CHALMERS

GÖTEBORGS UNIVERSITET

Experimentell spelmekanik

- Realtidsaction möter turbaserad strategi

Kandidatarbete inom Data och Informationsteknik

Andreas Wahlström
Anthony Rizk Gustafsson
Johan Wermensjö
Johan Häggström
Olof Karlsson
Robert Wennergren

Chalmers tekniska högskola
Göteborgs universitet
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, 2 juni 2015

Experimentell spelmekanik
- Realtidsaction möter turbaserad strategi

Andreas Wahlström
Anthony Rizk Gustafsson
Johan Häggström
Johan Wermensjö
Olof Karlsson
Robert Wennergren

Examiner: Jan Skansholm

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: the logotype of the game, depicting a bound tome with the words “Demon Lore” inscribed in red gothic writing]

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Abstract

In the gaming industry, it has long been popular to combine genres with the aim of creating games that bring together the best of several worlds. However, there are still many unexplored combinations with good potential.

This report addresses the planning and development of a game prototype that combines two game genres, turn-based and real-time strategy. In addition to presenting the results, the report also discusses the difficulties that were encountered and how they were handled.

The prototype contains a working combat system and a basic AI. The prototype is modular, which means that it is easy for an end user to extend the game with extra content. Most elements that were considered important from the two genres were implemented, but there is still room for improvement and further development.

The result is evaluated by means of quality assurance, a process that is commonly used in the gaming industry. The response has been generally positive, which indicates that the combination has potential and should be explored further.

Sammanfattning

I spelindustrin har det länge varit populärt att kombinera genrer, med syfte att skapa spel som samlar det bästa av flera världar. Det finns dock fortfarande många outforskade kombinationer med god potential.

Den här rapporten tar upp planeringen och utvecklingen av en spelprototyp som kombinerar de två spelgenrerna turbaserad- och realtidsstrategi. Utöver en presentation av resultatet diskuteras även de svårigheter som stöttes på och hur de hanterades.

Prototypen innehåller ett fungerande stridssystem och en grundläggande AI. Prototypen är modulärt uppbyggd, vilket innebär att det är enkelt för en slutanvändare att utöka spelet med extra innehåll. De flesta moment som ansågs viktiga från de två genrerna blev implementerade, men det finns fortfarande rum för förbättring och vidareutveckling.

Resultatet utvärderas med hjälp av användartester, en process som är vanlig i spelindustrin. Responsen har varit allmänt positiv vilket pekar på att kombinationen har potential och bör utforskas vidare.

Förord

Rapporten behandlar ett kandidatarbete inom Data- och Informationsteknik och genomfördes som ett samarbete mellan studenter på Chalmers Tekniska Högskola och Göteborgs Universitet. Kandidatarbetet utfördes under en termin och omfattar 15 hp.

Projektgruppen önskar tacka Alexander Sjösten som handledde oss genom projektet, samt de personer som deltog i användartesterna. Projektgruppen önskar även tacka Claes Ohlsson och Hans Malmström på Fackspråk för deras språkliga expertis och vägledning. Avslutningsvis önskar vi även tacka hästgruppen, vår kaffekopp i soluppgången, ledstjärna i mörkret och parkeringsplats i hamnen.

Ordförklaring

1. **AI, Artificiell intelligens**

   En gren inom datavetenskapen där syftet är att få program att bete sig intelligent. 5

2. **AP, Action Points**

   En resurs som begränsar mängden handlingar en enhet kan utföra per tur. 14

3. **Attribut**

   Representerar en egenskap hos en enhet. 2

4. **Backlog**

   En lista av uppgifter och funktionalitet som ska implementeras, sorterad efter prioritet. 8

5. **Branch**

   En fristående utvecklingsgren, se [1]. 25

6. **Cooldown**

   Speluttryck för att beskriva tiden en förmåga är otillgänglig efter användning. 15

7. **Enhet**

   En karaktär i spelet. 1

8. **Förmåga**

   En handling som utförs av en enhet för att påverka enheter eller omgivningen. 2

9. **GUI, Graphical User Interface**

   Engelska för grafiskt användargränssnitt. Består av grafiska komponenter som ger spelaren möjlighet att interagera med spelet. 11

10. **JVM, Java Virtual Machine**

    Ett system som tillåter javakod att vara plattformsoberoende. 6

11. **MVC, Model-View-Controller**

    Engelska för Modell-Vy-Kontrollenhet. Ett designmönster inom systemutveckling som går ut på att dela upp programmet i tre delar utifrån olika ansvarsområden. 9

12. **Pathfinding**

    Att hitta kortaste eller minst kostsamma vägen från punkt A till punkt B. 26

13. **Paus-läge**

    Fryst speltillstånd där inga enheter rör sig eller attackerar. 2

14. **Realtid**

    Innebär att tid i spelet fortlöper med konstant tempo. 1

15. **RPG, Role-playing game**

    Engelska för rollspel. Spelgenre där spelarkaraktärer blir starkare och får nya färdigheter under spelets gång. 1

16. **RTS, Real-time strategy**

    Engelska för realtidsstrategi. Spelgenre med högt tempo där spelaren kontrollerar många enheter samtidigt. 1

17. **Rutnätsbaserat**

    Positioneringssystem där objekt är placerade i ett två-dimensionellt rutnät. 3

18. **Scrum**

    Iterativ arbetsmetodik för systemutveckling. 8

19. **Scrum Master**

    En person som ansvarar för att arbetet utförs i enlighet med Scrum. 8

20. **Spelbalans**

    Hur varje sammansättning av enheter och förmågor står i relation till varandra med avseende på styrka. 23

21. **Spelinnehåll**

    Alla kartor, enheter, sprites, ljudeffekter och musik som finns i spelet. 1

22. **Spelmekanik**

    Hur spelet utför de spelregler som spelet styrs av. 1

23. **Sprite**

    En bild som används för att visuellt representera objekt i spelet. 7

24. **Statuseffekt**

    En effekt som påverkar enhetens egenskaper. 14

25. **Stridsstil**

    En stridsstil ändrar en enhets egenskaper och förmågor, exempelvis offensiv eller defensiv. 14

26. **TBS, Turn-based strategy**

    Engelska för turbaserad strategi. Spelgenre där spelarna turas om att agera. 1, 11

27. **Tileset**

    Ett rutnät av bilder som representerar den grafik som används av en karta. 7

28. **Tooltip**

    En beskrivande text som visas då musen är placerad över ett grafiskt objekt. 16

29. Tur

Avser den tidsperiod då enheter utför sina ordrar och spelare kan beordra nya. 1

30. Turklocka

En klocka som visar hur lång tid det är kvar på den nuvarande turen. 15

Innehållsförteckning

1. Inledning

Med målsättningen att undersöka hur en sammanslagning av moment ifrån två olika typer av strategispel fungerar i praktiken, har projektgruppen utvecklat ett spel vid namn Demonlore. Spelet är utvecklat som en prototyp för att undersöka hur väl sammanslagningen fungerar. Rapporten behandlar bakgrunden till spelets utveckling, en analys av spelmekaniska problem samt hur spelet utvecklats från idé till produkt. Vidare ges en redovisning av implementationen, följt av en avslutande diskussion och utvärdering av resultatet. Den avslutande delen behandlar de metoder som använts, huruvida resultatet uppfyllde syftet och hur konceptet kan vidareutvecklas.

1.1 Bakgrund

I den moderna spelindustrin produceras idag spel inom flera olika genrer. En metod för utveckling av nya typer av spel är att blanda inslag från flera genrer, vilket kan ge upphov till unika och innovativa spel som för spelindustrin framåt.

En viktig faktor att ta hänsyn till vid en blandning av spelgenrer är tempot. En jämn speltakt är viktig för att spelet inte ska kännas ryckigt, vilket kan uppstå vid en blandning av en långsammare och en snabbare genre. Om det inte hanteras väl riskerar spelet att upplevas som enerverande.

Två genrer med god potential för kombination är realtidsstrategi och turbaserad strategi. Genrerna kan komplettera varandra på ett gynnsamt sätt då det höga tempot från realtidsspel förenat med de strategiska aspekterna från turbaserade spel kan ge djup och intensitet.

Inom turbaserade strategispel, som exempelvis Fire Emblem [2], har spelaren kontroll över en grupp enheter där varje enhet har mer eller mindre unika strategiska egenskaper. Spelaren har gott om tid att utföra sina drag och när en spelare är klar går turen vidare till nästa spelare, likt i många klassiska brädspel [3].

I realtidsstrategispel, som exempelvis Starcraft [4], finns däremot inga turer utan alla spelare utför sina beslut och handlingar samtidigt. Då det inte alltid finns tid för planering måste beslut tas omedelbart, nästan reflexmässigt, vilket leder till att spelet blir mer intensivt och får ett högre tempo [3].

Den stora utmaningen med att få kombinationen av genrer att fungera är hur spelaren ska kunna känna att hen har kontroll över vad som sker i spelet. Hindret uppkommer när spelarens avsikt att ta optimala beslut hamnar i direkt konflikt med det högre tempot. Spel som tidigare har experimenterat med den här kombinationen av genrer är bland annat Baldur's Gate [5] och Planescape: Torment [6]. Även om spelen går under genren rollspel, härstammar mycket av spelmekaniken från realtidsstrategi och turbaserad strategi. De nämnda spelen har bidragit till intresset för att fortsätta utveckla den här kombinationen av spelmekaniker i hopp om att skapa nya typer av spel som hanterar de strategiska inslagen bättre.

1.2 Syfte

Syftet är att undersöka potentialen i kombinationen av de strategiska momenten från turbaserade rollspel och intensiteten från realtidsspel. Undersökningen sker genom utvecklingen av en spelprototyp, vilken utformas på ett sätt som känns intuitivt för användaren. Fokus ligger på användarvänlighet samt att övergången mellan olika spelmoment upplevs

naturlig. Spelet ska vara modulärt och användare ska kunna utöka spelet med eget spelinnehåll.

1.3 Spelegenskaper

För att uppnå samma strategiska kontroll som i ett turbaserat spel delas spelet upp i tidsbegränsade turer. Uppdelningen medför att spelaren har full kontroll över sina enheter men med begränsad mängd tid per tur. Tidsbegränsningen är till för att det ska vara svårt att ta optimala beslut för varje enhet, vilket tvingar spelaren att besluta var hen vill lägga sitt fokus under större strider.

För att ge spelaren en bättre överblick samt möjlighet att tänka över sina beslut implementeras också en pausfunktion, som fryser spelets gång men fortfarande tillåter spelaren att ge order till sina enheter.

Spelaren har möjligheten att kontrollera ett flertal enheter. Likt rollspel definieras enheterna av sina attribut samt äger speciella förmågor som kan användas av spelaren [7].

1.4 Avgränsningar

För att se till att syftet uppfylldes i så stor utsträckning som möjligt avgränsades några aspekter till viss mån från utvecklingen av spelet.

