databasen var ett välbehövligt steg i processen, men i samband med detta försvann möjligheten att registrera sig - mycket på grund av att snabbinloggningsfunktionaliteten utför just detta på ett simpelt sätt.

Med det sagt, vårt spel är inte komplett utan att spelarna kan registrera sig själva med ett godtyckligt namn som de andra spelarna kan se, med inloggningsuppgifter som användaren själv har valt.

Detta hänger även mycket ihop med spelmatchningen. Visserligen innehåller den de flesta element man kan önska sig med de valmöjligheter Vengeful Vikings har. Däremot saknas fortfarande möjligheten att kunna lägga till andra spelare som vänner och sedan bjuda in dem i spelomgångar. Sådan funktionalitet skulle tillföra en mer social aspekt av spelet då det skulle göra det betydligt bekvämare att starta nya spel med vänner.

6.1.3 Användarvänlighet

Vi tror att användargränssnittet till största delen är tydligt, även om det behöver arbetas på symbolerna så att de blir tydligare. Under betatestningen noterade vi att betatestarna i början hade svårt att förstå vad de skulle göra och de förstod ofta inte vad målet med spelet var. Hur de faktiskt gör dessa saker hittade de däremot i allmänhet lätt efter att spelregler blivit förklarade. Vi drar därför slutsatsen att en tutorial skulle göra mycket för att underlätta spelarnas inlärning. En tutorial skulle kunna förklara det allra mest grundläggande i spelet, nämligen att man bör rekrytera trupper, att man bör flytta dem mot byggnader och ta över dessa och så vidare.

6.1.4 Grafik

Eftersom vi i gruppen själva inte var särskilt grafiskt begåvade, så var det nog ett bra alternativ att ta extern hjälp för att göra grafiken i vår applikation. Vi gav grafikern väldigt fria händer när det gällde utformningen, men gruppen hade fortfarande kontroll och kunde bestämma ungefär hur enheter, byggnader och terräng skulle se ut.

När vi bestämde att spelet skulle ha ett vikingatema, ville vi att applikationens utseende i stort skulle vara någorlunda konsekvent med detta. Därför har knappar och liknande användargränssnittselement ett tydligt vikingatema, alltså att de ser gamla och välanvända ut. Vi tror att detta gör vår applikation mer estetiskt tilltalande vilket i sin tur borde locka en större publik.

6.1.5 Utvecklingsmiljö och ramverk

Användningen av ramverket LibGDX underlättade arbetet avsevärt. Att man lätt kan köra koden på en vanlig dator gör det lätt att utveckla till flera plattformar parallellt och det blir samtidigt enklare att testa den kod man har skrivit, eftersom man slipper att starta upp spelet på telefonen varje gång man vill testa ny kod. LibGDX har också ett färdigt paket för att skriva användargränssnittet som både är kraftfullt och lätt att använda. Vi har dock stött på en del problem med LibGDX, främst med dess integration med FreeType för autogenerering av typsnitt som orsakar krascher på vissa datorer. Dessutom är LibGDX inte särskilt väldokumenterat, något som håller på att åtgärdas [80].

6.2 Metoddiskussion - tillvägagångssätt

Mycket av arbetet i början utgick i helgrupp eftersom det krävdes att hela gruppen samlades för att bestämma hur arbetet skulle läggas upp, samt specificera vad som skulle uppnås och identifiera de problemområden som kommer behöva arbetas med för att nå önskat resultat.

Allt eftersom projektet fortskred och implementeringen kom igång allt mer, så blev det fler och fler arbetsuppgifter som kunde utföras individuellt eller i par. Till viss del skedde arbetet också självständigt, men den största delen av arbetet skedde ändå i helgrupp eller med en majoritet av gruppmedlemmarna närvarande.

Anledningar till det var dels att vi ansåg det lättare att arbeta tillsammans, då man trots olika individuella uppgifter ofta hade vissa frågor eller strukturer i implementeringen som även påverkade andra delar. Genom att arbeta tillsammans så var det lättare att kommunicera och komma överens om hur man skulle lösa dessa problem. Dessutom kände vi att arbete i helgrupp hjälpte vår studiemotivation och bidrog till att man hade lättare att koncentrera sig på uppgiften och bortse från andra distraktionsmoment.

6.2.1 Agil utvecklingsmetod

Nyckelorden för agil utveckling är utveckling, lagarbete och att ständigt vara mottaglig för förändringar rörande projektet. Vi har insett framförallt hur viktigt det är att ständigt kunna vara mottaglig för förändring. Hur bra en idé eller struktur i början av projektet än må vara, så kommer planen förmodligen att behöva modifieras, ändras eller kastas under projektets gång eftersom man inser vilken ny problematik och funktionalitet som finns dold i den ursprungliga uppgiften.