1.4.1 Målplattform

Spelet utvecklades för Windows, Linux och OS X. Stöd för mobila plattformar som Android och iOS övervägdes, dock behövde då stöd för pekskärmar tagits i åtanke under utvecklingen, vilket hade tagit viktig utvecklingstid från övriga områden.

1.4.2 Grafik

Spelet har relativt enkel 2D-grafik då det, för syftet, var mer relevant att lägga tid på implementering av spelmekaniken. Grafisk tydlighet var dock av stor vikt då grafiken inte skulle orsaka förvirring hos spelaren, utan istället hjälpa till att skapa en tydlig bild av händelseförloppet. Den grafiska fokusen låg därför på återkoppling och användargränssnittet, inte på avancerade effekter.

1.4.3 Handling

Spelet har ingen form av övergripande handling, eftersom det ej var nödvändigt för att undersöka hur väl konceptet fungerar. Dock har varje spelare minst ett uppdrag som måste genomföras för att hen ska vinna spelet.

1.4.4 Spelinnehåll

Eftersom fokus lades på själva spelmekaniken lades inget arbete på att producera en större mängd spelinnehåll. Mängden spelinnehåll har förvisso stor inverkan på spelets livslängd och variation, men med projektets syfte i åtanke ansågs det viktigare att spelmekaniken fungerade bra.

1.4.5 Flerspelarläge

I spelet kan spelaren slåss mot en eller flera AI-motståndare. Många strategispel innehåller även ett flerspelarläge, där människor kan spela mot varandra antingen lokalt eller över nätverk. Det skulle dock vara ett väldigt stort arbete att implementera ett flerspelarläge och även om det hade varit fördelaktigt var det inte nödvändigt för att uppfylla syftet.

1.5 Presentation av genrer

Det här avsnittet tar upp det som kännetecknar de valda genrerna: turbaserad strategi och realtidsstrategi, samt hur de skiljer sig från varandra. Avsnittet behandlar de utmaningar som kan uppstå vid en kombination av genrerna.

1.5.1 Turbaserad strategi

Det som definierar ett turbaserat spel är att varje spelare turas om att agera. Under spelarens tur får hen möjlighet att ta beslut utan att någon annan kan påverka spelet. När spelaren är nöjd med sina beslut överlåts turen till nästa spelare i turordningen. Ett enkelt exempel på ett turbaserat strategispel är schack som ligger till grund för många datorspel inom genren.

Moderna turbaserade strategispel skiljer sig dock från schack på flera olika sätt, exempelvis kan en spelare oftast flytta flera enheter under sin tur. Precis som i schack begränsas dock enheternas rörelse oftast av ett rutnät. Spelplanen består oftast av olika typer av terräng som kan ha spelmekaniska egenskaper, exempelvis att enheter inte kan röra sig genom terrängen. Terrängtyperna medför att varje enhets position på spelplanen har stor betydelse i ett turbaserat strategispel [8].

1.5.2 Realtidsstrategi

Inom realtidsstrategi existerar inte konceptet med turer. Alla spelare agerar samtidigt, vilket leder till hektiska moment där spelarna måste kontrollera flera saker samtidigt [3]. Strategimomenten i den här typen av spel innefattar bland annat att ta snabba beslut och förutse vad andra spelare kommer att göra. Spelare kontrollerar oftast många enheter samtidigt, vilket gör det svårare att utnyttja varje enhet på ett optimalt sätt [8].

2. Problemområden

Det här kapitlet presenterar det som ansågs vara de viktigaste eller svåraste problemområdena inom utvecklingen av spelet. Då problemen uppstod som en direkt konsekvens av syftets utformning var de tvungna att lösas för att slutprodukten skulle anses vara fullständig.

2.1 Balans mellan turer och realtid

Kombinationen av realtids- och turbaserade spel kan orsaka problem. Båda genrerna har märkbart olika tempon, vilket medför svårigheter med att få spelets tempo att upplevas naturlig. Spel som kombinerar de båda genrerna skiljer dem ofta åt genom att byta mellan olika spellägen beroende på vad som utspelas, som i Total War-serien [9], där strider sker i realtid medan förflyttning på kartan utförs i den turbaserade delen. För att på ett lyckat sätt kombinera de två genrerna behövdes beslut tas gällande var gränsen mellan de båda skulle dras.

2.2 Intuitivt att spela

För att uppfylla kriterierna för intuitivitet ska ett spel vara enkelt att förstå och sätta sig in i. När någonting händer ska spelaren omedelbart vara medveten om vad som hänt för att snabbt kunna reagera. Intuitiviteten kan uppnås genom att signalera om vad som händer eller kommer att hända i spelet och kallas för återkoppling. Det kan ske både grafiskt och med hjälp av ljudeffekter. Ett exempel på återkoppling är att musmarkören byter grafik då den placeras över något som spelaren kan interagera med, se figur 2.1.

Figur 2.1: Musmarkören (positionerad över fiendesnigeln) indikerar att snigeln kan attackeras

Då spelet skulle blanda spelmoment från två konceptuellt olika genrer, behövde spelarna snabbt kunna förstå hur kombinationen fungerade. Spelet skulle troligen upplevas som irriterande om spelaren inte snabbt förstod det grundläggande upplägget, vilket kunde förhindra vidare utforskning av kombinationens potential. Spelet var även tänkt att vara tilltalande för de med liten eller ingen erfarenhet av liknande spel, men ändå engagerande för de mer erfarna spelarna, vilket ställde ytterligare krav på spelets intuitivitet. För att tilltala en större publik krävdes en plan inlärningskurva, vilket ställde stora krav på framförallt användargränssnittet som skulle vara både enkelt och funktionellt. Att skapa spel med så jämn inlärningskurva som möjligt utan att offra spelkomplexitet och innehåll är en stor utmaning och ansågs därför vara en av de största problemområdena med utvecklingen.

2.3 Modularitet och utökningsmöjligheter

För att det ska vara enkelt att utöka ett spel med mer innehåll och funktionalitet, är det av stor vikt att det är modulärt. Ur ett programmeringsperspektiv är modulära enheter att föredra då det drastiskt minskar beroendet mellan olika komponenter och undviker hårdkodning av data. Minskat beroende mellan komponenter underlättar vidareutveckling då ny funktionalitet kan implementeras i nya moduler som enkelt kan anslutas till systemet [10].

Eftersom spelet skulle utvecklas som en prototyp var det viktigt att det fanns goda möjligheter för vidareutveckling, vilket gjorde det lättare att testa konceptets begränsningar och möjligheter.

2.4 Artificiell intelligens

I ett enspelarspel där spelaren kämpar mot datormotståndare är det oftast mycket viktigt att motståndet är någorlunda intelligent. Om datormotståndaren har en bristande Artificiell intelligens (AI) kan det leda till att spelet upplevs som för enkelt, men om AI:n istället är för intelligent kan det uppfattas som att den fuskar [11]. Det är därför mycket viktigt att spelets AI upplevs som rättvis och underhållande. Det här krävde att AI:n utvecklades med en struktur som gjorde den enkel att justera under utvecklingens gång.

3. Utvecklingsverktyg

I avsnittet redovisas val av versionshanteringsprogram, programmeringsspråk och andra tekniska hjälpmedel som använts, samt motiveringen bakom varför de kom att användas.

3.1 Versionshantering, Git och GitHub

I större projekt som både fortgår under flera månader samt utförs av flera personer, behövs ett enkelt och effektivt sätt att dela arbetet inom gruppen. För att hantera det användes ett så kallat versionshanteringssystem för att varje utvecklare skulle vara i fas med projektet. Det system som valdes var Git [12] då det ansågs effektivt, välanvänt och väldokumenterat.

För att dela kod mellan flera användare via Git krävs en fjärrserver där en kopia av arkivet lagras [13]. Användarna kan sedan hämta eller uppdatera filerna på servern. Istället för att sätta upp en egen server finns en mängd webbtjänster som både är enkla och kostnadsfria, som Github [14] och Bitbucket [15]. Projektgruppen valde att använda GitHub då den erbjuder privata arkiv till studenter [16].

3.2 Java

Det programmeringspråk som valdes var Java, vilket var ett självklart val då hela projektgruppen hade tidigare erfarenheter av språket. Det finns även många bibliotek för spelutveckling i Java [17][18], exempelvis LibGDX [19], vilket är det bibliotek som användes för utvecklingen av spelet.

Java är plattformsoberoende, vilket möjliggör utveckling till flera plattformar utan att behöva anpassa programkoden på något sätt. Den här flexibiliteten medför att mängden potentiella användare ökar då produkten inte enbart riktar sig till Windows-användare utan sträcker sig även till mindre vanliga operativsystem.

Då problem ofta uppkommer vid programvaruutveckling är det viktigt att det finns god dokumentation och möjlighet till hjälp från andra utvecklare. Java är ett av de största och mest använda programmeringsspråken i världen vilket underlättade utvecklingen avsevärt [20].

Java är som tidigare nämnt plattformsoberoende, vilket uppnås genom användning av en Java Virtual Machine (JVM). JVM tar plattformsoberoende Java-kod och översätter det till plattformsspecifika instruktioner som i sin tur kan tolkas av processorn.

Den virtuella maskinen blir ett ytterligare mellansteg mellan mänskligt läsbar kod och maskinkod. Den här flexibiliteten fås i utbyte mot vissa prestandaförluster [21], vilket kan ha stor påverkan på större projekt där prestanda är i fokus. Prestandaförlusten ansågs dock inte vara något problem för projektet, då spelet som utvecklades inte var beräkningstungt nog för att belasta systemet nämnvärt, vilket innebar att även äldre datorer kan hantera spelet utan större problem.

3.3 LibGDX

LibGDX är ett open-source-bibliotek som underlättar spelutveckling i Java [19]. Biblioteket abstraherar bort många vanliga problem, exempelvis rendering av grafik och hantering av ljud. Biblioteket bygger på ett open-source bibliotek som heter LWJGL

[22]. LibGDX är välutvecklat och håller relativt hög standard. Då biblioteket har öppen källkod med en licens som tillåter modifiering av koden [23], utvecklas och uppdateras biblioteket frekvent.

LibGDX ger även ytterligare flexibilitet då biblioteket stödjer utveckling för flera plattformar samtidigt. De plattformar som stöds är Windows, Linux, OS X och HTML, samt de mobila operativsystemen Android och iOS [24]. Beslutet togs att enbart utveckla till Windows, Linux och OS X, då utveckling till mobila plattformar kräver att en del fokus läggs på att hantera de problem som uppstår när användaren endast har en pekskärm som inmatningsmetod.

3.4 TexturePacker

Ett problem som uppstår vid rendering av många sprites är att varje enskild sprite måste skickas till grafikkortet, vilket orsakar en prestandaförlust [25]. För att kringgå problemet användes TexturePacker [25], vilket är ett verktyg som sätter samman enskilda bilder till en stor bild. Utöver det skapas en textfil som innehåller data om de enskilda bildernas placering i den stora bilden. Grafikprocessorn kan då på ett mer effektivt sätt hantera stora mängder enskilda bilder, eftersom huvudprocessorn endast behöver skicka ett fåtal bilder [25].