Det är därför mycket viktigt att man inte fäster sig för mycket vid den första idén man har, eftersom detta bara försvårar anpassningen till den nya problemställningen. Vilket även det poängterar vikten av att det finns ett samarbete inom gruppen kring uppgiften, så att man kan enas och komma överens om en ny lösning på den nya problemställning som dykt upp. Det är framförallt viktigt att det inte råder någon rivalitet mellan gruppmedlemmarna och att gruppdynamiken är välfungerande.

Vi har även försökt att använda oss av en arbetsmodell influerad av Scrum, men trots det ändå anpassa den för att bättre passa vårt projekt. I början av projektet märkte vi också att vi oftast lyckades bli klara med det vi skulle enligt tidsplanen snabbare än beräknat. Dels berodde det på att vi underskattat tidsåtgången och avsatt för mycket tid på relativt enkla arbetsuppgifter, men också beroende på att gruppdynamiken i gruppen fungerade väldigt väl och vi hade ett gott arbetstempo. Detta var främst ett problem i de första iterationerna och löstes genom att successivt specificera upp nya arbetsuppgifter.

Trots ett arbetet under projektets inledning redan låg i framkant märkte vi i mitten av projektets gång, att vi började frångå den ursprungliga tidsplanen som gjordes i början av projektet och att vissa delar började bli försenade istället. Detta berodde dels på att komplexiteten i systemet ökat och att vi dels identifierat nya problemuppgifter under arbetets gång som behövt lösas direkt och istället gjort att vi fått skjuta på vissa andra delar i tidsplaneringen. När vi började inse att detta skedde försökte vi dels prioritera ner vissa deluppgifter framför oss, för att kunna fördela mer resurser till de områden som ansågs viktiga och som hamnat något efter tidsplanen.

I efterhand så kan vi inse att den lösningen inte föll ut så väl som vi hoppats på då det saknades en tydlig struktur hur saker skulle prioriteras och vem som skulle göra och ta ansvar för vad. Inför den sista iterationen med fokus på implementationen strukturerades därför arbetet helt efter en prioteringsmodell hämtad från Scrum [59]. I denna modell får alla uppgifter ett värde och en kostnad och de uppgifter med högst värde för (värde/kostnad) prioriteras då att implementeras först. Denna modell var något som vi i projektgruppen tyckte fungerande mycket bra då det gav en tydlig struktur för vad som behöver göras, vem som gör det genom tilldelning av specifika uppgifter, samt tydligt vilka uppgifter och funktionalitet som borde prioriteras bort helt.

6.3 Systemdesignsdiskussion

Den övergripande systemdesignen för att försäkra att alla klienter befinner sig i samma tillstånd bygger på att precis samma instruktioner utförs på alla klienter. Detta syste-

met har visat sig fungera oerhört bra. Det är pålitligt och förutsägbart, även när man lägger till slumpelement i spelet. Förutsättningen för att detta systemet ska fungera är att spelmotorn inte tar någon hänsyn till tid. Det enda som behöver garanteras är att instruktionerna kommer fram och utförs i korrekt ordning, vilket sköts väl med existerande nätverksprotokoll. Systemdesignen har över lag fungerat väldigt bra. De tillfällen då buggar och problem har inträffat har främst varit på grund av antingen bristande dokumentation eller att programmeraren inte läst dokumentationen, vilket har lett till att funktioner använts på fel sätt. En lärdom man kan dra av detta är att även om vi hade dokumentation, så skulle det fungera bättre att vara ännu tydligare med att dokumentationen ska finnas och användas. Detta krävs i ett team på flera personer eftersom olika personer angriper problem på olika sätt, vilket leder till att de inte ser samma naturliga användning av funktioner med givna namn.

6.4 Generaliserbarhet

Tack vare att spelet är skrivet i LibGDX kan det med hjälp av Jogl [38] eller Lwjgl [39] direkt köras på alla plattformar som stödjer Java och OpenGL. Alltså finns stöd för Linux, Windows och OS X utan att någon kod behöver skrivas om. Det kan även med lite arbete portas till en HTML5-backend då LibGDX numera har stöd för detta. Ett problem med en HTML5-version av spelet är dock att eftersom Javascript inte stödjer direkt nätverkskommunikation via sockets så måste spelets nätverkskod modifieras. Det skulle eventuellt kunna lösas med hjälp av WebSocket-protokollet [81] i HTML5, men det är något som behöver undersökas utförligare.

Designen av spelets arkitektur är gjord för att vara generaliserbar. Den grafiska representation av enheter och hexagoner är helt skilt från dess roll i spellogiken. Det skulle därför vara möjligt att med en del mindre modifikationer återvända kärnan till spelet i andra projekt.

Dessvärre har implementationen av menysystemet inte följt någon stark struktur. Framförallt i slutet av projektet togs vissa genvägar för att koppla menyn till nätverksfunktionaliteten för att få klart en beta-version av spelet i tid. Lyckligtvis är detta inget stort problem eftersom menyn är starkt separerad från andra delar av programmet, hela menysystemet går att byta ut. Menysystemet och systemet med popups är tillräckligt skilt från resten av applikationen, att man lätt skulle kunna byta ut resten av applikationen och ändå kunna använda det. På samma sätt kan man bygga ett helt nytt menysystem som ingångspunkt för att köra själva spelet.

Instruktions-modellen går väldigt bra att återanvända för nätverksbaserade spel som inte tar någon hänsyn till tid. Ordningen i vilken instruktioner utförs bevaras. Modellen går dessutom att återanvända, men implementationen kräver specifik information om spelmodellen eftersom instruktionerna beskriver påverkan på spelmodellen.

Implementationen av spelets karta är tydligt separerad från spelmekaniken. Allt som behandlar kartan och det hexagonala rutsystemet är placerat i väldefinierade klasser, och genom att byta ut dessa klasser skulle man kunna byta systemet till ett fyrkantigt rutsystem.

6.5 Framtida arbete

Vengeful Vikings är som spel inte klart. Ett av projektets mål är att publicera Vengeful Vikings på marknaden. Redan från projektets start var planen att detta inte skulle kunna uppnås under projektets gång. Utvecklingen kommer att fortsätta efter projektets slut, och målet är att publicera spelet under sommaren 2012. En större del av framtida arbete är att implementera funktioner som var tänkta att vara med i projektet från början. Dessutom krävs en hel del mer arbete för att nå en viss nivå av kvalitet innan spelet kan släppas på Google Play eller en liknande marknad.

6.6 Ytterligare funktionalitet

Nedan följer några olika punkter med funktionalitet som vi planerar att ha med i framtida versioner av Vengeful Vikings.

Fog of War
Fog of War är ett militärstrategiskt begrepp som beskriver bristen på information om fienden och den omkringliggande terrängen under en militär operation. Detta är ett numera vanligt förekommande koncept i strategispel, som ofta simuleras med en dimma på spelplanen som döljer motståndarens förflyttningar. Att inte kunna se sin motståndares drag, samt att aktivt behöva positionera sina enheter för att hålla uppsikt över fienden, ger ett mer strategiskt djup och leder till mer intressanta taktiker. *Fog of War* är något som togs upp i den senare delen av projektet, men som ansågs kräva en för stor arbetsinstats och därav prioriterades bort.

Nätverk
En spellobby kommer att vara nödvändig i framtiden för att man ska kunna se de listade spelen och gå med i ett spel som ser passande ut. Det skulle även ta bort den omständiga delen med att behöva skriva in spelnamn, som man dessutom behöver veta i förväg, för att kunna gå med i sina vänners spelomgångar.

Framöver behöver det även implementeras en riktig registreringsfunktion, och med fördel funktionalitet för att kunna lägga till vänner och bjuda in dem till sina spelomgångar.

Spelaraktiverade magier
Tidigt i projektet planerades det att ha med magier som inte är kopplat till en viss enhet på kartan utan som spelaren själv kan aktivera. Detta har blivit bortprioriterat på grund av tidsbrist, men det är något som definitivt är ett intressant spelelement i ett senare skede av utvecklingen.

Rutsystem
Något som upptäcktes mot slutet av projektet är att med nuvarande animationsystem där det endast finns en animation för förflyttning och attack med riktning åt öster där

bilderna spegelvänds då enheten går västerut så skulle det vara fördelaktigt att ha ett rutsystem där hexagonerna är stående. Eftersom man med ett sådant rutsystem inte kan gå rakt norr eller söderut kommer man automatisk ifrån problemet att enheten ser ut att gå öster/västerut när man egentligen går norr/söderut.

6.6.1 Begränsningar

Nätverk
Mot den senare delen av projektet gjordes ett medvetet val att tills vidare endast stödja två spelare per spelomgång, i kontrast till grundplanen med valfritt antal spelare. Till följd av detta har kompromisser gjorts i implementeringen av nätverket för att ett få fram ett fungerande tvåspelarläge i tid till projektets avslutande. Vissa delar av nätverksarkitekturen är begränsat till två spelare i olika hög omfattning och en generalisering kommer att kräva en del arbete. Det finns även mindre detaljer som skulle behöva anpassas, bland annat så finns i nuläget inga kartor för fler än två spelare.

En annan nätverksfunktion som för tillfället är ofärdigt är registrering av användare med ett självvalt användarnamn och lösenord. Istället finns en funktion som automatisk generar ett användarkonto och är tänkt att användas för att snabbt kunna komma igång med spelandet, utan att behöva skriva in användaruppgifter. Detta är ingen begränsning i nuläget men det kan i framtiden bli ett problem om det blir aktuellt att koppla ytterligare funktionalitet till ett användarkonto.