3.5 Tiled

Tiled [26] är ett verktyg för att skapa grafiska rutnätskartor som kan användas i LibGDX [27]. Tiled ger möjligheten att tilldela attribut till olika rutor som sedan kan användas i spelet. Exempelvis skulle ett attribut kunna beskriva en rutas terrängtyp. Grafiken kommer från tilesets som är kopplade till kartan. Ett tileset är en fil som består av ett rutnät av bilder och används för att ge kartans rutor grafik [28].

3.6 Trello

Trello [29] är ett webbaserat projekthanteringsverktyg som gör det enkelt att hantera projekt med flera personer samtidigt. Projekt delas upp i olika så kallade *boards*, som i sin tur innehåller listor av *cards*. Ett card har ett namn och kan bland annat innehålla en beskrivning och en checklista med uppgifter som ska utföras. Cards kan färgmarkeras och fördelas ut på en eller flera projektmedlemmar, vilket gör det lättare att hålla reda på vem som arbetar med vad. Cards kan enkelt flyttas mellan listor vilket gör det lätt att sortera olika sorters cards i olika listor. Ett board kan exempelvis innehålla två olika listor, där en innehåller cards som arbetas med och den andra innehåller cards som är avslutade [30].

4. Metod

Syftet var att utforska genrekombinationen och därför lades inte mycket vikt på att utvärdera alternativa lösningar på de designproblem som uppkom. Det ansågs viktigare att snabbt komma igång med utvecklingen och därför togs designbesluten i ett tidigt skede. Det behövdes även planeras in tid för återkoppling för att bekräfta huruvida spelet uppfyller de mål som var satta. Därför delades projektet upp i en planeringsfas, en utvecklingsfas och en utvärderingsfas.

4.1 Planering

För att underlätta implementering av mjukvara är planering ett viktigt steg. Utan planering ökar risken att komponenter i ett senare skede inte är kompatibla med varandra. Det kan leda till att stora delar av kodbasen behöver omarbetas, vilket kan ta dyrbar tid. För att på förhand få en uppfattning om projektets struktur och eventuella problemområden, kan det vara praktiskt att modellera systemet innan vidare utveckling påbörjas.

Spelet modellerades därför för att skapa en uppfattning om vilken funktionalitet som skulle implementeras och hur den övergripande programstrukturen skulle se ut. För att utvecklingen av spelet skulle bli så effektiv som möjligt ingick det även bestämmelser i planeringen gällande programmeringskonventioner och versionshantering av kodbasen. Konventionerna var till för att få programkoden enhetlig, vilket gjorde koden mer lättläst och lättförståelig.

4.2 Utveckling

Vid utveckling av mjukvara kan det vara mycket svårt att få all planering på plats utan brister innan själva utvecklingen inleds. Vid användningen av en sekvensiell systemutvecklingsprocess [31], där projektet är uppdelat i faser som planering, utveckling och verifiering, är det inte säkert att allt är uppenbart i planeringsstadiet, vilket kan leda till problem. Det kan därför vara bra att använda sig av en mer flexibel arbetsmetod som tillåter justering och omprioritering av uppgifter under utvecklingsprocessen. Som arbetsmetod valdes därför Scrum [32], vilket är en välkänd och världsutbredd teknik för projektarbete inom systemutveckling.

4.2.1 Scrum

Scrum [32] är en iterativ arbetsmetodik som innefattar uppdelning av hela utvecklingsförloppet i så kallade sprintar. En sprint är ett tidsintervall med ett antal delmål som ska utföras under intervallet [33]. Delmålen organiseras med hjälp av en backlog, en lista med uppgifter sorterade efter prioritet [34]. Längden på en sprint varierar beroende på typen av projekt som genomförs. I planeringen av projektet valdes sprintar med en längd på en vecka, dock förekom viss variation av längden i samband med lov och sjukdom. Sprintlängden valdes av anledningen att den ansågs vara den kortaste period där märkbara framsteg kunde observeras.

När arbete sker enligt Scrum utses vanligtvis en Scrum Master som ska se till att projektgruppen kan arbeta utan hinder, samt säkerställa att övriga medlemmar faktiskt arbetar utifrån principerna i Scrum [35]. Projektgruppen valde dock inte ut någon specifik

Scrum Master, utan valde att låta hela gruppen dela på ansvaret. Samma sak gällde även ansvaret som Product Owner, vilket är den person som är ansvarig för produktens utformning samt för produktens backlog.

4.2.2 MVC

För att strukturera upp koden i projektet valdes designmönstret Model-View-Controller (MVC), vilket är ett designmönster inom systemutveckling där koden delas upp i tre självständiga komponenter. De tre huvuddelarna är Modell, Vy, och Kontrollenhet. En viktig fördel som separationen av komponenter medför är att ändringar i en komponent inte påverkar de andra komponenternas beteende [10].

Den huvudsakliga anledningen till valet av MVC som designmönster är den höga moduläritet som mönstret ger, vilket var ett av kraven på produkten. Tidigare erfarenhet av designmönstret inom projektgruppen var även en bidragande faktor.

- *Modell*, beskriver programmets tillstånd och hur det tar sig från ett tillstånd till ett annat.
- *Vy*, kan ses som en grafisk representation av spelets tillstånd som kan förstås av användaren och har ansvaret att rita ut den information som finns i modellen.
- *Kontrollenhet*, har som uppgift att se till att spelets tillstånd uppdateras. Kontrollenheten hanterar indata från användaren och ansvarar för att anropa rätt funktionalitet i modellen.

4.3 Utvärdering

Eftersom en viktig del av syftet var att spelet skulle vara intuitivt, behövde intuitiviteten verifieras på ett lämpligt sätt, vilket utfördes genom att låta personer med varierande strategi- och datorspelsvana testa spelet. Testningen bestod av tre delar: anteckning av spontana reaktioner vid speltestning, semi-strukturerad intervju efter speltesting samt en avslutande enkät.

Testpersonerna började genom att spela fritt, utan påverkan från testledaren. Efter ett par minuters fritt spelande blev testarna slutligen tillsagda att utföra ett antal uppgifter. Det gjordes för att försäkra att de tog del av allt som fanns tillgängligt i spelet. Under hela speltestningen anteckande testledaren vad spelarna gjorde, reagerade och kommenterade på. Efter att testarna spelade klart blev de intervjuade med frågor rörande projektets syfte, problemformulering och spelet i helhet. Intervjun var semi-strukturerad vilket innebar att den var öppen för nya frågor som inte var planerade att ställas innan intervjun [36]. Slutligen fick testpersonerna fylla i en enkät där ålder, kön och spelerfarenhet noterades, samt hur väl återkopplingen fungerade och vad spelaren upplevde och kände kring spelets begriplighet, funktionalitet och tempo.

Under testsessionerna lades vikt på anteckningarna från speltestningen och den efterföljande intervjun medan enkäterna fungerande som ett komplement. Anledningen till att enkäter värderades mindre är det kan vara svårt att få svar på de frågor som är relevanta för en undersökning. I speltestningens fall berodde det på att projektgruppen inte visste vad spelets brister var då enkäten utvecklades. Intervjuer kan däremot ge mer detaljerad information om styrkor och brister, även om de är svårare att använda sig av för att återskapa och bekräfta resultatet [37]. Då enkäterna främst användes för att ge intervjuerna mer struktur och inte hade någon större vikt i utvärderingen, kommer en

sammanställning av dem inte ge en korrekt representation av spelets kvalité. Därav är sammanställningen av enkäterna inte inkluderade i rapporten.

5. Implementation

Kapitlet presenterar spelets viktigaste delar och hur de implementerades, vilket innefattar Graphical User Interface (GUI), spelmekanik, speldesign och systemdesign.

5.1 Speldesign

För att hantera de problem som uppstod kring genrekombinationen och spelupplevelsen fattades ett flertal designbeslut. Avsnittet behandlar resultatet av de beslut som togs.

5.1.1 Speltempo

För att visualisera tid i spelet används en turklocka, vilket fungerar som en brygga mellan de två genrerna. Spelet är i grunden ett Turn-based strategy (TBS) med skillnaden att spelaren tvingas utföra ordrar under tidspress. Spelaren har en bestämd mängd tid på sig under varje tur, där tanken är att spelaren inte alltid hinner med att göra allt som önskas. Spelhastigheten, längden på en tur, kan justeras i inställningarna till det som passar spelaren eller till det som känns rimligt för en specifik karta.

5.1.2 Kontroll

Allt i spelet går att styra med enbart musen. Utöver det finns det ett antal kortkommandon som är tänkta att underlätta för mer avancerade spelare, exempelvis att spelaren snabbt kan markera någon av sina egna enheter genom sifferknapparna på tangentbordet, där varje siffra motsvarar en enhet.

5.1.3 Återkoppling och intuitivitet

En form av återkoppling som spelaren får är att stridshändelser, exempelvis att en enhet tar skada eller undviker en förmåga, förtydligas visuellt av att små meddelanden flyger upp från enheten, se figur 5.1. Ett meddelande kan exempelvis vara ett nummer som representerar skadan eller hälsotillskottet en enhet nyss åsamkades, eller en text som representerar att en förmåga missade eller blockerades. Meddelanden kan skilja i färg och storlek för att förtydliga skillnaden mellan olika typer av meddelanden.

Figur 5.1: Fiendesnigeln (den vänstra enheten) tar skada av soldaten (den högra enheten)

5.2 GUI

GUI:t är det huvudsakliga gränssnittet mellan spelaren och spelet, uppdelat i två delar, menysystem och spel.

5.2.1 Menysystem

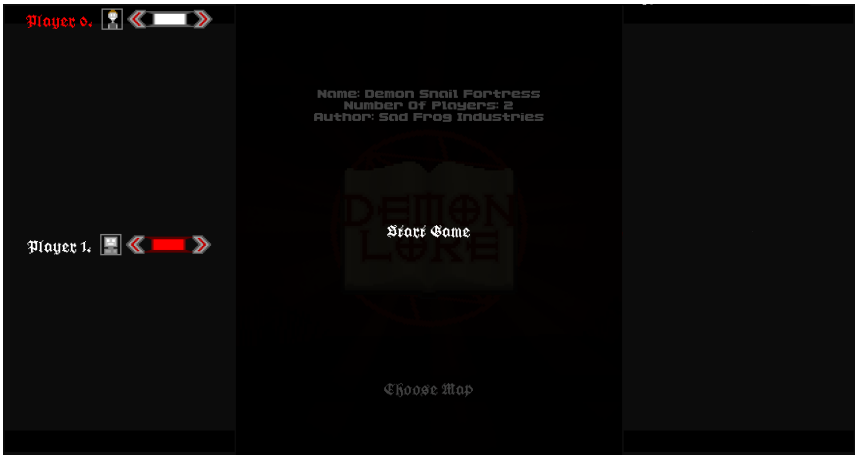
När spelet startas presenteras användaren med en kort laddningsskärm följt av en animerad introduktionssekvens som leder till spelets titelskärm. Därefter presenteras huvudmenyn som består av en horisontell lista med knappar varav en av knapparna är fokuserad och centrerad, se figur 5.2.

Figur 5.2: Huvudmeny

Via huvudmenyn kan användaren navigera till ett flertal undermenyer, exempelvis inställningsmenyn eller hjälpmenyn. Spelaren kan även komma åt en undermeny som visar en lista med namn på spelets utvecklare och andra viktiga omnämnanden. För att enkelt och snabbt starta en ny spelsession kan spelaren välja *New Game*. Spelet laddar då ett förvalt scenario där spelaren får kämpa med att ta sig in i ett fort skyddat av demoniska fiendesniglar. Spelaren kan även välja *Choose map*, vilket visar en undermeny där spelaren kan välja mellan alla tillgängliga kartor, se figur 5.3.

Figur 5.3: Meny för kartval

Efter valet av en karta presenteras ytterligare en undermeny där vissa kartspecifika inställningar kan göras, exempelvis val av lagfärg och spelartyp, se figur 5.4.



Figur 5.4: Meny för kartinställning

I inställningsmenyn kan spelaren ändra spelets inställningar, exempelvis nivån på musik och ljudeffekter, se figur 5.5.

Figur 5.5: Inställningsmenyn

I hjälpmenyn kan användaren läsa om hur spelet fungerar och få mer utförlig information, exempelvis vilka kortkommandon som är tillgängliga på tangentbordet, se figur 5.6.

Figur 5.6: Hjälpmenyn - Kontroller

5.2.2 Spel

Spelets GUI består av ett flertal komponenter utspridda över skärmen. Den centrala delen är spelplanen som tar upp större delen av skärmen. Panelerna i skärmens kanter (figur 5.7, [1-5]) ger information om enheter, spelets tillstånd samt ger möjlighet att kontrollera enheter i mer detalj.

Om spelaren klickar på en enhet, antingen på kartan eller i enhetslistan (figur 5.7, [2]), blir den markerad. När en enhet markeras visas en informationspanel (figur 5.7, [1]) i det nedre vänstra hörnet av skärmen.

Figur 5.7: Översiktsbild av spelets GUI. [1] Informationspanel, [2] Enhetslista, [3] Lista av förmågor, [4] Turklocka, [5] Pausknapp samt beteenden och ordertyper

Informationspanelen visar enhetens nuvarande hälsa och Action Points (AP) (figur 5.8, [3]), attribut (figur 5.8, [4]) och porträttbild (figur 5.8, [2]). Informationspanelen innehåller även en lista med bilder som representerar enhetens olika stridsstilar (figur 5.8, [5]) och genom att klicka på bilderna kan spelaren byta enhetens stridsstil. Vid sidan av informationspanelen finns små bilder som visar enhetens aktiva statuseffekter (figur 5.8, [1]).

Figur 5.8: Panel för enhetsinformation. [1] Statuseffekter, [2] Enhetsporträtt, [3] Nuvarande hälsa och AP, [4] Attribut, [5] Stridsstilar

I den centrala övre delen av skärmen visas de enheter som spelaren kontrollerar (figur 5.7, [2] figur & 5.9). Genom att dubbelklicka på en enhet kommer spelet att markera och centrera enheten.

Figur 5.9: Panel för enhetsval

När en enhet är markerad visas de förmågor som enheten har tillgång till i den centrala nedre delen av skärmen (figur 5.7, [3] & figur 5.10). Om en förmåga är på cooldown får dess ikon en grafik som visar samt räknar ner hur många turer det är kvar innan förmågan kan användas igen.

Figur 5.10: Panel för enhetsförmågor

I det övre högra hörnet av skärmen finns spelets turklocka (figur 5.7, [4] & figur 5.11). Klockan fylls upp av lila färg över tid och när hela klockan är fylld övergår spelet till en ny tur. I skärmens nedre högra hörn kan spelaren välja mellan enhetens olika beteenden (figur 5.12, [2]) och ordertyper (figur 5.12, [3]), samt nå spelets pausknapp (figur 5.12, [1]).

Figur 5.11: Turklocka

De flesta GUI-element har försetts med tillhörande tooltips. Ett tooltip är en textruta som innehåller mer ingående information om det element som muspekaren befinner sig över, se figur 5.13.

När en ny tur påbörjas lyser skärmens kanter upp, vilket är tänkt att göra det tydligare för användaren att en ny tur påbörjats.

Figur 5.12: [1] Pausknapp, [2] Ordertyppanel, [3] Beteendepanel

Figur 5.13: Panel för enhetsinformation - Tooltip för en stridsstil

5.3 Spelmekanik

För att förstå hur spelet fungerar behövs kännedom om spelmekaniken, implementationen av spelets regelverk. [38] Regelverket, själva kärnan i spelet, definierar de regler som uttrycker vad spelare kan göra i spelet. Avsnittet redovisar de centrala delarna av spelmekaniken.

5.3.1 Ordersystem

När spelaren beordrar en enhet att flytta på sig eller använda en förmåga, skapas en *order* som delas upp i mindre handlingar. Hur många handlingar enheten kan utföra under en tur beror på handlingarnas AP-kostnad. Om enheten inte har tillräckligt med AP för att slutföra sin nuvarande order under en tur kommer den att utföra ordern så långt dess AP räcker. Exempelvis kan den använda all tillgänglig AP för att flytta sig halva sträckan för att nästa tur flytta sig den återstående delen. Om spelaren väljer att ändra en enhets order kommer ändringen att ske först nästa tur.

De handlingar som utförs under varje tur animeras parallellt, till den grad det är möjligt. Det innebär att oberoende handlingar kan visas samtidigt, trots att alla handlingar spelmekaniskt utförs sekvensiellt. Två oberoende handlingar kan exempelvis vara förflyttningar vars vägar inte korsar varandra.

5.3.2 Uppdrag

Varje spelares enskilda mål på en karta definieras av uppdrag, vilket består av ett eller flera delmål som måste avklaras för att vinna spelet. Det finns två sorters uppdrag: huvuduppdrag och sidouppdrag. Huvuduppdrag måste slutföras för att vinna spelet och misslyckas spelaren med någon del utav uppdraget förlorar spelaren. Sidouppdrag är extra uppdrag som kan utföras om spelaren vill, dock utan någon form av belöning. Ett mål kan exempelvis vara att *besegra alla motståndare*, *överlev i x antal turer* eller *nå en viss position på kartan*. Det är även möjligt att kedja uppdrag, alltså ordna dem sekvensiellt, vilket innebär att spelaren måste klara av tidigare uppdrag i kedjan för att ta sig an nästa.

5.3.3 Stridssystem

Alla enheter turas om med att utföra sina handlingar, där ordningen beror på varje enhets hastighetsattribut. Varje förmåga som utförs har en räckvidd som avgör hur många rutor bort från enheten som målet för förmågan kan befinna sig.

Alla effekter en förmåga kan applicera, exempelvis skada, sker via statuseffekter. För att en förmåga ska göra direkt skada när den träffar, måste förmågan applicera en statuseffekt av typen *Damage*. Utöver Damage finns ett antal statuseffekter som kan appliceras:

- *Poison*, gör procentuell skada över tid, där skadan är baserad på enhetens maxhälsa.
- *Damage over time*, gör fast skada över tid.
- *Doom*, dödar enheten när statuseffektens tid har runnit ut.
- *Stun*, förhindrar enheten att agera under en viss tid.
- *Stealth*, gör enheten osynlig för fiender.
- *Heal*, återställer enhetens hälsa med en given mängd.
- *Slow*, minskar antalet rutor enheten kan röra sig.
- *Dispel*, tar bort negativa statuseffekter från enheten.

En statuseffekt reagerar på och utför effekter i enlighet med ett trigger-system. De triggers som finns samt tidpunkten de refererar till är:

- *Activation*, när statuseffekten appliceras.

- *Deactivation*, när statuseffekten försvinner eller tas bort.
- *Turn Begin*, när en ny tur påbörjas.
- *Turn End*, när en tur avslutas.
- *Move*, när enheten förflyttar sig.
- *Ability*, när enheten använder en förmåga.
- *Damage*, när enheten tar skada.
- *Heal*, när enheten återfår hälsa.

Det finns en risk att appliceringen av en statuseffekt misslyckas, vilket beror på flera faktorer: huvudsakligen anfallarens träffchans samt målets möjlighet att undvika attacken. Hur den faktiska beräkningen utförs beror bland annat på vilken slags statuseffekt det handlar om, samt huruvida målet är en enhet eller en ruta på kartan. När en statuseffekt som gör skada appliceras, varierar skadan som utdelas beroende på målets skyddsattribut. Vilken typ av skyddsattribut som används för beräkningen av skadan beror på statuseffektens typ.

5.3.4 Karta

En karta är en spelplan av godtycklig storlek uppbyggd av ett rutnät. Varje ruta består av två lager, där det undre lagret alltid innehåller någon sorts terräng, som gräs, och det övre lagret exempelvis enheter eller andra blockerande objekt.

Till varje karta finns en resursfil som specificerar kartans spelegenskaper, som hur många spelare kartan har och vilka enheter varje spelare börjar med samt deras position på kartan. Resursfilen specificerar även ett eller flera uppdrag för varje spelare.

5.3.5 Enhet

En enhet har bland annat namn, en position på kartan samt attribut. Attributen beskriver exempelvis enhetens maximala hälsa och AP. En enhet har även listor av stridsstilar, förmågor och aktuella statuseffekter som påverkar enheten.

Visuellt representeras en enhet på kartan och även i användargränssnittet. På kartan animeras enheten beroende på dess nuvarande status, som att enheten springer eller attackerar.

5.3.6 AI

Varje datormotståndare har en AI som styr dess enheter. AI:n ger enheten order om till exempel förflyttning eller vilket mål den ska attackera. Varje enhet förses med en egen AI beroende på vilken klass enheten har och vad den har för förmågor. De enhets-AI som finns är *Docile*, *Aggressive*, *Hunter* och *Healer*.

Docile-AI:n är väldigt enkel. Enheten går planlöst runt på kartan utan att attackera fiender, men går in i ett panik-läge och försöker fly om den blir attackerad.

Aggressive-AI:n är lite mer avancerad. Enheten går runt planlöst, men om den får syn på en fiende kommer enheten att försöka attackera fienden tills dess att fienden antingen

är död eller utom synhåll.

Hunter-AI:n är den mest avancerade. Enheten går runt på kartan för att söka upp fiender. Om en eller flera fiender kommer inom synhåll kommer enheten jaga fienden. En viktig skillnad jämfört med Aggressive-AI:n är att Hunter-AI:n prioriterar att attackera fiender som är svaga eller har förmågor som kan hjälpa sina allierade.