Musik
I nuläget har spelet inget ljud förutom en bakgrundsmusik i menyerna, och ljudet är i grundläget avstängt. Det ljud som finns är endast till för att demonstrera att det finns stöd för ljud och musik. Projektgruppen har gjort ett medvetet val att inte lägga in ljud och musik i nuvarande version av spelet, även med tanke på att spelets kvalitet skulle vara bättre med musik och även ljud till enheterna. Ingen inom projektgruppen är heller kunnig inom det området, vilket leder till att eventuell skapad musik inte lär hålla hög kvalitet.

Bra musik lyfter ett spel enormt mycket, men musik som saknar tillräcklig kvalitet skulle bara irritera användare. Det är inte avgjort ifall vi innan publicering kommer skapa musik och ljudeffekter. Däremot skulle det otvivelaktigen sänka spelets upplevda värde ifall spelet släpps på Google Play utan ljud.

Tutorial
Det saknas en tutorial i början av spelet så nya spelare förstår hur spelet fungerar och vad de olika elementen i användargränssnittet gör. Detta skulle kunna implementeras på flera olika sätt. Till exempel genom pilar, text och annan grafik som visar vad allt gör första gången man startar spelet, en inlärningskarta som långsamt låter spelaren lära sig hur spelet fungerar och vad de olika enheterna gör, eller helt enkelt en bild med förklaringar man kan få upp via en menyknapp någonstans i användargränssnittet.

Slumpgenererade kartor
En algoritm för att generera bättre slumpmässiga kartor skulle vara något som skulle öka spelets värde. I nuläget har varje terrängtyp en lika stor sannolikhet att placeras ut, utan att hänsyn tas till den omkringliggande terrängen. En bättre algoritm skulle kunna se till att sammanhängande vattenmassor, bergsområden och skogar placeras ut, samt att byar fördelas jämt över kartan.

Spelmöjligheter
AI:n är något som skulle kunna förbättras en hel del. Med några slumpelement skulle den bli mindre förutsägbar och det skulle inte bli lika lätt att hitta en strategi som fungerar mot den varje gång. Flera olika svårighetsnivåer behövs också, så att nya spelare kan spela mot en lättare AI, medan mer erfarna kan spela mot en svårare.

För att ge mer variation kan kunna ha flera olika spellägen annat än bara erövring. Till exempel ett läge där man ska utföra särskilda uppdrag, exempelvis att plundra byar, eller någon form av överlevnadsläge där den ena spelaren startar i ett sämre läge och ska överleva ett visst antal rundor.

7

Slutsats

Syftet med projektet var huvudsakligen att utveckla ett turordningsbaserat strategispel till Android, vilket har uppnåtts då ett fungerande spel har skapats. Däremot finns fortsatt implementering kvar för ett fulländat spel. Det är närmast ett faktum att det går att lägga till ytterligare funktionalitet när ett projekt är så stort och avancerat som Anstrat faktiskt är. Dessutom innehåller den slutgiltiga versionen för detta projekt en del buggar med varierande påverkan på applikationen. Somliga av buggarna är mindre detaljer som kan skapa vissa tveksamheter hos användaren, exempelvis då användargränssnittet inte fungerar som det ska, medan andra är mer allvarliga. Framförallt inom nätverkskommunikationen finns det buggar som ibland kan få applikationen att haverera.

Att implementera ett komplett system som är helt buggfritt har också visat sig vara i det närmaste omöjligt och alla mer eller mindre stora system anses idag innehålla en del buggar [75]. Systemen släpps ändå på marknaden så länge de allvarligaste buggarna för en majoritet av användarna är lösta. Den slutgiltiga versionen i detta projekt känns dock inte tillräckligt färdig eller buggfri för att kunna släppas på marknaden. Det har också prioriteras andra uppgifter som ansetts vara viktigare än buggfixande under projektet, då tidsrymden som vi haft för arbetet varit begränsad och att buggfixande kan vara mycket komplext. För att eliminera buggar måste man först hitta dem, se under vilka förutsättningar de inträffar och sedan finna en lösning, vilket kan vara en extremt tidskrävande uppgift.

Syftet med projektet var också att lära oss om hur implementering till en färdig produkt går till. Det noterades att det är svårt att planera för hur lång tid implementering av olika funktioner faktiskt tar. Nätverksimplementationen är det som var svårast att uppskatta tidsmässigt, då det finns många olika scenarion som kan skapa problem och som behöver tas hänsyn till. Dessutom har projektgruppens medlemmar aldrig tidigare varit med i ett spelprojekt av denna storlek, vilket gjorde att erfarenhet för att tidsuppskatta projektets alla delområden saknades.

Genom att mycket tid spenderades i början av projektet med att undersöka vilka delar

som behöver ingå i projektet så drogs även många lärdomar om vad som är avgörande för att kunna implementera dessa delar. Därför anser projektgruppens medlemmar att de har lärt sig vilka som är de huvudsakliga delområdena i ett spelprojekt och i framtida projekt vara förmögna att kunna göra bättre uppskattningar av hur stor del av projektet de olika delområdena upptar samt redan från början i större utsträckning fokusera mer på de viktigaste områdena.

Något som projektgruppen ansåg att definitivt bör tillämpas i framtida projekt är den scrum-inspirerade metoden för prioritering, som började användas när det blev ont om tid inför deadline att färdigställa betaversioner av applikationen. Metoden går ut på att identifiera väldefinierade uppgifter, för att sedan inom gruppen gemensamt uppskatta kostnaden att lägga till funktionaliteten, samt värdet av funktionaliteten. Med denna metodik får man för varje uppgift ett värde per kostnad och en ordnad lista av uppgifter efter hur värdefulla de anses. Det leder till att man i implementeringen kan prioritera uppgifter av stort värde och avvakta med de delar som inte bidrar till projektets totala värde i lika stor utsträckning. Inom projektet har detta bidragit till att betaversioner stödjer viktig funktionalitet men tillåts vara bristfälliga i detaljer.

Referenser

[1] W. Bright. (2000) "EMPIRE, Wargame of the Century (tm) A Brief History of Empire". [Online]. Tillgänglig på: http://www.classicempire.com/history.html [2012-05-13].

[2] "Sid Meier's Civilization - Official Site". [Online]. Tillgänglig på: http://www.civilization.com/ [2012-05-10].

[3] (2005, Okt 2) "Battle for Wesnoth 1.0". [Online]. Tillgänglig på: http://www.wesnoth.org/start/1.0/ [2012-05-12].

[4] "Sid Meier's Civilization for Amiga (1992)". [Online]. Tillgänglig på: http://www.mobygames.com/game/sid-meiers-civilization [2012-05-08].

[5] "PoxNora - Wikipedia". [Online]. Tillgänglig på: http://en.wikipedia.org/wiki/Poxnora [2012-05-08].

[6] Nintendo. "Famicom Wars VC". [Online]. Tillgänglig på: http://www.nintendo.co.jp/wii/vc/vc_fw/vc_fw_01.html [2012-05-08].

[7] ——. "Game Boy Advance Wars". [Online]. Tillgänglig på: http://www.nintendo.co.jp/n10/sw2001/softlist/gba/gbwars/ [2012-05-08].

[8] M. Hall. (2009, Mars 4) "Battle for Mars". [Online]. Tillgänglig på: http://larvalabs.com/blog/android/battle-for-mars/ [2012-05-08].

[9] C. Pettey. (2011, Maj 19) "Gartner Says 428 Million Mobile Communication Devices Sold Worldwide in First Quarter 2011, a 19 Percent Increase Year-on-Year". [Online]. Tillgänglig på: http://www.gartner.com/it/page.jsp?id=1689814 [2012-05-12].

[10] C. Pettey and L. Goasduff. (2011, Aug 11) "Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent". [Online]. Tillgänglig på: http://www.gartner.com/it/page.jsp?id=1764714 [2012-05-10].

[11] C. Pettey. (2011, Nov 15) “Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent”. [Online]. Tillgänglig på: http://www.gartner.com/it/page.jsp?id=1848514 [2012-05-10].

[12] ——. (2012, Feb 15) “Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth”. [Online]. Tillgänglig på: http://www.gartner.com/it/page.jsp?id=1924314 [2012-05-10].

[13] Apple. “iOS Developer Program - Apple Developer”. [Online]. Tillgänglig på: https://developer.apple.com/programs/ios/ [2012-05-10].

[14] Google. “Platform Versions | Android Developers”. [Online]. Tillgänglig på: http://developer.android.com/resources/dashboard/platform-versions.html [2012-05-10].

[15] “Swedish Game Awards 2012”. [Online]. Tillgänglig på: http://gameawards.se/ [2012-05-11].

[16] J. Kleinberg and É. Tardos, *Algorithm Design*. Boston: Pearson Education, 2006.

[17] Wikipedia. “Chebyshev distance - Wikipedia, the free encyclopedia”. [Online]. Tillgänglig på: http://en.wikipedia.org/wiki/Chebyshev_distance [2012-05-14].

[18] C. P. Birch, S. P. Oom, and J. A. Beecham, “Rectangular and hexagonal grids used for observation, experiment and simulation in ecology,” *Ecological Modelling*, vol. 206, no. 3–4, pp. 347 – 359, 2007. [Online]. Tillgänglig på: http://www.sciencedirect.com/science/article/pii/S0304380007001949

[19] BlackBerry. “BlackBerry Smartphones UI Guidelines Version: 6.0”. [Online]. Tillgänglig på: http://docs.blackberry.com/en/developers/deliverables/17964/BlackBerry_Smartphones-UI_Guidelines-T893501-980426-0721013746-001-6.0-US.pdf [2012-05-10].