Healer-AI:n är den sista tillgängliga AI:n. Enheten försöker hålla sig nära sina lagkamrater och kommer använda sig av sina hjälpsamma förmågor på de lagkamrater som ligger illa till. Exempelvis kommer Healer-AI:n försöka återställa hälsan på den mest skadade lagkamraten.

5.3.7 Beteendesystem

För att underlätta för spelaren implementerades ett enkelt beteendesystem. Spelarens enheter har tillgång till en mindre mängd beteenden, där spelaren väljer vilket beteende en enhet ska använda sig av. Ett beteende är en mindre avancerad AI och finns i tre olika varianter:

- *Hold Position*, innebär att enheten står still. Enheten använder dock automatiskt tillgängliga aggressiva förmågor på fiender inom räckhåll.
- *Auto Support*, innebär att enheten automatiskt använder vänliga förmågor på sina allierade enheter. En vänlig förmåga kan exempelvis vara en förmåga som återställer en allierad enhets hälsa.
- *Auto Attack*, innebär att enheten automatiskt använder sin valda förmåga på fiendeenheter som kommer i närheten. Enheten kommer även följa efter fiender i närheten så gott det går.

5.4 Systemdesign

Avsnittet redovisar hur systemet designades, både på en övergripande nivå och i detalj, samt hur vissa av de designbeslut som togs implementerades.

5.4.1 Orderexekvering

Ordersystemet finns på en sådan abstraktionsnivå att modellen varken vet eller behöver veta huruvida en order kom från en mänsklig spelare eller en datorspelare. Abstraktionen medför att det är väldigt enkelt att exempelvis låta två datormotståndare slåss mot varandra, utan att en mänsklig spelare deltar. En order innehåller därför endast information om vad en enhet avser utföra under framtida turer, oberoende av enhetstyp och spelare.

Nästa steg i ordersystemet, exekveringen, implementerades isolerat från övriga delar i modellen, för att bibehålla en modulär och effektiv struktur. Delen kallas för orderexekvering och hanterar utförandet av ordrar för alla enheter. Genom att begränsa möjligheten att direkt ändra på tillståndet kan avsedd spelmekanik garanteras då tillståndsändringar endast sker på ett ställe. Alternativet hade varit att exempelvis tillåta att AI:n direkt påverkar exempelvis enheters positioner eller attribut och därmed kringgå exekveringssystemet.

5.4.2 MVC

I enlighet med MVC delades huvuddelen av koden upp i tre huvudpaket: *Kontrollenhet*, *Vy* och *Modell*.

- *Kontrollenhet*, innehåller all kod som hanterar programflöde samt indata från spelare, exempelvis byte mellan spelskärmar. Utöver det hanterar paketet även uppspelning och övrig hantering av ljud och musik.
- *Vy*, innehåller all kod som hanterar rendering av spelet, vilket innefattar en grafisk representation av modellen som exempelvis kartan, enheter och andra grafiska effekter.
- *Modell*, innehåller all kod som hanterar spelets tillstånd och de komponenter som spelinnehållet är uppbyggt av, exempelvis enheter, förmågor och kartor. Utöver det innefattar paketet även orderexekvering, AI och uppdrag.

Utöver de tre huvudpaketen skapades två tilläggspaket med syfte att samla de klasser som används utav flera av huvudpaketen.

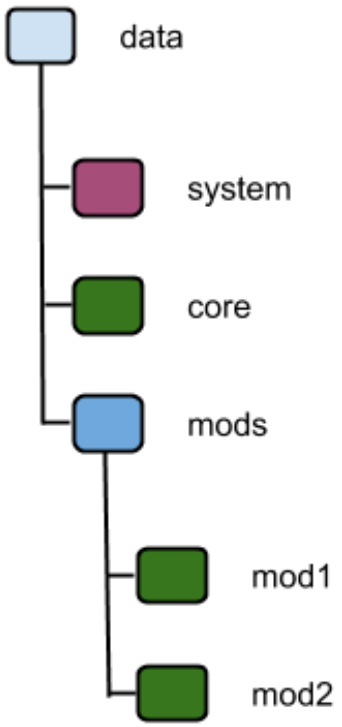
- *Factory*, ansvarar för det mesta som är kopplat till inläsning av spelets resursfiler. Här finns funktionalitet för skapande av de Java-objekt som representerar den inlästa datan.
- *Utils*, innehåller klasser som stöder inläsningen av resursfiler samt den logg-funktionalitet som används för att logga viss utvecklarinformation kopplat till olika områden i programmet.

5.4.3 Dynamiskt innehåll

I spelets används en dynamisk program-arkitektur för hantering av resurser, det vill säga grafik, ljud, kartor och enheter. Då få delar av spelinnehållet är skrivna direkt i programmets källkod, utan istället representeras av externa resursfiler, kan slutanvändare som inte är bekanta med programutveckling tillföra nytt spelinnehåll.

Spelets resursfiler, som definierar spelets innehåll, kan läsas och förstås av människor. Det ger spelare möjlighet att själva ändra och lägga till innehåll, se figur 5.15.

Spelets resurser är uppdelade i mapparna *system*, *core* och *mods*, se figur 5.14. System innehåller alla huvudkomponenter till spelet som typsnitt och grafik för menyer. Core, från engelska för kärna, innefattar de källfiler till kartor och enheter som medföljer spelet som standard.
Mods, likt core, innehåller källfiler till spelinnehåll. Syftet med mods är dock istället att slutanvändaren kan skapa eget innehåll, exempelvis genom att kopiera och ändra på resurser från core. På så sätt får användare möjlighet att enkelt skapa nya kartor och enheter eller ändra på redan existerande innehåll utan att behöva ändra på resurserna i core, vilket eventuellt kan göra spelet ospelbart.



Figur 5.14: Filstruktur för spelets resurser

Resurser i core och mods läses in när spelaren väljer karta. När en karta har valts söker systemet efter de refererade resurserna i core och eventuellt mods om kartan kommer därifrån, där innehåll från mods prioriteras över core. Om en sökt resurs inte skulle hittas eller om ett fel upptäcks genereras ett felmeddelande med information om viken fil problemet avser och eventuellt på vilken rad i filen problemet uppstod samt övrig specifikation.

```
<entities>
    <!--
        A unit with attributes:
        teamid = "the id corresponding to the controlling player" | REQUIRED |
        posx/posy = "the units position on the map" | REQUIRED |
        asset = "the name of graphical asset / entity file used" | REQUIRED |
        class = "the name of the class used" | REQUIRED |
        [primary,secondary,tertiary] = "the name of the mod used" | OPTIONAL |
            >> Defaults to using no mod of that type.
    -->
    <unit teamid="0" posx="20" posy="7" individualname="Alpa Johan" asset="fighter" class="fighter"/>
    <unit teamid="0" posx="20" posy="6" asset="druid" class="druid"/>
    <unit teamid="0" posx="21" posy="7" asset="wizard" class="wizard"/>
    <unit teamid="0" posx="22" posy="8" asset="ranger" class="ranger"/>

    <unit teamid="1" posx="2" posy="28" individualname="Olof 3000" asset="alphademonsnail" class="alphademonsnail"/>
    <unit teamid="1" posx="3" posy="29" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="3" posy="29" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="4" posy="27" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="6" posy="13" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="7" posy="17" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="11" posy="27" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="13" posy="26" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="20" posy="27" asset="demonsnail" class="demonsnail"/>
    <unit teamid="1" posx="22" posy="26" asset="demonsnail" class="demonsnail"/>
</entities>
```

Figur 5.15: Exempel på resursfil från spelet

6. Resultat och diskussion

I kapitlet diskuteras i vilken utsträckning implementationen uppfyller syftet med projektet. De arbetsmetoder som använts utvärderas med avseende på om de varit hjälpsamma för projektet eller huruvida andra metoder hade varit att föredra. Avslutningsvis diskuteras spelets möjligheter till vidareutveckling med utgångspunkt i användartesterna.

6.1 Blandning av genrer

Att få den valda kombinationen av genrer att fungera på ett sätt som känns naturligt för användaren har varit en av de största utmaningarna under utvecklingen. Spelet följer till stor del de mål som specificeras i syftet då kombinationen av genrer fungerar som koncept. Det visade sig dock att tursystemet inte upplevdes som intuitivt, och var svårt att förstå utan en förklaring, vilket hjälpmenyn var tänkt att ge.

Spelet är klart spelbart och utmanande, dock inte till en sådan nivå att det kan engagera spelare att fortsätta spela i någon längre tidsperiod då spelinnehållet är minimalt. För att bättre behålla spelarens intresse behövs mer innehåll, exempelvis en handling. Mer tid hade behövts läggas på utvecklingen av spelet för att minimera antalet irritationsmoment samt fullborda spelmekanikerna.

Som tidigare nämnts i 5.1.1 används en turklocka för att agera brygga mellan de turbaserade momenten och realtidsmomenten. Turbaserade strategispel delar ofta upp sina turer i två faser, en förflyttningsfas och en stridsfas. Uppdelningen övervägdes, dock är det diskutabelt om spelet hade varit bättre om delningen hade använts. Flera testpersoner upplevde dock frustration då enheter inte flyttade sig omedelbart efter att en order tilldelats, vilket kan motivera en sådan uppdelning.

Ett beslut som togs var att enheternas handlingar skulle renderas i realtid, vilket innebär att alla enheter flyttar sig och attackerar samtidigt i den mån det är möjligt. Det gör att enheterna grafiskt agerar samtidigt även om de spelmekaniskt följer turordningen. Beslutet togs för att få spelet att kännas mer som ett realtidsspel, även om det i grunden beter sig som ett turbaserat spel. Det här löser en del av den problematik som uppstår i samband med sammanslagningen av de två genrerna, men vissa modifikationer skulle krävas för att bättre hantera de tempoväxlingar som uppstår.

6.2 Effekt av designbeslut

De olika designbeslut som togs under projektets planeringsfas har både påverkat projektets utförande och resultat. Avsnittet redovisar mer i detalj hur väl de beslut som togs överensstämmer med syftet samt hur projektgruppen resonerade kring de beslut som togs.

6.2.1 Realtid mot turbaserat

Att kombinera de två genrerna var svårare än förväntat. De enda aspekter i spelet som egentligen kan klassificeras som realtidsstrategi är att varje tur är tidsbegränsad och att alla spelare ger ordrar samtidigt. Skulle de här momenten förminskas eller rent av raderas skulle spelet fungera som ett vanligt turbaserat strategispel. På grund av den naturliga inkompatibiliteten mellan de två genrerna var det här något som kan anses som oundvikligt.

Som nämns i 5.3.3 utför spelets enheter sina ordrar i ordning. Anledningen till att stridssystemet fungerar som det gör är för att minska de potentiella konflikter som skulle kunna uppstå ifall alla enheter kunde röra på sig samtidigt. En nackdel med hur stridssystemet fungerar är att spelet kan upplevas som stelare än om en mer realtidsbaserad variant togs fram. Beslutet att använda det här stridssystemet togs dock tidigt, då projektgruppen ansåg att det var en intressant kombination värd att utforska.

6.2.2 Strid

Som nämns i 5.3.3 gör varje enhet varierande skada samt har en möjlighet att missa eller göra extra skada med en attack, vilket tillför ett slumpmoment. Fördelen med slumpmomenten är att spelet blir mindre förutsägbart vilket kan öka spelets livslängd, medan en nackdel är att det blir svårare att uppnå perfekt balans. Spelbalansering är dock inget större problem eftersom spelet inte har något slags flerspelarläge, vilket innebär att det är viktigare att spelet är underhållande än att det är perfekt balanserat.

För att ge enheterna mer variation och strategisk betydelse har enheterna olika stridsstilar, som nämns i 5.3.5, där varje stridstil ger enheten tillgång till olika förmågor. Att välja rätt stridsstil eller förmåga blir då en strategisk utmaning för spelaren. Syftet med stridsstilar var att utöka det en enhet kan tillföra till spelet, då spelaren kontrollerar förhållandevis få enheter.

För att hindra att vissa förmågor används för ofta, kan de ha en så kallad *cooldown*, vilket nämnts i 5.2.2. När en sådan förmåga används blir den obrukbar i några turer, vilket var ett enkelt sätt att ge förmågor en mer strategisk roll i spelet, då en förmåga kan göras väldigt mycket mer kraftfull än andra men då ha en längre cooldown.

Som nämnts i 5.3.1 kostar enheternas förmågor och förflyttningar AP att utföra, vilket begränsar hur mycket en enhet kan göra under en tur. Eftersom enheter får tillbaka hälften av sin AP varje tur, kan det innebära att enheter som vilat under föregående tur har mer AP att spendera än de som agerat. Det här måste spelaren ha i åtanke för att utnyttja sina enheter maximalt, vilket ökar spelets strategiska djup.

En nackdel med hur AP-systemet fungerar är att enheterna kan utföra en varierande mängd handlingar varje tur, vilket kan upplevas som enerverande om spelaren inte är fullt medveten om mekaniken. För att uppnå bättre balans mellan realtid och turbaserat borde AP-systemet ses över och utvecklas vidare.

6.2.3 AI

Syftet med AI:n var att göra spelet utmanande genom att göra motståndet intelligent. Om fiender samarbetar med varandra och utnyttjar sina förmågor väl, ställer det större krav på spelaren vilket kan göra spelet mer underhållande i längden. Spelets AI blev tyvärr inte så avancerad som planerat, utan har endast grundläggande funktionalitet. Den implementerade AI:n blev ändå utmanande för spelaren, då den enkelt kan kontrollera många enheter samtidigt. Dock verkar det mer som att varje enhet slåss för sig själv och därför blir svårighetsgraden direkt beroende på den individuella styrkan hos fiendeenheterna, och inte laget som helhet.

Som nämns i 5.3.7 använder spelarens enheter olika beteenden. Ett beteende är en simplare AI som styr enheten om den inte har någon order att utföra. Fördelen med beteenden är att om mycket händer samtidigt kan spelaren fokusera på en specifik enhet och samtidigt låta de andra enheterna kontrolleras av sitt beteende. Är beteendesystemet för intelligent kan det dock vara till för stor hjälp för spelaren, vilket kan upplevas som att spelet spelar sig självt [11].

Mot slutet av projektet upptäcktes en del svårigheter med pathfinding, då viss funktionalitet i spelet som rörde förflyttning var svår att implementera. Därav undersöktes möjligheterna att modifiera den algoritm som användes. Det visade sig tyvärr vara svårare än förväntat och fokus lades istället på att optimera den kod som redan användes.

6.3 Modularitet

Som nämns i 1.2 var det ett mål att spelet skulle vara modulärt. För att en spelare enkelt ska kunna lägga till nytt innehåll i ett spel är det hjälpsamt om utförlig dokumentation om tillvägagångssätten finns att tillgå. Går något fel i processen är tydliga felmeddelanden till stor nytta. Spelets resursfiler är dokumenterade till den grad att användare med programmeringsvana bör kunna lägga till eget innehåll utan större problem.

När felaktigt formaterade resursfiler läses in, genereras felmeddelanden om de felaktiga filerna samt vad som måste åtgärdas. Dock kräver informationen från felmeddelanden viss kunskap om resursstrukturen för att tolkas.

Ett redigeringsverktyg ansågs nödvändigt för att göra det enkelt nog att lägga till nytt innehåll. Då tid inte fanns för utveckling av ett sådant verktyg, har inte mycket fokus lagt på att göra modifiering av resurser användarvänligt.

6.4 Metod och utvecklingsverktyg

I avsnittet diskuteras och utvärderas arbetsmetoderna som har använts, med avseende på hur väl de följts, vilken påverkan de haft på arbetet och huruvida andra metoder skulle varit mer effektiva.

6.4.1 Scrum

Det var svårt att få rätt mängd innehåll i varje sprint, då enbart ett fåtal gruppmedlemmar hade tidigare erfarenhet av metoden eller spelutveckling i någon större utsträckning. Därför var det svårt att uppskatta uppgifternas tidsåtgång, vilket ledde till att enbart några få arbetsuppgifter lades till i sprintens backlog under den inledande fasen. Uppgifterna tog bara någon enstaka dag att färdigställa vilket ledde till att fler uppgifter behövde läggas till mitt i sprinten. Det här bröt mot den ursprungliga idén om att all planering och synkronisering av arbete skulle ske under Scrum-möten i slutet av veckorna. Det här ställde dock inte till några större problem, men fick gruppen att förstå att mer tid behövde läggas på planeringen av sprintar.

Det blev med tiden lättare att uppskatta en lämplig mängd uppgifter för varje sprint, vilket ledde till att de allra flesta uppgifterna hann göras samt att nya uppgifter sällan behövdes läggas till mitt i sprinten. Mot slutet av projektet blev dock sprintarna mindre väldefinierade på grund av tidsbrist, vilket ledde till att uppgifter lades till vid behov och inte vid planerade möten. Som helhet fungerade dock strukturen förhållandevis bra och var till stor hjälp för att se till att arbetet kunde ske parallellt på olika uppgifter, utan större konflikter.

Vanligtvis när man arbetar med Scrum hålls möten i början av varje arbetsdag för att varje gruppmedlem ska kunna hålla sig uppdaterad om hur det det går för resten av gruppen, samt ha möjlighet att fråga om hjälp eller liknande ifall det skulle behövas [39]. Dagliga möten var dock inget som gick att åstakomma för det här projektet, då det inte kunde garanteras att hela gruppen var samlad varje dag på grund av arbeten på annat håll. Bristen på sammankomster ledde stundtals till kommunikationsproblem i gruppen,

där flera personer hade börjat jobba på samma eller liknande uppgifter och fick reda på det först i slutet av veckan. Trello, som nämns i 3.6, borde i teorin ha eliminerat risken för sådana krockar, men då uppgifterna, särskilt till en början, kunde vara löst definierade uppstod problemen ändå. Det blev därmed viktigt att sätta upp tydliga uppgifter och se till att alla följde aktiviteten på Trello.

Trots det var Trello var ett mycket bra hjälpmedel för att strukturera upp sprintplanering och backlogen, och användes därför flitigt genom hela arbetsprocessen. Tjänsten är väldigt smidig och enkel att använda, vilket gjorde gruppmedlemmarna mer positiva till den. Som nämns i 4.2.1 ska vanligtvis en Scrum Master utses för att garantera att samtliga medlemmar följer Scrum och att allt går rätt till. Avsaknaden av en Scrum Master hade dock troligtvis ingen större inverkan på arbetet, då gemensam beslutsfattning ändå fungerade på ett godtagbart sätt utan större konflikter.

6.4.2 Git

Ett versionshanteringsprogram är en absolut nödvändighet om storskaliga utvecklingsprojekt med flera inblandade överhuvudtaget ska fungera. Git fyllde den rollen väl och programmets kraftfulla merge-verktyg var mycket användbara. Förutom några mindre problem och konflikter kunde Git användas utan svårigheter. Git var lätt att använda så länge varje gruppmedlem arbetade med skilda uppgifter, vilket inte var något problem så länge medlemmarna höll sig uppdaterade via Trello. När konflikter väl skedde, gick det oftast snabbt att lösa då Git tydligt visar och gör det enkelt att hantera förändringar mellan två skilda versioner av en fil.

Som nämns i 3.1 användes Github som en fjärrserver för att centralt lagra källkoden, så att varje gruppmedlem enkelt kunde nå den. Eftersom Trello användes för i stort sett all Scrum-verksamhet, behövde inte Githubs extra funktionalitet, exempelvis buggh anteraren *Issues* [40], användas överhuvudtaget. Github är även ett bra verktyg för att följa projektets arbetsgång, då det finns möjlighet att kolla grafer och annan statistik [41].

Genom större delen av projektet skedde en sammanställning av arbetet, en så kallad merge, i slutet av varje vecka i samband med ett Scrum-möte, där veckans arbete sammanställdes på en gemensam utvecklingsbranch. Tanken var att mängden branches skulle hållas på en hanterbar nivå och att utvecklingsbranchen inte skulle förgrena sig för mycket. Arbetsmetoden gav en bra struktur i arbetet och gjorde det lättare för gruppmedlemmarna att hålla sig uppdaterade med den senaste versionen, vilket exempelvis innebar att ingen arbetade med komponenter som inte längre existerade. Trots att gruppmedlemmarna använde olika utvecklingsmiljöer gick det utan problem att sammanföra arbetet som gjorts under veckan. Metoden följdes dock inte alltid till punkt och pricka vilket ibland orsakade konflikter eller bortkastat arbete, då viss kod som skrivits inte gick att använda.

6.4.3 LibGDX

Som nämns i 3.3 användes biblioteket LibGDX, vilket innebar både för- och nackdelar. LibGDX är dock ett relativt välanvänt och väldokumenterat bibliotek, vilket gjorde det enkelt att få svar på eventuella frågor som uppstod.

En fördel med att använda LibGDX är att nästan alla grundläggande delar som behövs för spelutveckling redan finns i biblioteket. Det innebär att utvecklare inte behöver lägga onödig tid på att skapa grundläggande komponenter, utan kan börja direkt med att implementera de delar som är unika för spelet.

LibGDX innehåller ett paket vid namn *Scene2D* [42], med huvuduppgift att göra det enklare att utveckla användargränssnitt. I projektet togs beslutet att använda Scene2D för

utveckling av användargränssnitt, vilket visade sig vara både praktiskt och problemfyllt. Praktiskt var att väldigt många användbara saker såsom tabeller, förloppsindikatorer och textrutor redan fanns färdiga att använda [43]. Dock upplevdes Scene2D som problemfyllt då det inte alltid var uppenbart hur man skulle få till den formgivning som eftersträvades.