[20] Google. “What is Android | Android Developers”. [Online]. Tillgänglig på: http://developer.android.com/guide/basics/what-is-android.html [2012-05-10].

[21] ——. “Application Fundamentals | Android Developers”. [Online]. Tillgänglig på: http://developer.android.com/guide/topics/fundamentals.html [2012-05-10].

[22] ——. “Android NDK | Android Developers”. [Online]. Tillgänglig på: http://developer.android.com/guide/topics/graphics/index.html [2012-05-10].

[23] Xamarin. “Mono for Android - Create amazing Android apps with C# and .NET.”. [Online]. Tillgänglig på: http://xamarin.com/monoforandroid [2012-05-10].

[24] ——. “Mono”. [Online]. Tillgänglig på: http://www.mono-project.com [2012-05-10].

[25] D. Ehringer. (2010, Mar) "The Dalvik Virtual Machine Architecture". [Online]. Tillgänglig på: http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf [2012-05-10].

[26] T. Bray. (2010, Maj 25) "Dalvik JIT | Android Developers Blog". [Online]. Tillgänglig på: http://android-developers.blogspot.se/2010/05/dalvik-jit.html [2012-05-10].

[27] Google. "Designing for Performance | Android Developers". [Online]. Tillgänglig på: http://developer.android.com/guide/practices/design/performance.html#internal_get_set [2012-05-10].

[28] ——. "Graphics | Android Developers". [Online]. Tillgänglig på: http://developer.android.com/guide/topics/graphics/index.html [2012-05-10].

[29] The Khronos Group. "OpenGL Overview". [Online]. Tillgänglig på: http://www.opengl.org/about [2012-05-10].

[30] ——. "OpenGL ES - The Standard for Embedded Accelerated 3D Graphics". [Online]. Tillgänglig på: http://www.khronos.org/opengles/ [2012-05-10].

[31] Google. "OpenGL | Android Developers". [Online]. Tillgänglig på: http://developer.android.com/guide/topics/graphics/opengl.html [2012-05-10].

[32] ——. "playN - Cross platform game library for N $\geq$ 4 platforms". [Online]. Tillgänglig på: http://code.google.com/p/playn [2012-05-10].

[33] ——. "DemoLinks - playn - Links to PlayN games and demos - Cross platform game library for N$\geq$4 platforms". [Online]. Tillgänglig på: http://code.google.com/p/playn/wiki/DemoLinks [2012-05-10].

[34] M. Zechner. "Bad Logic Games ■ About". [Online]. Tillgänglig på: http://www.badlogicgames.com/wordpress/?page_id=2 [2012-05-08].

[35] ——, *Beginning Android games*. New York: Apress, 2011.

[36] ——. (2012, Mars 12) "LibGDX goes HTML5". [Online]. Tillgänglig på: http://www.badlogicgames.com/wordpress/?p=2308 [2012-05-10].

[37] Google. "Google Web Toolkit Overview - Google Web Toolkit – Google Developers". [Online]. Tillgänglig på: https://developers.google.com/web-toolkit/overview [2012-05-10].

[38] "JOGL - Java Binding for the OpenGL API". [Online]. Tillgänglig på: http://jogamp.org/jogl/www/ [2012-05-03].

[39] "lwjgl.org - Home of the Lightweight Java Game Library". [Online]. Tillgänglig på: http://www.lwjgl.org/ [2012-05-03].

[40] U. Technologies. “UNITY: Unity 3 Engine Features”. [Online]. Tillgänglig på: http://unity3d.com/unity/engine/ [2012-05-10].

[41] ——. “UNITY: Programming - Unity 3 Engine Features”. [Online]. Tillgänglig på: http://unity3d.com/unity/engine/programming [2012-05-10].

[42] ——. “UNITY: Store”. [Online]. Tillgänglig på: https://store.unity3d.com/products [2012-05-03].

[43] “AndEngine - Free Android 2D OpenGL Game Engine”. [Online]. Tillgänglig på: http://www.andengine.org/ [2012-05-10].

[44] “cocos2d-android - cocos2d for Android: A framework for building 2D games for the Android platform”. [Online]. Tillgänglig på: http://code.google.com/p/cocos2d-android/ [2012-05-10].

[45] I. Millington, *Artificial Intelligence for Games*. San Francisco: Morgan Kaufmann Publishers, 2006.

[46] D. Lindh, “Adaptive combat ai in strategy games - an approach based on dynamic scripting,” Master’s thesis, Lunds Universitet, Lund, 2008.

[47] Ahlström, Johannes, “Artificiell intelligens i spel,” [2012-05-13]. [Online]. Tillgänglig på: http://www.ida.liu.se/~729G11/projekt/studentpapper-08/johannes-ahlstrom.pdf

[48] WPI. “Next Basic AI Technique: Scripting”. [Online]. Tillgänglig på: http://web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/C-Scripting.pdf [2012-05-10].