Avslutningsvis kan det sägas att Scene2D mest upplevdes som förvirrande och frustrerande, även om det blev lättare att hantera i slutfasen då de som utvecklade gränssnitten mer eller mindre lärt sig hur det fungerade. Trots problemen som uppstod var det nog enklare och mer effektivt att använda Scene2D än att utveckla ett eget GUI-bibliotek, så valet att använda Scene2D var i slutändan rätt beslut.

Hanteringen av ljudeffekter och musik i LibGDX är inte helt optimal. Eftersom LibGDX är utvecklat för att fungera på flera olika plattformar med samma källkod uppstår ett problem vid hantering av ljud, då uppspelning och övrig hantering av ljud skiljer sig rätt markant mellan olika plattformar. Av den anledningen är bibliotekets stöd för ljud inte mer än dugligt, med ett bristande stöd för att enkelt kunna hantera start och stopp av flera ljudeffekter samtidigt.

LibGDX tillhandahöll även ett bibliotek för AI och pathfinding [44]. Biblioteket möjligjorde en snabb start när det kom till att flytta enheter även om vissa problem uppstod framåt slutet av projektet. Utifrån det här kan det diskuteras om inte en egen pathfinding-algoritm hade fungerat bättre för oss även om det tagit längre tid. Dock är den algoritm som används i biblioteket mycket effektiv och det är tveksamt om den skulle kunna förbättras nämnvärt.

Trots alla problem som stöttes på var det nog ett bra beslut att använda LibGDX, då arbetet med implementationen av spelidén kunde komma igång väldigt snabbt.

6.4.4 MVC

Som nämns i 4.2.2 användes MVC som designmönster för utvecklingen och var tänkt att skapa struktur på programmet, vilket visade sig vara svårare än förväntat. Problematiken uppstod huvudsakligen av två anledningar: att biblioteket som används är LibGDX och att produkten är ett spel.

En svårighet som LibGDX medför är att biblioteket har en starkt sammankopplad Vy-Kontrollenhet, särskilt *Actor*-objektet [45], vilket har lett till att de två komponenterna i programmet var svåra att separera. Om mer tid lagts på att separera komponenterna skulle en lösning säkert hittats, men då problemet upptäcktes i ett sent skede fanns det inte tillräckligt med tid för att åtgärda det.

Det är väldigt svårt att planera systemstrukturen för ett spel då spelutveckling oftast är en i synnerhet iterativ process. Även om ett välskrivet och välgjort designdokument följs brukar oväntade problem uppstå som tvingar utvecklarna att till viss del modifiera den planerade designen. Spel ställer även oftast större krav på prestanda, då det brukar anses viktigt att spel är genomgående responsiva och stabila [46]. Kravet på prestanda kan stå i direkt konflikt med rigida strukturer som MVC, vilket innebär att det blir svårt att följa ett sådant designmönster till punkt och pricka.

Prestandakravet är troligen anledningen till att LibGDX har den starka kopplingen mellan Vy och Kontrollenhet, särskilt då LibGDX är tänkt att fungera bra även på mobila plattformar [24], som oftast har sämre prestanda än persondatorer.

Trots svårigheterna hade troligtvis programstrukturen varit väsentligt sämre och mer kaotisk om inte MVC använts, vilket är en stark anledning till att designmönstret ändå ansågs vara ett bra val i slutändan.

6.5 Användartester

Som nämns i 4.3 utfördes speltester med ett antal testpersoner. Genom en analys av resultatet från speltesterna kunde både positiva och negativa delar av spelet identifieras.

Reaktionerna från intervjuerna var till stor del positiva. De flesta fann att spelet var roligt och var intresserade av att se en vidareutveckling av konceptet. Innan testpersonerna insåg att spelet till viss del var turbaserat, fann de spelet långsamt då det dröjde innan spelet reagerade på deras handlingar. Den återkoppling som gavs när en enhet fick en order om att förflytta sig, en pil från enheten till slutdestinationen, gjorde att testpersonerna förstod att något faktiskt hände även om det inte skedde på en gång. Det dröjde dock inte länge innan de upptäckte turklockan vilket ledde till att de genast upplevde spelet som mer förståeligt och roligt.

Att kunna pausa spelet var väldigt uppskattat bland de spelare som ville ha mer kontroll, medan de som var vana vid ett högre speltempo valde att helt undvika pausfunktionen. En del frustration uppkom även då spelarna behövde vänta på turklockan för att enheter skulle flytta sig. Turklockan fungerade bäst på snabb hastighet, då spelarna inte hann bli frustrerade av att enheterna inte rörde på sig innan en ny tur. Förslag på att förbättra responsen vore att tillåta enheter flytta på sig innan turens slut, samt ge bättre grafisk återkoppling när en enhet är tvungen att vänta på att få använda en förmåga igen.

Många anmärkte att typsnittet som användes i menyer var svårt att läsa, vilket kan bero på att det typsnitt som användes innehöll några otydliga tecken. Testarna tycktes uppskatta spelets grafiska stil samt de ljudeffekter som användes och ingen hade svårigheter med att skilja enheterna åt eller förstå vad varje ljudeffekt betydde.

Positivt var att testare med god spelvana snabbt kunde sätta sig in i spelet och förstod vad som hände. Enheters förmågor och pausfunktionen medförde att testpersonerna spelade spelet på olika sätt, varav en del använde pausfunktionen och förmågor regelbundet, medan andra undvek funktionerna och spelade mer likt realtidsstrategi.

Testpersonerna utvecklade egna spelstilar, med varierande aggressivitet och utnyttjande av enheternas synergier. Att de inte spelade på exakt samma sätt bör ses som något positivt, då det indikerar att det inte finns ett rätt sätt att spela på. Variationen av spelmöjligheter kan ge spelet en större målgrupp då spelstilen kan anpassas efter användaren.

Även om utmaningen i spelet var begränsad hade testpersonerna roligt, vilket tyder på att spelkonceptet har potential och kan vara värt att utveckla vidare. Spelets återkoppling var effektiv då testarna fick svar på vad de gjorde med hjälp av bilder och ljudeffekter, vilket var viktigt då en stor del av syftet fokuserade på att spelet skulle vara intuitivt.

En negativ aspekt av spelet var att det inte fanns möjlighet att markera flera enheter samtidigt, vilket är vanligt inom RTS [3]. Det ledde till frustration bland vissa testpersoner som därför ofta valde att enbart kontrollera en eller två enheter på kartan, för att sedan välja nya när en enhet gick förlorad. Vid eventuell vidareutveckling av spelet bör därför möjligheten att kunna kontrollera flera enheter samtidigt övervägas.

Spelets GUI upplevdes delvis som bristfälligt, då spelarens möjligheter inte alltid var tydliggjorda. Egenskaper som stridsstilar och beteenden användes inte alls, vilket delvis berodde på att de inte behövdes för att vinna, men mestadels för att spelarna inte kände till att möjligheterna existerade. Knapparna för stridsstilar är små och ligger i botten av enhetens informationspanel vilket gjorde de svåra att upptäcka. Samma problem gällde för beteenden som var tillgängliga i det nedre högra hörnet av skärmen. För att spelarna ska kunna ta del av allt innehåll behöver GUI:t omarbetas så alla komponenter blir tydliga för användarna.

Spelets pathfinding gav upphov till frustration hos testpersonerna, då blockerande enheter medförde att andra enheter valde icke önskvärda vägar. Om en passage var tillfälligt blockerad, kunde enheter välja vägar åt rakt motsatt håll än planerat, vilket kunde få katastrofala följder för uppdraget. Spelets pathfinding behöver ses över och möjligen göras om från grunden för att fungera önskvärt.

6.6 Vidareutveckling

Det finns många delar av spelet som kan vidareutvecklas, exempelvis grafiken som skulle kunna vara mer effektfull och detaljerad. En handling och rollspelsinslag hade till exempel gett varje spelsession mer sammanhang samt ökat spelarens inlevelse. Som nämns i 1.4.5 utvecklades inte ett flerspelarläge. Ett sådant skulle dock göra spelet till en social aktivitet, vilket kan vara ett enkelt sätt att förlänga spelets livlängd. Sammantaget skulle de nämnda tilläggen göra att spelet riktar sig till en större marknad och får därmed större kommersiell potential.

Eftersom spelet bara är en prototyp behöver mer tid läggas på att hantera alla de problem som upptäcktes i utvärderingsfasen, vilket skulle innefatta att förbättra spelets GUI och utöka spelinnehållet. Sett till spelets brister är det relevant att ställa frågan om det är värt att arbeta vidare på spelet eller om det är en bättre idé att utveckla en ny version från grunden.

6.7 Spel och samhälle

Datorspel är en digital produkt som idag, genom tjänster som Steam [47], kan spridas till en väldigt stor marknad till låga miljökostnader för varje såld enhet. Som en produkt på marknaden finns det en potentiell stor ekonomisk vinst om tillräckligt många finner produkten intressant nog att köpa.

Då spelet är utvecklat som ett enspelarspel kommer de sociala aspekterna begränsas till diskussion gällande olika spelstilar och strategi. Vore ett flerspelarläge implementerat skulle spelarna kunna uppleva spelet tillsammans, i likhet med sällskapsspel. Om spelets rollspelsmoment utvecklas vidare finns möjlighet till diskussion kring spelets handling och karaktärer. Spelet är inte utvecklat på ett sätt som innebär att det skulle kunna spelas professionellt, utan är gjort för att spelas i icke tävlingsinriktade sammanhang.

7. Slutsats

Syftet med projektet var att utveckla ett spel som kombinerar realtidsstrategi med turbaserad strategi. Resultatet är en prototyp som uppfyller syftet till den mån att det visar att konceptet fungerar. De element som representerar kombinationen är inte nog fullbordade för att spelet ska anses som färdigutvecklat. Den ursprungliga idén om hur kombinationen skulle fungera följdes genom hela projektet utan större modifikation. Det märktes dock att designen har brister som behöver åtgärdas för att spelet ska kunna skapa ett större intresse hos potentiella spelare. Responsen från testpersonerna var till stor del positiv, och ytterligare vidareutveckling är motiverad för att utforska kombinationens fulla potential.

Referenser

[1] Atlassian, "Git tutorial - branches." https://www.atlassian.com/git/tutorials/using-branches/d, 2015. [Online; accessed 2-June-2015].

[2] I. S. Nintendo, "Fire emblem awakening." http://fireemblem.nintendo.com/, 2012. [Online; accessed 31-May-2015].

[3] A. Adams, Ernest, Rollings, *Game Design and Development: Fundamentals of Game Design*, ch. Strategy Games, pp. 468–501. Pearson Prentice Hall, 2007.

[4] B. Entertainment, "Starcraft II." http://us.battle.net/sc2/en/, 2015. [Online; accessed 31-May-2015].

[5] B. I. S. Bioware, "Baldur's gate." http://www.baldursgate.com/, 1998. [Online; accessed 31-May-2015].

[6] B. I. Studios, "Planescape: Torment." [CD-ROM], 1999.