[49] J. Freeman-Hargis. (2012, Maj 10) “Introduction to Rule-Based Systems”. [Online]. Tillgänglig på: http://ai-depot.com/Tutorial/RuleBased.html [2012-05-10].

[50] P. Spronck, M. Ponsen, E. Postma, and I. Sprinkhuizen-Kuyper, “Special Issue: Machine Learning and Games,” in *Machine Learning*. Kluwer, 2006, pp. 217–248.

[51] Encyclopædia Britannica, Inc. “client-server architecture (computer science) – Britannica Online Encyclopedia”. [Online]. Tillgänglig på: http://www.britannica.com/EBchecked/topic/1366374/client-server-architecture [2012-05-12].

[52] N. Krishnan. (2001, Okt.) “The Jxta solution to P2P - JavaWorld”. [Online]. Tillgänglig på: http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html [2012-05-08].

[53] (2010, Mar.) “Peer-To-Peer and The Effects On Network Performance”. [Online]. Tillgänglig på: http://www.webexploits.co.uk/2010/03/peer-to-peer-and-effects-on-network_02.html [2012-05-08].

[54] QuinStreet Inc. "All About Peer-To-Peer Architecture - Webopedia.com". [Online]. Tillgänglig på: http://www.webopedia.com/DidYouKnow/Internet/2005/peer_to_peer.asp [2012-05-12].

[55] *TRANSMISSION CONTROL PROTOCOL*, University of Southern California Std. 793, 1981.

[56] J. Postel and J. Reynolds, *FILE TRANSFER PROTOCOL (FTP)*, Network Working Group Std. 959, 1985.

[57] J. Postel, *User Datagram Protocol*, Std. 768, 1980.

[58] R. Hunicke, M. LeBlanc, and R. Zubek. (2004) "Introduction to Rule-Based Systems". [Online]. Tillgänglig på: http://www.cs.northwestern.edu/~hunicke/pubs/MDA.pdf [2012-05-12].

[59] Scrum.org. "Scrum Official Webpage". [Online]. Tillgänglig på: http://www.scrum.org/ [2012-05-11].

[60] Ciber. "All About Peer-To-Peer Architecture - Webopedia.com". [Online]. Tillgänglig på: http://www.ciber.se/pdf/Krav%20-%20behover%20vi%20dem.pdf [2012-05-12].

[61] Sogeti. "Fördelar med agil utveckling". [Online]. Tillgänglig på: http://www.sogeti.se/Vara-tjanster/Hogaktuella-tjanster/Agila-metoder-och-Scrum/ [2012-05-12].

[62] "Battle for Wesnoth". [Online]. Tillgänglig på: http://www.wesnoth.org/ [2012-03-16].

[63] Sony Online Entertainment. "PoxNora - Welcome to PoxNora". [Online]. Tillgänglig på: http://poxnora.station.sony.com/index.do [2012-03-16].

[64] D. Adams, S. Butts, and C. Onyet. (2007, Mar.) "Top 25 PC games of all time, IGN (2007)". [Online]. Tillgänglig på: http://pc.ign.com/articles/772/772285p3.html [2012-05-08].

[65] Larva Labs. "Larva Labs - Battle for Mars for Android". [Online]. Tillgänglig på: http://larvalabs.com/product_detail.php?app=25 [2012-03-16].

[66] hbwares. "WordFeud - multiplayer game for iPhone, Android and Windows Phone". [Online]. Tillgänglig på: http://wordfeud.com/ [2012-03-16].

[67] Google. "Wordfeud - Android Apps on Google Play". [Online]. Tillgänglig på: https://play.google.com [2012-05-12].

[68] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of reusable Object-Oriented software*, 1st ed. Reading, MA: Addison-Wesley Professional, 1994.

[69] Unity Technologies. "UNITY: System Requirements". [Online]. Tillgänglig på: http://unity3d.com/unity/system-requirements [2012-05-10].

[70] V. Davis. (2006, Mar. 13) "Forgotten Lore ■ Blog Archive ■ Hexing Matters". [Online]. Tillgänglig på: http://www.crypticcomet.com/blog/?p=9 [2012-05-08].

[71] I. Her, "Geometric transformations on the hexagonal grid," *Image Processing, IEEE Transactions on*, vol. 4, no. 9, pp. 1213 –1222, sep 1995.

[72] R. Zerega Bravo and B. Lazarov, "To touch or not to touch : A comparison between traditional and touchscreen interface within personal computers," p. 64, 2011.

[73] B. Shneiderman and C. Plaisant, *Designing the user interface : strategies for effective human-computer interaction.* Upper Saddle River, N.J: Pearson/Addison Wesley, 2004.

[74] Oracle. "Remote Method Invocation Home". [Online]. Tillgänglig på: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html [2012-05-12].

[75] ——. "Java Sockets". [Online]. Tillgänglig på: http://docs.oracle.com/javase/tutorial/networking/sockets/ [2012-05-12].

[76] PostgreSQL Global Development Group. "PostgreSQL: Welcome". [Online]. Tillgänglig på: http://www.postgresql.org/ [2012-05-13].

[77] E. Welch. "Designing AI Algorithms For Turn-Based Strategy Games". [Online]. Tillgänglig på: http://www.gamasutra.com/view/feature/129959/designing_ai_algorithms_for_.php [2012-05-10].

[78] K. Burgun. (2011, Nov.) "UI Tips for iOS & Android Developers". [Online]. Tillgänglig på: http://www.dinofarmgames.com/ui-tips-for-ios-android-developers/ [2012-05-10].

[79] Qulturum. (2010) "PGSA-hjulet - instruktion". [Online]. Tillgänglig på: http://www.lj.se/info_files/infosida31713/pgsa_instruktion.pdf [2012-05-10].

[80] M. Zechner. (2012, Maj) "LibGDX documentation initiative". [Online]. Tillgänglig på: http://www.badlogicgames.com/wordpress/?p=2411 [2012-05-10].

[81] A. Melnikov and I. Fette, *The WebSocket Protocol*, IETF Std. 6455, 2011.

A

Appendix

A.1 Bilaga A - Ordlista

Action points: Poäng som används för att utföra olika typer av handlingar.

AP: Se *Action Points*.

Arkitektur: Definition för hur olika delar av mjukvara fungerar och hur de relaterar till andra delar.

Balansering: Modifiering av värden för att det ska vara rättvist och jämlikt.

Desktop: Avser platformen desktop, här definerad som en dator där prestandan inte är lika begränsad som på ett inbyggt system, så som en smartphone eller läsplatta.

Eclipse: En utvecklingsmiljö - textredigerare och kompilator med stöd för en mängd programmeringsspråk, men oftast förknippad med Java-programmering.

Enhet: Synonym för 'spelpjäs'.

Flash: Adobe Flash, ett program för att spela upp ljud och film samt skapa interaktiva webbplatser.

FreeType: Ett fontrastiseringssystem.

FTP: File Transfer Protocol, ett protokoll ovanpå TCP/IP-lagret som används för att överföra filer (oftast större sådana).

Google Play: En onlinebutik skapad av Google där användare kan ladda ner applikationer till enheter som kör Android, tidigare känt som Android Market. Jämförbart med App Store för iOS.

Guld: Spelets valuta. Används för att rekrytera nya enheter.

Heuristik: En matematisk term som beskriver att man gör 'smarta gissningar' för att lösa ett problem. En heuristisk algoritm garanterar inte en korrekt eller optimal lösning.

Hotseat: Flerspelarläge för användare på samma enhet.

Instruktion: En enskild handling som förändrar speltillståndet. Typiska instruktioner är att flytta en enhet, anfalla en fiende eller lämna över turen till nästa spelare. Att exempelvis markera enheter är inte instruktioner. Synonymer är 'drag', 'handling', 'kommando' och kodtekniskt 'Command'.

Iteration: Här synonym för 'sprint'.

Klient: Programmet som körs på telefonen.

Mana: En resurs i spelet. Används för att kasta spelaraktiverade magier.

Nonce: Ett godtyckligt valt tal som används för ett kort syfte, vanligtvis någon form av identifiering eller verifiering. (A digit **'n'** to be used **once**).

Pinch: Rörelsen på en pekskärm där två fingrar dras från eller mot varandra. Används vanligtvis för zoom för de program som kräver funktionalitet för zoom.

Runda: En lista av instruktioner från det första man är tillåten att göra, till instruktionen att lämna över turen till motståndaren.

Socket: En ändpunkt i nätverkskommunikationen mellan två datorer. En socket identifieras med en IP-address samt ett portnummer.

Speldesign: Ett mer utförligt koncept för ett spel. Innefattar spelmekanik och tema.

Spelidé: Ett grundläggande koncept för ett spel.

Spelkoncept: Synonym till spelidé.

Spelmekanik: Regler för hur spelet fungerar.

Spelomgång: Ett spel från det att två eller fler personer börjar spela mot varandra till det att en person vinner spelet.

Sprint: Definierat av utvecklingsmetoden Scrum [55]. Utvecklingsfas som sträcker sig ett fördefinierat tidsspann, med mål för vad som ska vara färdigställt i slutet av tidsspannet. I detta projekt är en sprint och iteration av samma längd.

Timeout: I nätverkssammanhang, en tidsbegränsning som behövs då förfrågningar eller anslutningar som inte erhållit svar på en längre tid förmodligen har stött på fel och bör termineras. Även tillämpligt för att identifiera inaktivitet.

Tutorial: En introduktion för upplärning och vägledning till nya spelare.

A.2 Bilaga B - Tidsplanering