[7] A. Adams, Ernest, Rollings, *Game Design and Development: Fundamentals of Game Design*, ch. Roleplaying Games, pp. 507–535. Pearson Prentice Hall, 2007.

[8] S. Johnson, "Analysis: Turn-based versus real-time." http://www.gamasutra.com/view/news/116864/Analysis_TurnBased_Versus_RealTime.php, 2009. [Online; accessed 18-May-2015].

[9] Total War Wiki, "Total War series." http://wiki.totalwar.com/w/Main_Page, 2015. [Online; accessed 31-May-2015].

[10] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5, pp. 99–108, 2001.

[11] D. Jamieson, "Making ai fun: When good enough is good enough." http://gamedevelopment.tutsplus.com/articles/making-ai-fun-when-good-enough-is-good-enough--cms-23460, 2015. [Online; accessed 18-May-2015].

[12] git scm, "About git." https://git-scm.com/about/, 2015. [Online; accessed 31-May-2015].

[13] git scm, "About git: Distributed." https://git-scm.com/about/distributed, 2015. [Online; accessed 31-May-2015].

[14] G. Inc, "About." https://github.com/about, 2015. [Online; accessed 31-May-2015].

[15] Atlassian, "Git and mercurial code management for teams." https://bitbucket.org/, 2015. [Online; accessed 31-May-2015].

[16] G. Inc, "Guide - github education." https://education.github.com/guide/private_repos, 2015. [Online; accessed 31-May-2015].

[17] L. J. G. L. Project, "Lwjgl." http://www.lwjgl.org/, 2015. [Online; accessed 31-May-2015].

[18] Slick2D, "Slick2d." http://slick.ninjacave.com/, 2015. [Online; accessed 31-May-2015].

[19] B. Games, "Features." http://libgdx.badlogicgames.com/features.html, 2015. [Online; accessed 31-May-2015].

[20] O. Corporation, "Learn about java technology." java.com/en/about/, 2015. [Online; accessed 12-May-2015].

[21] dynatrace, "How garbage collection works." http://www.dynatrace.com/en/javabook/how-garbage-collection-works.html, 2015. [Online; accessed 31-May-2015].

[22] B. Games, "Standing on the shoulders of giants." https://github.com/libgdx/libgdx/wiki/Introduction, 2015. [Online; accessed 12-May-2015].

[23] B. Games, "License." https://github.com/libGDX/libGDX, 2015. [Online; accessed 12-May-2015].

[24] B. Games, "Goals and features." libgdx.badlogicgames.com/features.html, 2015. [Online; accessed 12-May-2015].

[25] B. Games, "Texturepacker." https://github.com/libgdx/libgdx/wiki/Texture-packer, 2015. [Online; accessed 12-May-2015].

[26] T. Lindeijer, "Tiled map editor." http://www.mapeditor.org/, 2015. [Online; accessed 31-May-2015].

[27] B. Games, "Tile maps." https://github.com/libgdx/libgdx/wiki/Tile-maps, 2015. [Online; accessed 31-May-2015].

[28] T. Lindeijer, "Best practices." https://github.com/bjorn/tiled/wiki/Best-Practices#The_Tileset, 2015. [Online; accessed 31-May-2015].

[29] T. Inc, "Trello." https://trello.com/about, 2015. [Online; accessed 31-May-2015].

[30] T. Inc, "Trello tour." https://trello.com/tour, 2015. [Online; accessed 31-May-2015].

[31] W. Model, "Waterfall model." http://www.waterfall-model.com/, 2015. [Online; accessed 18-May-2015].

[32] K. Schwaber, "Scrum development process," in *Business Object Design and Implementation*, pp. 117–134, Springer, 1997.

[33] M. James, "Scrum sprint." http://scrummethodology.com/scrum-sprint/, 2015. [Online; accessed 31-May-2015].

[34] M. James, "Scrum backlog." http://scrummethodology.com/the-scrum-backlog/, 2015. [Online; accessed 31-May-2015].

[35] M. James, "The scrummaster role." http://scrummethodology.com/the-scrummaster-role/, 2015. [Online; accessed 31-May-2015].

[36] B. Alan, *Samhällsvetenskapliga metoder*, ch. Kvalitativa intervjuer, pp. 299–323. Elanders Hungary, 2002.

[37] B. Alan, *Samhällsvetenskapliga metoder*, ch. Kvalitativa forskning, pp. 249–275. Elanders Hungary, 2002.

[38] M. Sicart, "Defining game mechanics." http://gamestudies.org/0802/articles/sicart, 2015. [Online; accessed 01-June-2015].

[39] M. James, "Scrum sprint planning meeting." http://scrummethodology.com/scrum-meetings/, 2015. [Online; accessed 31-May-2015].

[40] G. Inc, "Mastering issues." https://guides.github.com/features/issues/, 2015. [Online; accessed 31-May-2015].

[41] G. Inc, "About repository graphs." https://help.github.com/articles/about-repository-graphs/, 2015. [Online; accessed 31-May-2015].

[42] B. Games, "Scene2d." https://github.com/libgdx/libgdx/wiki/Scene2d, 2015. [Online; accessed 31-May-2015].

[43] B. Games, "Scene2d.ui." https://github.com/libgdx/libgdx/wiki/Scene2d.ui, 2015. [Online; accessed 31-May-2015].

[44] B. Games, "gdxai." https://github.com/libgdx/gdx-ai/wiki, 2015. [Online; accessed 31-May-2015].

[45] B. Games, "Actor (libgdx api)." http://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/scenes/scene2d/Actor.html, 2015. [Online; accessed 31-May-2015].

[46] M. West, "Measuring responsiveness in video games." http://www.gamasutra.com/view/feature/3725/measuring_responsiveness_in_video_.php, 2015. [Online; accessed 31-May-2015].

[47] Steam, "about." http://store.steampowered.com/about/, 2015. [Online; accessed 18-May-2015].

A. Frågeformulär

Ålder

<18 18-21 21-25 25-30 30-40 40<

Svaren besvaras genom att ringa in en siffra mellan 0-5 där 0 är väldigt negativ och 5 är väldigt positiv.

TItelskärm

Hur begriplig anser du att titelskärmen är?

0 1 2 3 4 5

Kommentar_____
_____
_____

Hur begriplig anser du att options är?

0 1 2 3 4 5

Kommentar_____
_____
_____

Hur begriplig anser du att hjälpmenyn är?

0 1 2 3 4 5

Kommentar_____
_____
_____

Hur anser du att hjälpmenyn påverkade din förståelse av spelet?

0 1 2 3 4 5

Kommentar_____
_____
_____

B. Sammanfattning av intervjuer

Användartestning

Uppgifter vid spelsession

Titelskärm

- Gå in i inställningsmeny
- Ändra ljudvolymen
- Backa till huvudmenyn
- Gå in i Mapselection
- Ändra Spelare till AI och sedan tillbaka till spelare
- Starta spel

Spelläge

- Zooma ut
- Markera en karaktär
- Flytta karaktären
- Markera en annan karaktär
- Attackera en fiende
- Ändra stance
- Öppna spelets huvudmenyn
- Starta spelet igen
- Pausa spelet ingame
- Använd en Ability
- Ändra behavior
- Starta spelet igen

Frågor under Intervju

- Hur upplever du att spelet hanterar kombinationen av turbaserat och realtid?
- Hur lätt var det att förstå hur spelet fungerar?
- Anser du att du fick tillräcklig feedback av dina handlingar?
- Hur upplever du spelets GUI?
- Hur roligt finner du spelet?

Sammanfattning av spontana reaktioner och intervjuer

Antal speltestare: 11

Spontana reaktioner och svar från ostrukturerade intervjufrågor

- Rutorna är för små och det är svårt att klicka på dem
- Sniglarna är för hårda
- Options behöver fixas till
- Fonten är otydlig
- Namn på karaktärer istället för nummer i hjälpmenyn
- Vill ha bildexempel i hjälpmenyn
- Jobbigt att kontrollera alla enheter samtidigt. Vill ha möjlighet att kontrollera alla samtidigt
- Pathfindingen är dum, karaktärer fastnar i varandra
- Tooltips för abilitites behöver göras tydligare
- Klickar mycket för att få enheter att röra på sig. Efter spelets turbaserade moment upptäcks minskar klickandet.
- Tar ett tag innan spelet uppfattas som turbaserat.
- Spelets grafiska stil är omtyckt.
- Svårt att flytta kartan med musen.
- Behaviors används inte, de insåg inte att de fanns föränn de nämdes av testledaren
- Stridsstilar används inte, de insåg inte att de fanns föränn de nämndes av testledaren
- Pausfunktionen antingen ignoreras eller används flitigt beroende på spelare
- Använder en eller två enheter. Det är jobbigt att flytta runt alla det blir bättre efter de upptäcker att de kan flytta utanför deras färgade rörelse-rutor
- Försöker använda vänstermusknapp till allt. Kontrollschemat verkar inte vara lättförståligt
- Enheterna blir blockade an gravstenar. Det borde inte hända
- Enheter skjuter över väggar. Det verkar konstigt.
- Enheter skjuter fiender som de inte ser.
- Sniglarna blev populära.

Intervjuer

Antal intervjuer: 11

Sammanfattning av deras svar från de strukturerade frågorna

- Hur upplever du att spelet hanterar kombinationen av turbaserat och realtid?
    - Det var jobbigt att vänta på att enheterna gör saker (Det blev bättre då turerna gick snabbare)
    - Pathfindingen var jobbig då de gick åt fel håll

    - Blev roligare då de förstod spelmekaniken
- Hur lätt var det att förstå hur spelet fungerar?
    - Det tog ett tag att förstå hur man flyttar enheterna och använder förmågorna
    - Tog ett tag innan de förstod att det var turbaserat
    - Hittade inte stridsstilar och behaviors
    - Att flytta kartan var jobbigt med musen
    - Förstod inte vad alla abilities gjorde
- Anser du att du fick tillräcklig feedback av dina handlingar?
    - Feedbacken som ges är inte relevant då det är turbaserat.
    - Behöver mer indikation på att karaktärer är redo för att agera nästa runda.
    - Abilites behöver mer feedback. Frost armor fick mest kritik
    - Turklockan behöver göras tydligare
    - Pilarna för att flytta karaktärer var väldigt uppskattat
    - Ljud och animationer i strid var passande
- Hur upplever du spelets GUI?
    - Det behöver göras tydligare
    - Ni gömmer saker i skärmens hörn
    - Behövs tydligare tooltips
    - Det blockerar kartan i toppen av skärmen. Det är jobbigt.
- Hur roligt finner du spelet?
    - Det var väldigt roligt att spela även om det inte innehöll mycket
    - Kan tänka sig att köpa det om det utvecklas vidare
    - Gillar inte turbaserat utan men gillar realtidsstrategi, men fann denna blandning rolig
    - Det är väldigt lätt, fienderna gör ingen skada men de dör inte heller. Hade varit roligare om det var svårare.
    - Känner att det inte finns någon strategi i spelet.